Chapter 20
The Kinetic 3D Voronoi Diagram: A Tool for Simulating Environmental Processes

Hugo Ledoux

Abstract

Simulations of environmental processes are usually modelled by partial differential equations that are approximated with numerical methods, based on regular grids. An attractive alternative for simulating a fluid flow is the Free-Lagrange Method (FLM). In this paper, I discuss the use of the FLM—based on the Voronoi diagram (VD)—for the modelling of fluid flow in three dimensions (e.g. the movement of underground water or of pollution plumes in the ocean). Such a technique requires the *kinetic* three-dimensional VD, which is a VD for which the points are allowed to move freely in space. I present a new algorithm for the movement of points in a three-dimensional VD, and show that it can be relatively easy to implement as it is the extension of a simple two-dimensional algorithm.

20.1 Introduction

The integration of simulation models and geographical information systems (GISs) is a major source of problems because GISs have not been designed for handling time, and even less for handling processes which involves continual movements [50]. Full integration is almost unheard of, and the two are usually simply 'linked', i.e. a GIS is used as a pre-processing tool (e.g. to prepare a dataset or convert formats) and as a post-processing tool (e.g. visualisation and further spatial analysis), althoug the simulation itself is done using a specialised tool. Many argue for 'full integration', as both tools would ultimately gain [19, 43]. The simulation of some environmental processes, e.g. the track-

Delft University of Technology (OTB—section GIS Technology)
Jaffalaan 9, 2628 BX Delft, the Netherlands
h.ledoux@tudelft.nl

ing of pollution plumes in the ocean or dispersion models in meteorology, is even more problematic because these phenomena are three-dimensional by nature, and three-dimensional GISs are still their infancy (see Zlatanova et al. [52] and Raper [44] for discussions). Simply to be integrated into a GIS, the results of environmental simulations must often be 'sliced' into several 2D datasets [8, 41].

Many disciplines require simulations of real-world processes, and the methods they use obviously differ. Simulations in geoscience or in engineering have usually been based on partial differential equations (PDEs) that describe the behaviour of some fluid (e.g. the fluid flow around an aircraft, or the movement of underground water) [10]. PDEs are solved, or rather approximated, by mainly two numerical methods: the *finite difference method* (FDM), which was developed for regular tessellations; and the *finite element method* (FEM) [48], which is possible on any tessellations, regular or irregular. The solution of a PDE is obtained by first approximating the behaviour of the process studied for each element of the tessellation, and the final solution is obtained by accumulating all the results. The FDM performed on grids, as used for instance by systems for weather forecasting, is well-known, efficient and mostly accurate. However, the use of grids can sometimes lead to unreliable results [5], and some other technical problems also arise (for instance the curvature of the Earth is problematic for large datasets).

For the modelling of fluid flow, an alternative approach to using a fixed/rigid tessellation is the *Free-Lagrange method* (FLM) [20]. With this method, the flow is approximated by a set of discrete points (called particles) that are allowed to move freely and interact, with each particle having a mass and a velocity. A tessellation of the space is still required, although this will be modified as the particles are moving. As briefly explained in Sect. 20.2, the Voronoi diagram 'naturally' tessellates the space based on a set of points, and has therefore been used (see Erlebacher [16] for instance). Although the FLM can theoretically better represent a physical process [20], it is handicapped by the many difficulties encountered when implementing it. As Mostafavi and Gold [40] note, the adjacency relationships of the cells of the tessellation must be kept consistent at all times, and there must be a way to model time, because fixed time steps can comprise the adjacency of the tessellation. Indeed, earlier implementations of the FLM were very slow as the adjacency relationships between cells had to be rebuilt at each step of the process. With Mostafavi and Gold's solution, the *kinetic* VD, all the topological events are managed locally, and the time steps that were previously used (which could lead to overshoots and unwanted collisions) can be avoided as topological events are used. They further show the advantages of the kinetic VD with the simulation of global tides on the Earth (thus using the VD on a sphere).

In this paper, I extend the work presented in Mostafavi and Gold [40] to three dimensions, and present a novel algorithm for keeping a 3D VD up-to-date as the points defining it are moving over time. In other words, an algorithm for the kinetic VD in three-dimensional space is presented. Such

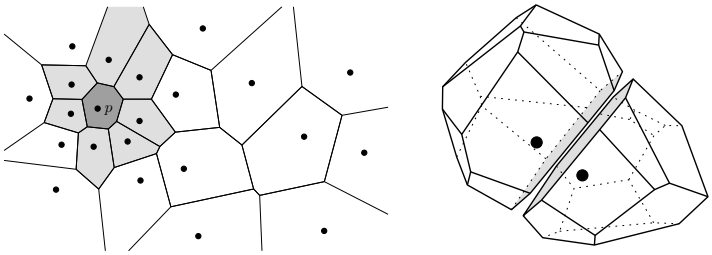


Fig. 20.1 **(a)** VD of a set of points in the plane. The point p (whose Voronoi cell is dark grey) has seven neighbouring cells (light grey). **(b)** Two Voronoi cells adjacent to each other in $\mathbb{R}^3$ (they share the grey face).

an algorithm is interesting for two reasons: (i) it permits us to perform simulation with the VD (which potentially yields more accurate results); and (ii) it is a step in the direction of integrating simulation models and GIS (that is, if the VD is used as an alternative spatial model to the usual point-line-polygon model, as in the work of Gold [25], Gahegan and Lee [21] and Chen et al. [11]). Because the paper is fairly technical, important concepts related to the VD are first introduced in Sect. 20.2, and then the algorithm itself is presented in Sect. 20.3, along with a literature review of other potential methods. Sections 20.4 and 20.5 discuss the potential applications of the algorithm presented, and also briefly discuss its implementation. Notice that most of the concepts and methods discussed are firstly introduced by describing their two-dimensional counterparts (because readers are often more familiar with these), and that most figures are for the 2D case, as they are much simpler to understand.

20.2 Voronoi Diagram & Related Issues

Let S be a set of n points in an n-dimensional Euclidean space $\mathbb{R}^n$. The Voronoi cell of a point $p \in S$, defined $\mathcal{V}_p$, is the set of points $x \in \mathbb{R}^n$ that are closer to p than to any other point in S. The union of the Voronoi cells of all generating points $p \in S$ form the Voronoi diagram of S, defined VD(S). Fig. 20.1 shows two- and three-dimensional examples. The VD is arguably one of the most important geometric/spatial structures in sciences because it is very simple, and yet is so powerful that it helps in solving many theoretical problems, and also helps in many real-world applications; Aurenhammer [6] and Okabe et al. [42] offer exhaustive surveys.

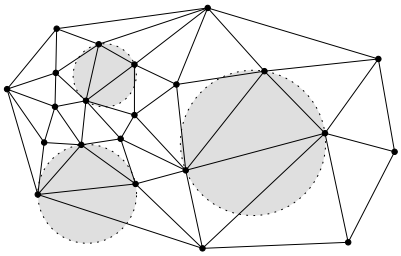


Fig. 20.2 DT in 2D of the same set of points as in Fig. 20.1(a).

The Delaunay triangulation.

The VD is closely related to another structure: the Delaunay triangulation (DT). The DT is popular in 2D in many application domains because it has many useful properties, among others are the fact that the triangles created are as equilateral as possible, and that it can be modified locally. In 3D, the Delaunay *tetrahedralization* is defined by the partitioning of the space into tetrahedra—where the vertices of the triangles are the points in S (generating each Voronoi cell)—that satisfy the *empty circumsphere* test (a sphere is empty is no points are in its interior, but points can lie directly on the sphere). Fig. 20.2 illustrates the idea in 2D.

Duality.

The VD and the DT are dual structures, which means that the knowledge of one implies the knowledge of the other one. In other words, if one has only one structure, he can always extract the other one. The concept of duality is important for the construction, the manipulation and the storage of the VD, because all the operations can be performed on its dual, and when needed, the VD extracted. The algorithm in Sect. 20.3 uses this idea.

General position.

An important concept when discussing the VD and the DT is that of the position of the points in a set of points. A set S of points is said to be in *general position* when the distribution of its points does not create any ambiguity in the structures derived from the points (e.g. the VD or the DT). For the VD and/or the DT in $\mathbb{R}^d$, the degeneracies, or special cases, occur when $d+1$ points lie on the same hyperplane and/or when $d+2$ points lie on the same ball. For example, in three dimensions, when five or more points in S are *cospherical* there is an ambiguity in the definition of DT(S): since all the points lie on a sphere, all the tetrahedralizations of the points respect the Delaunay criterion. When four or more points are coplanar in 3D, DT(S)

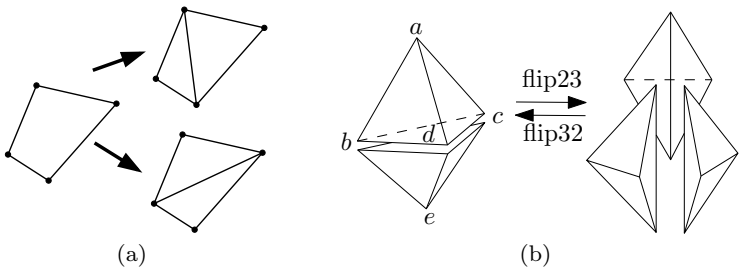


Fig. 20.3 Flips in **(a)** two dimensions, and **(b)** three dimensions.

and VD(S) are unique, but problems with the computation of the structures can arise (see for instance Sugihara and Inagaki [49] and Field [18]).

20.2.1 Construction of the VD/DT

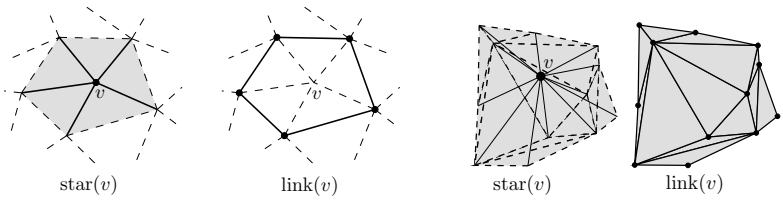
The construction of a VD, or a DT, is a well-known problem in computational geometry and different efficient algorithms, based on different computational geometry paradigms, are available (see for instance Edelsbrunner and Shah [15], Watson [51] or Cignoni et al. [12]).

Predicates.

An important consideration is that for all the construction algorithms, essentially only the following two basic geometric tests (called *predicates*) are required: ORIENT determines if a given point is over, under or lies on a plane defined by three points; and INSPHERE determines if a given point is inside, outside or lies on a sphere defined by four points. Both tests can be reduced to the computation of the determinant of a matrix, see Guibas and Stolfi [30] for more details.

20.2.2 Flips

A flip is a local topological operation that modifies the configuration of some adjacent tetrahedra [34, 15]. The concept of flip is valid in any dimensions for triangulations. In 2D, many algorithms to construct and modify a triangulation use the *flip22*, as illustrated in Fig. 3(a). It permits us to transform a triangulation of four points into the only other one possible. In 3D, the same idea can be applied to a set $S = \{a,b,c,d,e\}$ of five points in general position. According to Lawson [34], there are exactly two ways to tetrahedralize such



(a) In 2D. (b) In 3D.

Fig. 20.4 The star and the link of a vertex v.

a polyhedron: either with two or three tetrahedra. As illustrated in Fig. 3(b), in the first case, the two tetrahedra share a triangular face *bcd*, and in the latter case the three tetrahedra all have a common edge *ae*. A *flip23* is the operation that transforms one tetrahedralization of two tetrahedra into another one with three tetrahedra; and a *flip32* is simply the inverse operation. Notice that the numbers refer to the number of triangles/tetrahedra before and after the flip.

It is worth noticing that three-dimensional flips do not always apply to adjacent tetrahedra [32]. For example, in Fig. 3(b), a flip23 is possible on the two adjacent tetrahedra *abcd* and *bcde* if and only if the line *ae* passes through the triangular face *bcd* (which also means that the union of *abcd* and *bcde* is a convex polyhedron). If not, then a flip32 is possible if and only if there exists in the tetrahedralization a third tetrahedron adjacent to both *abcd* and *bcde*.

20.2.3 Star, Link and Ears

Three concepts related to triangulations are introduced here, and they will be used in Sect. 20.3 where the proposed algorithm is described.

Star.

Let v be a vertex in a d-dimensional triangulation. Referring to Fig. 20.4, 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.

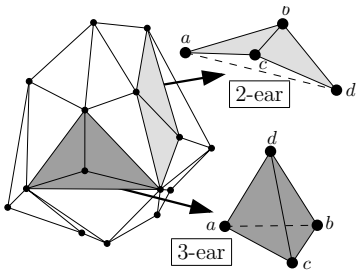


Fig. 20.5 Perspective view of the outside of a polyhedron. Two adjacent triangular faces (e.g. in light grey) form a 2-ear, and three triangular faces incident to the same vertex (e.g. in dark grey) form a 3-ear.

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 $(d-1)$ triangulation. For example, if v is a vertex in a tetrahedralization, then link(v) is a two-dimensional triangulation formed by the vertices, edges and triangular faces that are contained by the tetrahedra of star(v), but are disjoint from v.

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 Fig. 20.5, such a tetrahedron—that does not exist yet—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 note that not every ear is a potential tetrahedron to tetrahedralize P, as some adjacent faces form a tetrahedron lying outside P. Referring again to Fig. 20.5, 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.

20.3 Moving Points in a VD/DT

It should first be said that when a point in a VD/DT is continually moving over time and if one is interested in every intermediate state of the VD/DT, it makes no sense to continually insert, delete and reinsert it again somewhere else, as this is a computationally expensive operation. A more efficient

option is to literally *move* the point and update the topological relationships of the VD/DT when needed. In other words, instead of using 'discrete updates', 'continuous updates' to the VD/DT are made. Discrete updates are nevertheless an adequate solution for many applications where points move a lot and where the intermediate states are not important (just the start and end states are of interest). For example, De Fabritiis and Coveney [13] use a combination of discrete and continuous updates (depending on the situation) for the simulation of fluids. Similarly, for the simulation of physical processes (where molecules are moving only by very small distances), Guibas and Russel [29] found that continuous updates permit them to update the VD/DT in approximately half to three quarters the time it takes to recompute the entire structure.

The algorithms to maintain a VD/DT of a set S of points up-to-date as one or more points in S are moving are based on the following observation.

Observation 1 *Let $\mathcal{T}$ be the DT(S) of a set S of points in $\mathbb{R}^d$ in general position. If one point p is moved by a sufficiently small amount so that S stays in general position at all times, then the combinatorial structures of DT(S) (and of VD(S)) will not change (see Fig. 20.6).*

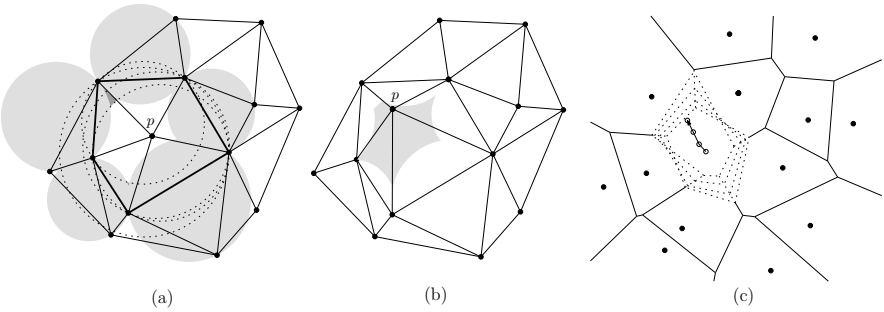


Fig. 20.6 **(a)** The combinatorial structure of the DT will not change as long as the vertex p is moved within the white polygon. **(b)** p has moved but S is still in general position. The combinatorial structure of DT(S) has not changed. Only the location of p has changed, and so have the edges incident to it. **(c)** Example of the consequences of moving p on the VD.

Notice that S will remain in general position until p is cospherical (lies on the same ball in $\mathbb{R}^d$) with $d+1$ other points in S. At the critical moments when the loss of general position arises, the topological structures of VD(S) and DT(S) will be modified; the critical moments are called *topological events*. Observe in Fig. 20.6(c) that the VD where one point is moving looks different as the point is moving, but that the adjacency relationship of the Voronoi cell of the moving point remain the same.

The result of Observation 1 is that in order to move one or more points in S, one has to detect when the topological events will arise, and modify VD(S) and/or DT(S) consequently.

20.3.1 Related Work in Two Dimensions

Observation 1 was used by Roos [45] to analyse the complexity of the movement of points in a two-dimensional VD, and proposed an algorithm to update the VD. Although he discusses the movement of points in a VD, the algorithm he developed is based on the dual (because it is simpler and because the VD can be trivially extracted). When a topological event arises, the update to the DT is made with a flip. He considers that all (or most) of the points in S are moving according to a linear trajectory and that they have a constant velocity. His algorithm starts by computing DT(S), then all the potential topological events for all the quadrilaterals (every pair of adjacent triangles in DT(S) is tested) are computed and put in a priority queue (e.g. a balanced search tree), sorted according to the time they will arise. The time is computed by finding the zeros of the InCircle (two-dimensional counterpart of InSphere, as briefly discussed in Sect. 20.2.1) developed into a polynomial; in other words, the aim is to find when the four points forming a quadrilateral will become cocircular (if ever). After that, the first topological event is popped from the queue, DT(S) 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. The algorithm is efficient as only $\mathcal{O}(\log n)$ is needed for each topological event (n being the number of points in the set).

Similar algorithms have also been proposed, see for instance Bajaj and Bouma [7] and Imai et al. [31]. Moreover, Gold [23] and Gold et al. [26] (with more details in Mostafavi [38] and Mostafavi and Gold [40]) propose a different algorithm and give more implementation details. They focus on the operations necessary to move a single point p, and then explain how to have many points move. To detect topological events, only the triangles inside star(p) and the ones incident to the edges of link(p) need to be tested, and the changes in the DT are also made with flip22 operations.

20.3.2 Related Work in Three Dimensions

The work of Roos [45] has been generalised to three- and higher-dimensional space by Albers et al. [1, 3, 2]. Their work is mostly theoretical as they aim to find upper bounds on the number of topological events when the points are moving according to some trajectories. They state that only the two-

dimensional case has been implemented, and they demonstrate that in three dimensions the flip23 and flip32 can be used to update the DT. Gavrilova and Rokne [22] discuss the movement of d-dimensional balls (not only points, but balls with defined radii) while the *additively weighted* Voronoi diagram (or Apollonius diagram) is maintained up-to-date with flips; the operations are performed on the dual of the Apollonius diagram. Their algorithm is exactly the same as the one used by Albers et al., but they show how the InSphere test must be modified to consider the radius of each ball.

The major impediment to the implementation of the algorithm used by Albers et al. in three dimensions is that, as Gavrilova and Rokne [22] observe, calculating the zeros of the function InSphere cannot be done analytically, as is the case for the InCircle function. Indeed, the polynomial for the three-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. On the other hand, Guibas et al. [28] recently proposed a generic framework for handling moving objects. The methods they use for the kinetic 3D VD/DT is theoretically the same as in Roos [45], although they use different methods for finding the zeros of polynomials (InSphere) using fixed precision and exact arithmetic, and they claim that 3D VDs/DTs can be updated relatively fast in most cases.

It appears that the computational geometry community is more interested in studying the complexity of the problem than implementing it. To my knowledge, the only reports of implementations are that of Guibas and Russel [29] and Guibas et al. [28], whose algorithm has recently been added to `CGAL`[1], and some reports in related disciplines where there is a real need. For instance, Ferrez [17] and Schaller and Meyer-Hermann [46] did practical implementations of the algorithm for respectively the simulation of granular materials and cell tissues. Ferrez's algorithm is for spheres in Laguerre space (power distance is used), and thus the regular tetrahedralization is built; this is almost the same as the DT, for only the InSphere test has to be modified slightly. Both seem to have missed out several theoretical issues, e.g. they do not consider Observation 1, and use 'time steps' to move a point. In other words, a point is simply moved to a certain location without first verifying if topological events will arise. Flips are performed after each move to restore the Delaunay criterion, and their only constraints is that the combinatorial structures must stay valid between two steps, i.e. a point is not allowed to penetrate another tetrahedron. While this may work for some cases, defining a time step that works for all cases is impossible, and they do not consider the fact that unlike in two dimensions, it is sometimes impossible to flip adjacent tetrahedra. Their solution could therefore not work for every cases, and more importantly, their tetrahedralization does not respect the Delaunay criterion at all times, which could be problematic for some applications.

[1] The Computational Geometry Algorithms Library (`www.cgal.org`).

20.3.3 A Flip-based Algorithm

Here I discuss a new algorithm used to move points in $\mathbb{R}^3$ and update the VD/DT when topological events arise. Since the implementation of Albers et al.'s algorithm is intricate, I describe a generalisation to three dimensions of Gold and Mostafavi's method [23, 26, 38, 40]. Unlike Albers et al.'s method where all the pairs of simplices must be tested, the algorithm I present permits the movement of one or only a few points in the set S by using only local information, i.e. if p is moved, only the geometry of the neighbouring tetrahedra of p will be used, and tetrahedra not in the neighbourhood of p or the trajectory do not need to be tested. Moreover, it is not necessary to find the zeros of the function InSphere because the topological events are detected by testing the intersections between the circumsphere of neighbouring tetrahedra and the trajectory of p.

The different types of tetrahedra that must be considered are first discussed, then the algorithm to move a single point is presented, and finally the movement of several points in S is discussed. The concepts described are direct generalisations of the algorithm of Gold and Mostafavi, and I describe the intricacies that one more dimension brings. The algorithm is based on the same operations that are necessary for constructing a VD, the flips, and is thus conceptually very simple and easy to implement.

20.3.3.1 Types of tetrahedra

Three types of tetrahedra, with respect to the moving point p, must be defined (see Fig. 20.7(a) for an example in the plane):

Real tetrahedra: are the tetrahedra τ_i that are incident to the faces of link(p), but outside star(p).

Imaginary tetrahedra: are the ears σ_i of star(p), as defined in Section 20.2.3. They are imaginary because they do not exist yet, but some would exist if p was removed or moved somewhere else. Remember that 2-ears and 3-ears can exist.

Behind tetrahedra: are real tetrahedra that are 'behind' p and its trajectory. In theory, they are not mandatory, but in practice they permit us to test fewer tetrahedra (for the intersection with the trajectory), and not to retest tetrahedra that have been previously tested. The criterion for a real tetrahedron τ to be a behind tetrahedron is if the orthogonal projection of the centre of its circumsphere, denoted sphere(τ), onto the trajectory falls behind p, see Fig. 20.7(a).

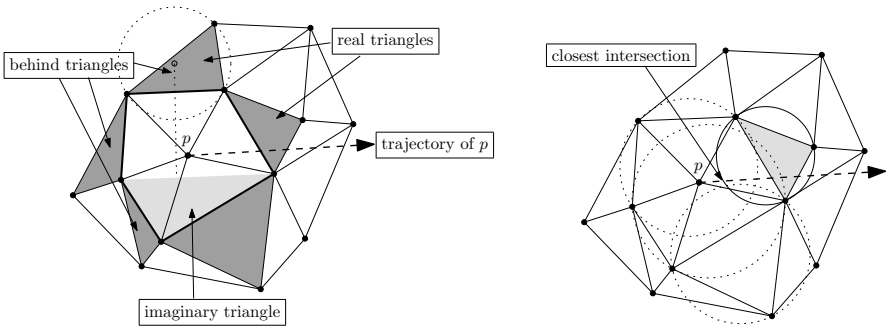


Fig. 20.7 (a) The different types of triangles needed to move the vertex p along the trajectory. Real triangles are the dark shaded ones, and one example of an imaginary triangle is light shaded. Notice that a behind triangle is always also a real triangle. **(b)** p must be moved to the closest intersection of a circumcircle along the trajectory. The triangle having the closest intersection is the shaded triangle (a real triangle).

20.3.3.2 The algorithm

The general idea of the algorithm is that, given the moving point p and its final destination x, we must move p step-by-step to the closest topological event, perform a flip, and then do these two operations again until p reaches the location x. As shown in Fig. 20.7(b), the closest topological event is the location along the trajectory (the line segment $\overline{px}$) where the intersection between $\overline{px}$ and the circumspheres of the real tetrahedra of p and the imaginary tetrahedra of p (only the valid ears are tested) is the closest. Observe that there are two possible cases (this is illustrated in Fig. 20.8):

(1) p is 'moving in' the circumsphere of a real tetrahedron;
(2) p is 'moving out' of the circumsphere of an imaginary tetrahedron.

The new algorithm I present, MoveOnePoint (Fig. 20.3.3.2), is for moving a single point in a set S (while all the other ones are fixed). It is assumed that S is in general position. MoveOnePoint start by initialising a list, denoted B, containing all the behind tetrahedra of p. B is built by checking if the orthogonal projection of every centre of sphere(τ_i) falls 'before' the trajectory $\overline{px}$; if it is then τ_i is added to B. Although it is not a necessity to store lists for the real and imaginary tetrahedra, it may be a good idea to built them at the beginning and simply update them as flips are performed. The lists are not necessary because at each step of the process the real and imaginary tetrahedra can be efficiently retrieved on the fly by simply identifying all the tetrahedra forming star(p); the data structure used to store the DT should permit that, but these are not discussed here due to space constraints.

In order to get the tetrahedron (real or imaginary) whose circumsphere has the closest intersection, a simple test that computes the intersection between

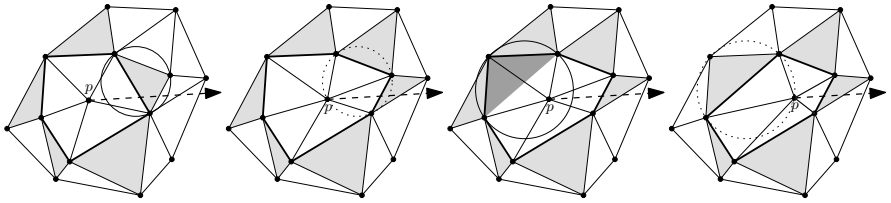


Fig. 20.8 Two cases are possible when p is moved along a trajectory. At all times, the real triangles are the light shaded ones. **(a)** The closest intersection is with a real triangle. **(b)** p is moved to the circumcircle, a flip22 is performed and the real triangles are updated. **(c)** The next closest intersection is with an imaginary triangle (dark shaded triangle). **(d)** p is moved to the circumcircle, a flip22 is performed and the real triangles are updated. Notice also that the behind triangles are updated, while they were not when p moved in a real triangle.

Input: DT(S) $\mathscr{T}$; the point p to move; final destination x
Output: $\mathscr{T}$ is modified: p is at location x

```
initialise B {let B be a simple dynamic list}
while p is not at location x do
    τ_l ← get tetra (real or imaginary) having closest intersection with trajectory
    move p to intersection
    if τ_l is a real tetrahedron then
        τ ← get tetrahedron inside star(p) adjacent to τ_l
        if τ ∪ τ_l is convex then
            flip23(τ, τ_l)
        else
            flip32(τ, τ_l, τ_2)
        end if
    else {τ_l is an imaginary tetrahedron}
        if τ_l is a 2-ear then
            remove from B the 2 tetrahedra outside star(p) incident to τ_l
            flip23(τ_l)
            add τ_l to B {the ear becomes a behind tetrahedron}
        else {τ_l is a 3-ear}
            remove from B the 3 tetrahedra outside star(p) incident to τ_l
            flip32(τ_l)
            add τ_l to B {the ear becomes a behind tetrahedron}
        end if
    end if
end while
```

Fig. 20.9 Algorithm MoveOnePoint($\mathscr{T}$, p, x)

a line segment and a sphere is used. As explained in Bourke [9], the idea is to start with the parametric equation of the line segment $\overline{px}$, such that at $t = 0$ we are at location p, and at $t = 1$ we are at location x. By substituting the equation of $\overline{px}$ with the equation of a sphere, we can get a quadratic equation that has no solution (no intersection), one solution (sphere is tangent to $\overline{px}$), or two solutions (the line intersects the sphere). We also know where along

the line segment the intersection(s) occur(s): if $t < 0$ then it is before p, and if $t > 0$ it is after x. Observe that when we are dealing with imaginary tetrahedra, the intersection with the highest value of t is to be considered as we are moving out of a sphere, while the smallest value of t is considered for real tetrahedra.

When S is in general position, the rest of the algorithm is straightforward. Indeed, Albers et al. [2] show that when p is moved to the closest topological event, then a flip (either flip23 or flip32) is always possible, i.e. there will not be unflippable cases.

An important point is that when p moves out of an imaginary tetrahedron, the list B must be updated. As shown in Fig. 20.8, the behind tetrahedra are modified when an ear is flipped, but not when p moves in the circumsphere of a real tetrahedron. In three dimensions, the ear σ flipped becomes a tetrahedron τ (spanned by the four vertices of σ) that must be added to B. The two or three neighbours of τ (depending if σ was a 2- or 3-ear) outside star(p) must also be deleted from B.

20.3.3.3 Moving several points

Let S be a set of points in $\mathbb{R}^3$, where several points are moving over time. Each point is moving according to a linear trajectory and has a velocity v; the time taken to reach its next topological event, called t_t, is therefore $t_t = d/v$, where d is the distance to the closest intersection between its trajectory and the circumspheres of the real and imaginary tetrahedra. As explained in Mostafavi and Gold [40], to ensure that at a given time t all the moving points are where they should be, a global time needs to be kept (e.g. since the start of the simulation at $t = 0$). If only t_t was considered, then a case where a point having many topological events close to each others would delay the movement of other points.

Three following types of time must therefore be considered: t is the current time (time elapsed since $t = 0$); t_t is the time to the next topological event; and t_p is the predicted time (global time) for a point to reach the next topological event. The three types are linked:

$$t_p = t + t_t \,. \tag{20.1}$$

To ensure that the points are moved in the correct order, i.e. in such a way that the combinatorial structure of the VD/DT stays valid, a priority queue containing the t_p^i of every moving point $i \in S$ is kept. At each step, the point i whose t_p^i is the earliest is popped from the queue and processed. After the movement of the point i, t_p^i must be recalculated: the updated t^i becomes t_p^i, and a new t_p^i must be computed with the new t_t^i. The types of time are depicted in Fig. 20.10 for the movement of two points a and b. Observe that a went through two topological events before reaching its current position,

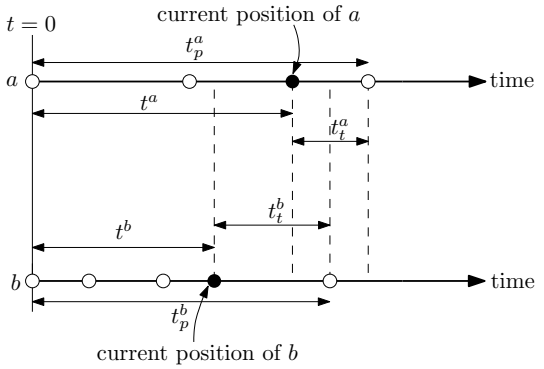


Fig. 20.10 The three types of time needed for moving several points at the same time. The next movement to be made is for point b, since t_p^b is before t_p^a. (Figure after Mostafavi and Gold [40])

and that b went through three. The total predicted time for point b is before that of point a, so b will be moved before a (although $t_t^a < t_t^b$).

After a point i has been moved, the t_p^j of all the points j that were 'influenced' by the movement must update their t_t^j (and thus their t_p^j). The movement of i modifies the shapes of all the tetrahedra incident to i, and since these can be the real tetrahedra of other points, the points j that form link(i), plus the points forming the link of these, must be updated.

20.4 Applications

The FLM based on the VD could obviously be used for simulation purposes in three dimensions, provided that we can formalise the physical forces applying to every location in space. Because the movement of points in a VD is rather computationally expensive (when all the points are moving simultaneously), the simulation of atmospheric or oceanographic phenomena on a large scale might not be the most suitable examples right now—we want to obtain the weather forecast for tomorrow, today! A representative example is the simulation of underground water, for instance for a city. Questions such as 'where does the ground water come from?', 'how does it travel?', and 'where do water contaminants come from, and where are they going?', can all be answered if we can adequately model the phenomenon. The work presented in this paper has already been used, by Dr Mostafavi at the Université Laval, Québec City, Canada [39], for the development of a prototype GIS modelling underground water. His team is currently working on defining the governing equations to obtain the vector and the velocity of every point in three dimensions, and it

is hoped that the kinetic VD will yield results that are more accurate than the ones with methods currently used.

20.5 Discussion

It should be noticed that the simulation of environmental processes is one of many of the potential applications of the kinetic three-dimensional Voronoi diagram. As briefly mentioned, it can be very useful in other disciplines, such as in engineering of physics. It is also potentially a tool for the interactive modelling of datasets [4], and one can think of an application where the user can 'play' at will with a dataset by adding, removing or moving objects.

The description of the algorithm MoveOnePoint, as presented in this paper, is not totally complete because the degenerate cases were not covered (it assumed that the set of points is in general position). Fixing an algorithm so that it is robust for all inputs/situations is usually a cumbersome and difficult task, especially for 3D geometric algorithms that are plagued with special cases [47]. The handling of degenerate cases (coplanar/cospherical points, collisions between points, etc.) was excluded due to of space constraints, but details and insights have been published previously by Ledoux [35].

It should also be stated that the algorithm presented offers one solution to the general problem of managing temporal data in a GIS. Indeed, if, as Gold has been advocating for years [24, 27], the VD is used to manage topological relationships between objects in a map, then we obtain a spatial model where insertion, deletion and movement of objects is possible *locally*, without the need to reconstruct from scratch the topological relationships when there is a modification to the map. This means that every operation is reversible. As shown in Gold [25] and Mioc [37], by simply keeping a 'log file' of every operation done it is possible to rebuild each 'topological state' of a VD, at any time. There is no need to keep various 'snapshots' of the data at different times for further analysis: when a representation at a specific time is required, it is reconstructed from the original data and from the log file. The work of Gold [25] and Mioc [37] was made for the VD in 2D, but, as shown in Sect. 20.2, the construction is also possible in 3D, and so is the deletion of vertices [14, 36]. At this moment, only algorithms for the VD of points in 3D have been implemented, but, as is the case for 2D [33], we can envision that in the forseeable future efficient algorithms for 3D VD for a set in which lines and surfaces are present will be available.

Acknowledgements

I would like to thank Christopher Gold and Maciej Dakowicz for several useful discussions concerning the kinetic Voronoi diagram in two dimensions.

References

[1] Albers G (1991) *Three-dimensional dynamic Voronoi diagrams* (in German). Ph.D. thesis, Universität Würzburg, Würzburg, Germany.

[2] Albers G, Guibas LJ, Mitchell JSB, 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 3rd Scandinavian Workshop On Algorithm Theory (SWAT'92)*, volume 621 of *Lecture Notes in Computer Science*, pages 399–409. Springler-Verlag, Helsinki, Finland.

[4] Anselin L (1999) Interactive techniques and exploratory spatial data analysis. In PA Longley, MF Goodchild, DJ Maguire, and DW Rhind, editors, *Geographical Information Systems*, pages 253–266. John Wiley & Sons, second edition.

[5] Augenbaum JM (1985) A Lagrangian method for the shallow water equations based on the Voronoi mesh-flows on a rotating sphere. In MJ Fritts, WP Crowley, and HE Trease, editors, *Free-Lagrange method*, volume 238, pages 54–87. Springer-Verlag, Berlin.

[6] Aurenhammer F (1991) Voronoi diagrams: A survey of a fundamental geometric data structure. ACM Computing Surveys, 23(3):345–405.

[7] Bajaj C and Bouma W (1990) Dynamic Voronoi diagrams and Delaunay triangulations. In *Proceedings 2nd Annual Canadian Conference on Computational Geometry*, pages 273–277. Ottawa, Canada.

[8] Bivand R and Lucas A (2000) Integrating models and geographical information systems. In S Openshaw and RJ Abrahardt, editors, *Geocomputation*, pages 331–363. Taylor & Francis, London.

[9] Bourke P (1992) Intersection of a line and a sphere (or circle). `http://astronomy.swin.edu.au/~pbourke/geometry/sphereline/`.

[10] Burrough PA, van Deursen W, and Heuvelink G (1988) Linking spatial process models and GIS: A marriage of convenience or a blossoming partnership? In *Proceedings GIS/LIS '88*, volume 2, pages 598–607. San Antonio, Texas, USA.

[11] Chen J, Zhao R, and Li Z (2004) Voronoi-based k-order neighbour relations for spatial analysis. ISPRS Journal of Photogrammetry & Remote Sensing, 59:60–72.

[12] Cignoni P, Montani C, and Scopigno R (1998) DeWall: A fast divide & conquer Delaunay triangulation algorithm in E^d. Computer-Aided Design, 30(5):333–341.

[13] De Fabritiis G and Coveney PV (2003) Dynamical geometry for multi-scale dissipative particle dynamics. Computer Physics Communications, 153:209–226.

[14] Devillers O (2002) On deletion in Delaunay triangulations. International Journal of Computational Geometry and Applications, 12(3):193–205.

[15] Edelsbrunner H and Shah NR (1996) Incremental topological flipping works for regular triangulations. Algorithmica, 15:223–241.

[16] Erlebacher G (1985) Finite difference operators on unstructured triangular meshes. In MJ Fritts, WP Crowley, and HE Trease, editors, *Free-Lagrange Method*, volume 238, pages 21–53. Springer-Verlag, Berlin.

[17] Ferrez JA (2001) *Dynamic triangulations for efficient 3D simulation of granular materials*. Ph.D. thesis, Département de Mathématiques, École Polytechnique Fédérale de Lausanne, Switzerland.

[18] Field DA (1986) Implementing Watson's algorithm in three dimensions. In *Proceedings 2nd Annual Symposium on Computational Geometry*, volume 246–259. ACM Press, Yorktown Heights, New York, USA.

[19] Freda K (1993) GIS and environment modeling. In MF Goodchild, BO Parks, and LT Steyaert, editors, *Environmental modeling with GIS*, pages 35–50. Oxford University Press, New York.

[20] Fritts MJ, Crowley WP, and Trease HE (1985) *The Free-Langrange method*, volume 238. Springler-Verlag, Berlin.

[21] Gahegan M and Lee I (2000) Data structures and algorithms to support interactive spatial analysis using dynamic Voronoi diagrams. Computers, Environment and Urban Systems, 24(6):509–537.

[22] Gavrilova ML 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.

[23] Gold CM (1990) Spatial data structures—the extension from one to two dimensions. In *Mapping and Spatial Modelling for Navigation*, volume 65, pages 11–39. Springer-Verlag, Berlin, Germany.

[24] Gold CM (1991) Problems with handling spatial data—the Voronoi approach. CISM Journal, 45(1):65–80.

[25] Gold CM (1996) An event-driven approach to spatio-temporal mapping. Geomatica, Journal of the Canadian Institute of Geomatics, 50(4):415–424.

[26] Gold CM, Remmele PR, and Roos T (1995) Voronoi diagrams of line segments made easy. In *Proceedings 7th Canadian Conference on Computational Geometry*, pages 223–228. Quebec City, Canada.

[27] Gold CM, Remmele PR, and Roos T (1997) Voronoi methods in GIS. In M van Kreveld, J Nievergelt, T Roos, and P Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 21–35. Springer-Verlag.

[28] Guibas L, Karaveles M, and Russel D (2004) A Computational Framework for Handling Motion. In *Proceedings 6th Workshop on Algorithm Engineering and Experiments*, pages 129–141. New Orleans, USA.

[29] Guibas L and Russel D (2004) An empirical comparison of techniques for updating Delaunay triangulations. In *Proceedings 20th Annual Symposium on Computational Geometry*, pages 170–179. ACM Press, Brooklyn, New York, USA.

[30] Guibas LJ and Stolfi J (1985) Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. ACM Transactions on Graphics, 4:74–123.

[31] Imai K, Sumino S, and Imai H (1989) Geometric fitting of two corresponding sets of points. In *Proceedings 5th Annual Symposium on Computational Geometry*, pages 266–275. ACM Press, Saarbrücken, West Germany.

[32] Joe B (1989) Three-dimensional triangulations from local transformations. SIAM Journal on Scientific and Statistical Computing, 10(4):718–741.

[33] Karavelas MI (2004) A robust and efficient implementation for the segment Voronoi diagram. In *International Symposium on Voronoi Diagrams in Science and Engineering*, pages 51–62. Tokyo, Japan.

[34] Lawson CL (1986) Properties of n-dimensional triangulations. Computer Aided Geometric Design, 3:231–246.

[35] Ledoux H (2006) *Modelling three-dimensional fields in geoscience with the Voronoi diagram and its dual*. Ph.D. thesis, School of Computing, University of Glamorgan, Pontypridd, Wales, UK.

[36] Ledoux H, Gold CM, 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*, volume 3480 of *Lecture Notes in Computer Science*, pages 737–747. Springer-Verlag, Singapore.

[37] Mioc D (2002) *The Voronoi spatio-temporal data structure*. Ph.D. thesis, Département des Sciences Géomatiques, Université Laval, Québec, Canada.

[38] Mostafavi MA (2001) *Development of a global dynamic data structure*. Ph.D. thesis, Département des Sciences Géomatiques, Université Laval, Québec City, Canada.

[39] Mostafavi MA (2006) Personal communication.

[40] Mostafavi MA and Gold CM (2004) A global kinetic spatial data structure for a marine simulation. International Journal of Geographical Information Science, 18(3):211–228.

[41] Nativi S, Blumenthal MB, Caron J, Domenico B, Habermann T, Hertzmann D, Ho Y, Raskin R, and Weber J (2004) Differences among the data models used by the geographic information systems and atmospheric science communities. In *Proceedings 84th Annual Meeting of the American Meteorological Society*. Seattle, USA.

[42] Okabe A, Boots B, Sugihara K, and Chiu SN (2000) *Spatial tessellations: Concepts and applications of Voronoi diagrams*. John Wiley and Sons, second edition.

[43] Parks BO (1993) The need for integration. In MF Goodchild, BO Parks, and LT Steyaert, editors, *Environmental Modeling with GIS*, pages 31–34. Oxford University Press, New York.

[44] Raper J (2000) *Multidimensional geographic information science*. Taylor & Francis, London.

[45] Roos T (1991) *Dynamic Voronoi diagrams*. Ph.D. thesis, Universität Würzburg, Germany.

[46] Schaller G and Meyer-Hermann M (2004) Kinetic and dynamic Delaunay tetrahedralizations in three dimensions. Computer Physics Communications, 162(1):9–23.

[47] Schirra S (1997) Precision and robustness in geometric computations. In M van Kreveld, J Nievergelt, T Roos, and P Widmayer, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 255–287. Springer-Verlag, Berlin.

[48] Strang WG and Fix GJ (1973) *An analysis of the finite element method*. Prentice-Hall, Englewood Cliffs, USA.

[49] Sugihara K and Inagaki H (1995) Why is the 3D Delaunay triangulation difficult to construct? Information Processing Letters, 54:275–280.

[50] Sui DZ and Maggio RC (1999) Integrating GIS with hydrological modeling: Practices, problems, and prospects. Computers, Environment and Urban Systems, 23:33–51.

[51] Watson DF (1981) Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. Computer Journal, 24(2):167–172.

[52] Zlatanova S, Abdul Rahman A, and Pilouk M (2002) 3D GIS: Current status and perspectives. In *Proceedings Joint Conference on Geo-spatial Theory, Processing and Applications*. Ottawa, Canada.