A submission presented in partial fulfilment of the requirements of the University of Glamorgan/Prifysgol Morgannwg for the degree of Doctor of Philosophy

Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual

Hugo Ledoux

November 2006

Certificate of Research

This is to certify that, except where specific reference is made, the work described in this thesis is the result of the candidate. Neither this thesis, nor any part of it, has been presented, or is currently submitted, in candidature for any degree at any other University.

Candidate:	------------------------------------
Director of Studies:	------------------------------------
Date:	------------------------------------

Abstract

The objects studied in geoscience are often not man-made objects, but rather the spatial distribution of three-dimensional continuous geographical phenomena such as the salinity of a body of water, the humidity of the air or the percentage of gold in the rock (phenomena that tend to change over time). These are referred to as *fields*, and their modelling with geographical information systems is problematic because the structures of these systems are usually two-dimensional and static. Raster structures (voxels or octrees) are the most popular solutions, but, as I argue in this thesis, they have several shortcomings for geoscientific fields.

As an alternative to using rasters for representing and modelling three-dimensional fields, I propose using a new spatial model based the Voronoi diagram (VD) and its dual the Delaunay tetrahedralization (DT). I argue that constructing the VD/DT of the samples that were collected to study the field can be beneficial for extracting meaningful information from it. Firstly, the tessellation of space obtained with the VD gives a clear and consistent definition of neighbourhood for unconnected points in three dimensions, which is useful since geoscientific datasets often have highly anisotropic distributions. Secondly, the efficient and robust reconstruction of the field can be obtained with natural neighbour interpolation, which is entirely based on the properties of the VD. Thirdly, the tessellations of the VD and the DT make possible, and even optimise, several spatial analysis and visualisation operations. A further important consideration is that the VD/DT is locally modifiable (insertion, deletion and movement of points), which permits us to model the temporal dimension, and also to interactively explore a dataset, thus gaining insight by observing on the fly the consequences of manipulations and spatial analysis operations.

In this thesis, the development of this new spatial model is from an algorithmic point of view, i.e. I describe in details algorithms to construct, manipulate, analyse and visualise fields represented with the VD/DT. A strong emphasis is put on the implementation of the spatial model, and, for this reason, the many degeneracies that arise in three-dimensional geometric computing are described and handled. A new data structure, the *augmented quad-edge*, is also presented. It permits us to store simultaneously both the VD and the DT, and helps in the analysis of fields. Finally, the usefulness of this Voronoi-based spatial model is demonstrated with a series of potential applications in geoscience.

About this Thesis

The work described in this thesis represents the results of research I carried out at the Department of Land Surveying & Geo-Informatics, The Hong Kong Polytechnic University, Hong Kong, from October 2001 to August 2004; and at the GIS Research Centre, School of Computing, University of Glamorgan, Wales, UK from August 2004.

I would first of all like to thank the Hong Kong Polytechnic University for financially supporting me during my time there. While I was there, preliminary results—on which this thesis is partly based—concerning the development of a 'Marine GIS' were published:

Ledoux H (2003). Development of a marine geographical information system based on the Voronoi diagram. Department of Land Surveying & Geo-Informatics, The Hong Kong Polytechnic University. Unpublished transfer report.

Gold CM, Goralski R, and Ledoux H (2003). Development of a 'Marine GIS'. In *Proceedings 21st International Cartographic Conference*. Durban, South Africa.

Ledoux H and Gold CM (2004b). Modelling oceanographic data with the three-dimensional Voronoi diagram. In *ISPRS 2004—XXth Congress*, volume II, pages 703–708. Istanbul, Turkey.

A new natural neighbour interpolation algorithm was also published:

Ledoux H and Gold CM (2004a). An efficient natural neighbour interpolation algorithm for geoscientific modelling. In Fisher PF, editor, *Developments in Spatial Data Handling—11th International Symposium on Spatial Data Handling*, pages 97–108. Springer.

Subsequently, other results coming out of this research have been published, as follows. All the papers were written in collaboration with Christopher Gold, and for some other authors also contributed.

Preliminary results discussing the idea of modelling oceanographic datasets with the Voronoi diagram were published in:

Ledoux H and Gold CM (2006a). La modélisation de données océanographiques à l'aide du diagramme de Voronoï tridimensionnel (in French). *Revue internationale de géomatique*, 16(1):51–70.

The deletion of a vertex in a DT (Section 4.4), a joint work with George Baciu and Christopher Gold, is based on:

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.

Ledoux H, Gold CM, and Baciu G (n.d.). Deleting a vertex in a Delaunay tetrahedralization by flips. Submitted to *International Journal of Computational Geometry and Applications*.

The parts about the map algebra framework (Sections 2.4.2 and 5.3) are based on:

Ledoux H and Gold CM (2006c). A Voronoi-based map algebra. In Kainz W, Reidl A and Elmes G, editors, *Developments in Spatial Data Handling—12th International Symposium on Spatial Data Handling*. Springer-Verlag. In Press.

The section about spatial interpolation (Section 5.2) is an extension to $\mathbb{R}^d$ of the algorithm published in Ledoux and Gold (2004a), and is also based on:

Ledoux H and Gold CM (2005b). Interpolation as a tool for the modelling of three-dimensional geoscientific datasets. In *Proceedings 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS*, pages 79–84. Pontypridd, Wales, UK.

Finally, the new data structure, the *augmented quad-edge* (AQE), as described in Section 6.3, is the result of a collaboration between Christopher Gold, Marcin Dzieszko and myself:

Gold CM, Ledoux H, and Dzieszko M (2005). A data structure for the construction and navigation of 3D Voronoi and Delaunay cell complexes. In *Proceedings 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 21–22. Plzeň, Czech Republic.

Ledoux H and Gold CM (2005a). The 3D Voronoi diagram and its dual: Simultaneous construction and storage. In *Topology and Spatial Databases Workshop 2005*. Glasgow, Scotland, UK.

Ledoux H and Gold CM (2006b). Simultaneous storage of primal and dual three-dimensional subdivisions. *Computers, Environment and Urban Systems*. In press.

The original idea of the AQE came from Christopher Gold, and a preliminary implementation of the data structure was done by Marcin Dzieszko.

Acknowledgements

Had I not met Chris Gold when I was an undergraduate at the Université Laval, I wouldn't be writing these lines. His *love* of Voronoi diagrams was so contagious that I ended up first moving to Hong Kong to start my research, and then moving to Wales to continue under his supervision. My deepest thanks go to him for the many ways he has supported and encouraged me over the last five years.

I would also like to thank the members of the 'Polish crew', who have all at some stage contributed to my research. Marcin Dzieszko and Rafał Goralski developed the graphical interface used in this thesis, Krzysztof Matuk helped me several times with technical details that bugged me, and the enthusiasm of Maciej Dakowicz for the triangles kept me going throughout my research! Special thanks also to Joanna Wojewódzka for hearing me whingeing about the triangles more than she should have.

Many thanks to Franz Aurenhammer, George Baciu, Jack Snoeyink and Dawn Wright, who have all helped improve this document with their comments and suggestions. Sometimes, a brief email can help a lot...

My thanks also go to all my office mates in Hong Kong and Pontypridd (Rebecca, Kourosh, Eric, Vijay, Dörte and others) that made my time more enjoyable, and also to my friends (Sylvain, Julie, Sophie, Vincent, Jeff and others) who made sure I was wasting just enough time on *MSN*.

Finalement, mille mercis à Monique, Marc et Kim pour m'avoir encouragé, supporté et tenu au courant de ce qui se passe au Québec pendant mon absence!

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The first geographical information systems (GISs) were developed around 40 years ago to automate the production and the analysis of maps, and have since evolved from their origins (Coppock and Rhind, 1991). Before computers became widely available, the map-making process was tedious and time-consuming for the cartographer, and the analysis and interpretation of different themes for a given area was cumbersome because it had to be done visually (with the help of transparent map sheets layered one over another). Computers and GISs changed that during the 1960s. An example of one of the earliest GIS is the Canada GIS (CGIS), which was developed to store, manipulate and analyse data collected by the Canada Land Inventory. The land capability for rural Canada was determined by extracting statistics for specific areas and by producing map layers for different themes, e.g. agriculture, wildlife, forestry and recreation. The CGIS was probably the first large-scale system to structure geographical data in a database, to partition data into themes (map layers), and to have functions for the measurement of areas and for the overlay of polygons (Tomlinson, 1988); many of its key innovations are still being used in today's commercial GISs. The Canadian government still uses a similar system for the planning and management of its natural resources, but the system in place obviously has more power and a much broader scope.

Although there is still a debate going on about what constitutes a GIS, and although not everyone agrees on a unique definition of GIS (Gold, 2006), there is nevertheless a certain consensus for a definition, which is similar to this one:

> A geographical information system is a computer system capable of capturing, storing, analysing, manipulating, sharing and displaying geographically-referenced data, and their attributes.

The GIS is different from other systems handling spatial data (e.g. CAD systems, medical visualisation systems, engineering design tools, etc.) firstly because it can integrate geographical data from different sources and formats (remote sensing and photogrammetry images, field surveys, GPS data, etc.) into a single system, and secondly because it can analyse them to help the decision-making process. Specialised tools usually have proprietary formats and are designed only for few specific tasks; by contrast, the GIS claims to be open to different datasets and can be used in conjunction with specialised tools to clean or format datasets, and also to analyse (visually and quantitatively) the output of these tools. Although the definition of a GIS theoretically includes any data positioned on the surface of the Earth (and near it), almost all available GISs are built on the idea of a 'smart map', i.e. a computer-based system for creating, querying and producing two-dimensional representations of the surface of the Earth. GISs have indeed *evolved* from their origins, and have never been transformed: visualisation tools, analytical functions, and the possibility to integrate different sources of data, among others, have all been added to the latest GISs, but, the same original structure (the definition of spatial relationships between objects and the data structures used) has not changed much since the 1960s (Mark et al., 1997). Indeed, the definition of the spatial relationships between objects is still based on the intersections of lines to create polygons (e.g. the node-arc-polygon structure), and continuous phenomena are still represented with grid files.

The end of the 1980s saw the arrival of the personal computer, and the GIS morphed from a tool used only by specialists to one that could be used by practitioners in many different fields. Nowadays, commercial GIS companies are trying to accommodate different users by producing modules that extend the functionalities of their GISs for different specialities such as transportation, business, communication, forestry, agriculture, etc. The problem with these extensions is that they only add new features to a GIS (e.g. extra analyses and tools required by the application, or the extension of the database model), and do not change the rigid two-dimensional structure on which these systems are built.

Although current GISs are adequate for many applications—the successes of some commercial GIS companies are there to prove it—their static and flat representation of our world is fundamentally flawed. The world we live in is three-dimensional, and many things are also continually in movement. As a result, many geographical phenomena can not be correctly represented with such systems.

1.1 Three-dimensional Datasets in Earth Sciences

The reduction of the world to a static and two-dimensional representation is particularly problematic for some disciplines related to the Earth, for example oceanography, meteorology, geology and geophysics (these are also called the geosciences). For these disciplines, adding new functionalities on top of a GIS is simply not enough since their data are totally different from the data GISs were designed to handle. This problem has been identified by many researchers in the geosciences. In oceanography and marine applications, Davis and Davis (1988) state that systems capable of handling three dimensions, plus time and attributes are needed to adequately represent marine phenomena. Li and Saxena (1993) notice that very few artificial objects are found at sea and hence phenomena are mostly represented by points (not lines and polygons), and they also propose a 'wish list' of features that GISs should have, one of them being the possibility to do simulation of oceanographic processes. In trying to define what a Marine GIS is, Lockwood and Li (1995) warn the users that the use and the interpretation of marine data could lead to wrong conclusions if they forget that their nature is different (e.g. the distribution of data collected at sea is often highly irregular and the precision of these data is sometimes not as accurate as the precision obtained on the land). In meteorology, Bernard et al. (1998) and Nativi et al. (2004), among others, come to similar conclusions: the representation of the atmosphere requires three spatial dimensions, and a way to integrate time with the spatial dimensions. Such systems are sometimes referred to as four-dimensional systems, since they consider time as an extra dimension. In geology and geophysics, the inadequacy of current GISs has also been noticed, and alternative solutions have been proposed; see for example Jones (1989), Raper (1989) and Sambridge et al. (1995).

What is arguably more symptomatic of the shortcomings of current GISs is the fact that numerous papers in the literature, discussing the use of GIS for environmental modelling, describe how to circumvent the GIS's structure, in order to integrate and analyse datasets. Examples of such papers are the exhaustive review of the use of GISs in oceanography and fisheries by Valavanis (2002); the different applications of ESRI's products in oceanography (Breman, 2002); and the reviews of Chapman and Thornes (2003) and Sui and Maggio (1999) respectively in meteorological and hydrological modelling. The authors of these papers do state that GISs are useful to their application domains, but they still have to reduce their datasets to a two-dimensional plane—usually by 'slicing' it according to a certain direction—to be able to take advantages of the analytical functions of GISs. The advantages of GISs often cited are: (1) they can help to derive datasets from raw data; (2) once a simulation has been run in a specialised tool, the output can be analysed by taking into consideration other datasets; (3) they can convert the formats and/or the cartographic projections of datasets.

An important consideration when dealing with geoscientific datasets is that the objects studied are usually not man-made objects, but rather the spatial distribution of continuous geographical phenomena. Three-dimensional—or volumetric—examples of such phenomena are the salinity of the water, the humidity of the air or the percentage of gold in the rock. These phenomena can be considered as attributes of every location over a given three-dimensional spatial domain, and are continuous because every location $x-y-z$ over the domain has a value (there is *something* at every point). The spatial distribution of an attribute a over a certain spatial extent is also referred to as a *field*. Briefly, a field is a model of the variation of a and can be modelled mathematically by a function (or a mapping) between the location and the value of a, i.e. $a = f(x, y, z)$. In this thesis, a is always considered to be a scalar value, but it is also possible to have vector fields. Notice that the concept of a field is theoretically valid in any dimensions (a becomes a function of the location in d-dimensional space), and time can also be considered (that is, $a = f(location, time)$). Section 2.2 discusses at length the different types of fields, their properties, the operations possible on them, and also the methods used to store them in computers.

Because it is usually impossible to measure continuous phenomena everywhere, we have to resort to collecting samples at some finite locations and reconstructing fields from these samples. Each sample indicates the value of the attribute studied at a certain location $x-y-z$ in space. They can be very hard and expensive to collect because of the difficulties encountered and the technologies involved. To collect samples in the ground we must dig holes or use other devices (e.g. ultrasound penetrating the ground); underwater samples are collected by instruments moved vertically under a boat, or by automated vehicles; and samples of the atmosphere must be collected by devices attached to balloons or aircraft. More details about geoscientific data and the way they are collected is available in Chapter 7. Because of the way they are collected, three-dimensional geoscientific datasets often have a peculiar property: their distribution is highly anisotropic. As shown in Figure 1.1, samples can for instance be abundant vertically but very sparse horizontally. Another peculiar property of geoscientific datasets is that the samples they contain represent spatial variability of a given attribute only at time t, and many geographical phenomena in oceanography and meteorology change and evolve quickly over time.

1.2 Modelling Three-dimensional Fields

Because fields are continuous functions, they must be *discretised*—broken into finite parts—to be represented in computers, which are discrete machines. The space covered by a field can be partitioned, or tessellated, into *regular* or *irregular* parts. In a regular tessellation, all the

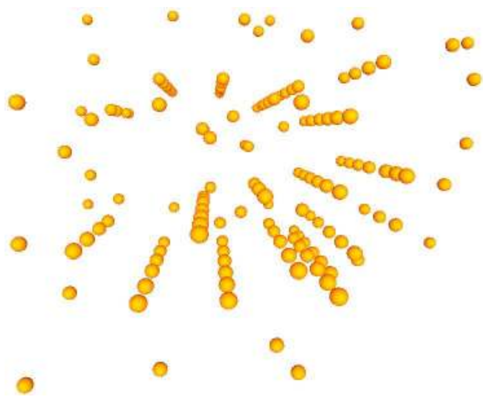


Figure 1.1: Example of a dataset in geology, where samples were collected by drilling a hole in the ground. Each sample has a location in 3D space ($x - y - z$ coordinates) and one or more attributes attached to it.

elements will be of the same shape and size, while in an irregular one, elements of any shape and size are allowed. In the plane (2D), each element is a polygon, while in three dimensions it is a polyhedron. What follows is a brief overview of the most common methods to represent and analyse fields (more details are available in Chapter 2).

Within the GIS community, the raster structure has been for years more or less synonymous with continuous fields. In two dimensions, a raster structure is a regular tessellation with grid squares, and to each element (called a pixel) is assigned the value of an attribute. The popularity of raster structures is probably due to the fact the they are naturally managed in a computer in the form of an array of numbers, and also because no complex data structures are required for the location of objects: the position of an object (a pixel) is implicitly known from the sequential position of pixels in the array. Most commercial GIS products use rasters to represent continuous phenomena, and the tools implemented to model and analyse different fields are based on *map algebra* (Tomlin, 1983), which is a framework developed for the analysis of fields stored in a raster format. With this approach, each field is represented by a grid, and a set of primitive GIS operations on and between fields can be used and combined together to extract information and produce new fields. It has been severely criticised by many who claim that raster structures have drawbacks and are not appropriate to model the reality (Kemp, 1993; Haklay, 2004). Triangulated irregular networks (TINs) are also widely used in the GIS community to model the elevation of a terrain (Peucker et al., 1978). For this particular case, the elevation of a terrain is considered as continuous and without any vertical cliffs or overfolds; the elevation can thus be seen as an attribute of the planimetric locations on the surface of the Earth. A TIN is an irregular tessellation of the $x - y$ coordinates of the terrain into triangles, and the surface of a terrain is approximated by lifting every vertex of the TIN to its elevation, thus creating a mosaic of triangles in 3D space. Although the storage, construction and analysis of a TIN is more complex than with a grid, it is seen by many as a superior method to represent the elevation of a terrain. But many also disagree, and a debate between the advocates of each representation has been going on for years—see Kumler (1994) for an exhaustive review of the topic.

Raster is also the most popular and used structure to represent 3D fields, and different raster systems have been proposed and designed (see for example Jones (1989); Raper (1989); Mason et al. (1994)). For the 3D case, the space is divided into regular cubes (called *voxels*), and each one is linked to the value of a certain attribute. The advantages of 3D grids over other methods are basically the same as in 2D (e.g. simplicity of implementation and manipulation), but the many shortcomings (e.g. amount of memory used) are amplified in 3D, as demonstrated in Chapter 2. Also, the map algebra approach has been recently extended to incorporate the temporal and/or the third dimension (Mennis et al., 2005). As in the 2D case, the disadvantages of a raster representation in 3D have also led to the use of 3D triangulations, called tetrahedralizations since the space is tessellated into tetrahedra, for modelling sub-surface phenomena (Lattuada, 1998; Xue et al., 2004).

The GIS community is not the only one that needs to discretise space, and it is worth studying the methods used in other disciplines. The simulation of physical phenomena in engineering and science, for instance the heat transfer of a mechanical part, the fluid flow around an aircraft or the electromagnetic wave propagation, is usually modelled by partial differential equations (PDEs). Because these equations are continuous and can be impossible to solve in some cases, a numerical approximation of the solution is usually sought. The most popular method to approximate PDEs is the *finite element method* (FEM) (Strang and Fix, 1973). Without being too technical, the solution of a PDE is obtained by first partitioning the object under study into small pieces (called elements), then approximating the behaviour of the phenomena for each element, and by accumulating the result in each element, it is possible to approximate the phenomena being studied. The discretisation of objects (2D or 3D) is referred to as *meshing*, and automatically generating a mesh whose elements are well-shaped for the simulation of a given phenomenon has historically been proved to be more challenging than the simulation process itself (Shewchuk, 1997b). The elements used can be of any shape, but irregular tessellations tend to be more popular since they can adapt easier to irregularly shaped objects. As is the case in Earth sciences, the object being studied needs to be sampled (e.g. the temperature at several locations is recorded), but because we usually have access to the objects, the sampling process is much easier and obtaining samples that are well-distributed on or in the object and cover the whole object is possible. Although a lot of work is being done by the meshing community, the methods they develop are not necessarily adapted to the modelling of geoscientific fields. The aim of the meshing community is to tessellate objects as best as they can, which usually means with elements that are as well shaped as possible. Once this is done, the mesh is usually used to approximate the PDEs. Their algorithms to mesh objects are usually *static*, i.e. their meshes, once constructed, can not usually be modified on the fly, and their data structures are usually not flexible enough to permit further analyses (they do not represent many topological relationships between elements). By contrast, the modelling of fields and datasets in geoscience is optimised when interactive tools to manipulate and analyse a datasets are present (Gold, 1993; Anselin, 1999; Lee and Gahegan, 2002), and dynamic algorithms also permit us to represent time-changing phenomena.

Another related application domain is medicine, particularly medical imaging. Three-dimensional datasets are also frequent, and several tools have been developed to analyse and extract (visually and quantitatively) meaningful information from raw data. The major impediment to the use of medical imaging techniques in geoscience is the fact that their datasets are usually ‘complete’, i.e. they have technologies to collect full representation of one part of a body (a ‘3D image’). Three-dimensional grids are therefore naturally used, hence techniques to analyse 3D objects are based on this structure.

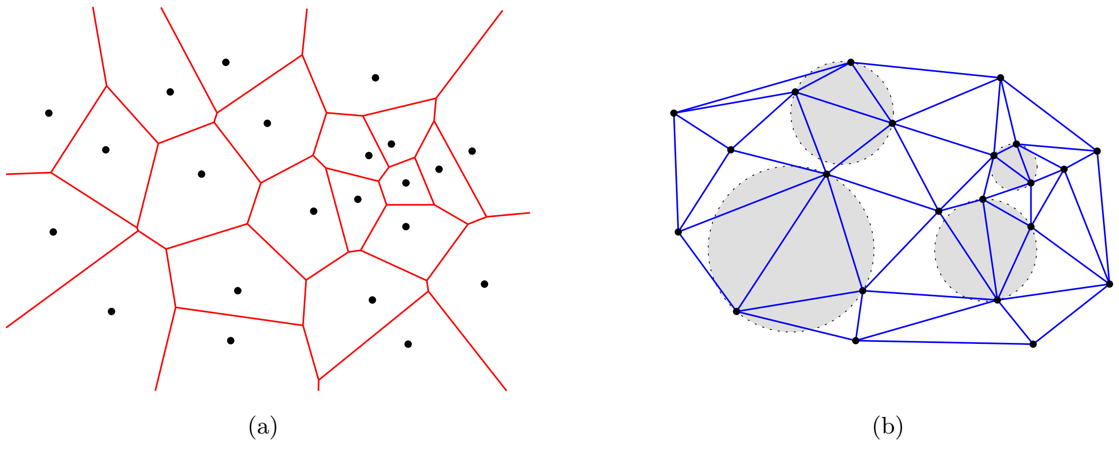


Figure 1.2: **(a)** The Voronoi diagram, **(b)** the Delaunay triangulation, for the same set of points in the plane.

1.3 Objectives of the Research

The main objective of this research is to develop generic tools for the modelling of three-dimensional fields in Earth sciences. Three-dimensional datasets collected to study the Earth have properties that are usually not present in datasets in other disciplines, for example engineering, medicine or computer-aided design. Tools, methods and data structures developed in these fields are not adapted to the modelling of 3D fields (I elaborate more on the shortcomings of techniques in other disciplines in Chapter 2). For this reason, this research aims at developing a new spatial model that is adapted to the properties of the datasets commonly found in Earth sciences. The term 'spatial model', or spatial data model as it is often called, is "a limited representation of reality, constrained by the finite, discrete nature of computing devices; the term *discretization* conveys much the same meaning as data model" (Goodchild, 1992a).

The development of this three-dimensional spatial model is based on previous results in two dimensions with two related geometric structures called the *Voronoi diagram* (VD) and the *Delaunay triangulation* (DT). Given a set of points in the plane, the former structure partitions the plane into 'proximity zones', i.e. each point is mapped to the region of the plane (a polygon) closest to it (see Figure 1.2(a)). The latter structure is closely related to the Voronoi diagram. It partitions the plane into triangles—where the points generating the Voronoi polygons are the vertices of the triangles—that have the property of having an *empty circumcircle*, i.e. the circle circumscribed to each triangle does not contain any point in its interior. The two structures are *dual* to each other, i.e. they represent the same thing, just from different viewpoints. They have interesting properties that make them important and useful in a wide range of applications (see Aurenhammer (1991)), and they can also be generalised to any dimensions (Okabe et al., 2000). Their formal definition and a review of their properties are presented in Chapter 3. For GIS applications, the Delaunay triangulation has been used for many years as the triangulation of choice in TINs for terrain modelling applications. The Voronoi diagram is also important in GIS: it has been used to answer different types of neighbourhood queries (Okabe et al., 1994), and its 'tiling' properties, which can be used to define spatial relationships between objects in a map, has led to its use as an alternative to current GIS structures (Gold, 1991; Gold et al., 1997). Its major advantage over other structures is that it can be *locally* updated when a new object is inserted, deleted or moved, and there is therefore no need to reconstruct large parts of a map when a change occurs. This is particularly interesting for the management of time in a GIS (Mioc, 2002) and for the interactive exploration of a dataset (Lee and Gahegan, 2002). An outline of a two-dimensional Marine GIS, entirely based on the VD to handle topology and perform spatial analysis and operations, has been developed by Gold and Condal (1995). They show it has many advantages over the traditional spatial model of GISs for a wide range of applications at sea. Wright and Goodchild (1997), in a review, affirmed that this method was the only published attempt at that time to solve many important problems related to the nature of marine data. The marine data they refer to have several similarities with data collected in Earth sciences: marine datasets are formed mostly by unconnected points (e.g. bathymetric samples or boats), the distribution of samples is highly irregular (data are collected according to ship's tracks), and things at sea tend to change position over time.

In brief, a spatial model based on the two-dimensional VD elegantly solves many problems encountered with traditional GISs, and, for this reason, the assumption on which this thesis is based on is that the three-dimensional VD could bring the same benefits for three-dimensional datasets. The aim of my research is therefore to extend to three dimensions the tools, algorithms and data structures developed for the 2D VD, and ultimately to build a new spatial model—based on the 3D VD—to represent and analyse trivariate fields. In other words, my research attempts to answer the following question:

> Is the Voronoi diagram an appropriate structure to model three-dimensional and dynamic fields in geoscience?

It is worth mentioning here that a spatial model based on the 3D Voronoi diagram has been briefly discussed in Gold and Edwards (1992), but, to my knowledge, no attempts were made to build one.

To achieve my objectives, I will:

1. Look closely at the applications in geoscience dealing with trivariate fields and review the known spatial models.
2. Validate the use of the Voronoi diagram in 3D as a spatial model. Studying its properties is an important part of this research, to ensure that it can appropriately represent 3D fields for a wide range of applications, including those having dynamic components.
3. Develop and implement algorithms for the construction and the manipulation of 3D datasets.
4. Detail the GIS operations (spatial analysis) needed to model and analyse trivariate fields as found in the Earth sciences, and implement these operations.
5. Validate the concepts with a prototype that will be tested with a few datasets from different disciplines related to the Earth.

1.4 Scope of the Research

This research aims at developing a prototype GIS that will manage three-dimensional geoscientific fields. The research is conducted from an algorithmic point of view: its main results are the description of a spatial model, the data structures to store it, and a set of algorithms for manipulating and analysing 3D fields. A strong emphasis is put on the design of *dynamic* and *kinetic* algorithms, that is algorithms that permit us to modify a structure (the 3D Voronoi diagram in this case)—by inserting, deleting or even moving data points—without recomputing it from scratch. Many optimal algorithms in the literature are static, which means that for an algorithm to run, its input must be totally known beforehand. If the input changes afterwards, even slightly, then the algorithm has to run again. Static algorithms are a major obstacle when one wants to interactively explore a dataset to extract information, as explained in Section 2.5. As for kinetic algorithms, which basically means that the input to a function continually changes over time, they are used to model time and to perform simulations.

Many efficient algorithms that would be useful in GIS are being developed in other disciplines, particularly in computational geometry. Although computational geometers design algorithms and data structures for many geometric and GIS-related problems, their solutions are in most cases theoretical, i.e. the asymptotic worst-case complexity of an algorithm is the factor deciding the performance of an algorithm. Having a theoretical framework to compare algorithms is important, but the discipline has also been criticised for not being practical enough. In a report, the computational geometry community itself acknowledged that: "Paper descriptions of worst-case optimal algorithms do not constitute significant improvement in a community [GIS] that wants implementations of solutions" (Chazelle et al., 1996). Criticism of the report triggered a lively discussion[1] about what computational geometry should be. Many argued that it should be more application-oriented, focus more on the development and analysis of practical algorithms, and also work more in collaboration with practitioners in computer graphics, GIS, biology, etc. Some also argued that worst-case complexity of an algorithm is not always the best indicator of its practical performance, and that theoretical results in two dimensions do not necessarily generalise to higher dimensions. Another problem is the gap that sometimes exists between a theoretical algorithm and its implementation; this is mostly caused by degeneracies that arise in geometric computing, as explained in Chapter 3.

The goal of this research is not to design theoretically optimal algorithms, but rather robust algorithms that are simple to implement and in practice take a reasonable time to solve reasonable problems. Simpler algorithms can usually be implemented easily and they are also less error-prone—some theoretically optimal algorithms have been proved almost impossible to implement! A strong emphasis is put on *robustness* of the algorithms, so that they output a correct solution, regardless of the spatial distribution of the points in the input. In brief, I want to take ideas and results from the field of computational geometry and apply them in the context of the modelling of 3D fields in the geosciences. The work in this thesis is thus at the interface of the disciplines of GIS and computational geometry.

Building a completely operational system is clearly out of scope considering the time schedule available for this research. Therefore, I aim at developing generic concepts to model and analyse three-dimensional fields, and testing them by building a prototype GIS. The following is a list of relevant topics to the modelling of 3D fields that are *not* covered in this research:

- Data storage and maintenance: for this research, it is assumed that attributes for each object can be stored in a database linked to the GIS, but no data model is proposed for handling the different types of data and their attributes.

[1]On the `compgeom` mailing list, in 1996.

- Data quality and errors: all input data in the prototype are assumed to be ‘clean’ and ‘error-free’.
- Very large spatial databases: datasets collected to study the Earth, because they are three-dimensional and because of the time factor, can become enormous. This research is not concerned with this problem and assumes that all the data can be stored in the main memory of a computer.

1.5 Outline of the Thesis

The thesis is organised in eight chapters (including this one). Although the next two chapters act as literature review respectively in GIS and computational geometry, the problems, operations and applications described in Chapters 4, 5, 6 and 7 are all introduced by citing the appropriate related work. Also, most of the concepts, operations and algorithms discussed are introduced by describing first their two-dimensional counterparts, because readers are often more familiar with two-dimensional modelling techniques, and the visualisation of certain three-dimensional issues is also rather difficult.

Chapter 2 introduces the GIS concepts needed for the understanding of the rest of the thesis, and acts as a literature review of the techniques used in different disciplines for representing and analysing spatial data. A big part of the chapter is devoted to describing the different spatial models to represent fields in a computer. The chapter concludes by highlighting the shortcomings of available techniques for representing geoscientific fields.

Chapter 3 introduces several concepts in computational geometry that are necessary to the understanding of Chapters 4 and 5. The Voronoi diagram and the Delaunay tetrahedralization, and their relationships, are formally defined. The difficulties that arise when implementing geometric algorithms are also discussed.

Chapter 4 presents the algorithms for manipulating a three-dimensional Voronoi diagram. Detailed algorithms, which are robust against all degenerate cases, are given for the construction of the VD, the deletion of a vertex in a VD, and also the movement of a vertex in a VD.

Chapter 5 presents several GIS and spatial analysis operations for trivariate fields that are possible when a field is represented with the VD.

Chapter 6 discusses some implementation issues related to the prototype developed for this research. The problem of computing with floating-point arithmetic is discussed, as well as the data structures that can be used to store the VD. A new data structure that was developed for this work is presented. The practical performance of the algorithms developed are also tested.

Chapter 7 presents a few applications in geoscience where a spatial model based on the VD is useful. I explain how some properties of the VD can help for many ‘generic’ applications, and also discuss the use of the VD for real-world applications in oceanography, meteorology and geology.

Chapter 8 concludes the thesis by summarising the advantages of using a Voronoi tessellation of space to represent trivariate fields as found in the geosciences, and by outlining the major contributions of this research. Recommendations for further research are also given.

Chapter 2

State-of-the-art GIS

This chapter gives an overview of the GIS concepts and techniques that are related to the modelling of fields in geoscience. Since GIS has traditionally been two-dimensional, many of the concepts discussed are first introduced with two-dimensional examples, and then their three-dimensional counterparts are presented. I first discuss notions related to the space, and the different models available to represent it in a computer are reviewed. Fields are then formally defined and a survey of the different spatial models described in the literature is presented. The existing methods to extract meaningful information from fields and to analyse the spatial variation within a field are also discussed. I conclude this chapter by highlighting the weaknesses of current systems and summarising what is needed to accurately model trivariate fields in geoscience.

2.1 Spatial Modelling

Space is a rather peculiar concept because, although every human has an intuitive idea of what it is since it interacts with it constantly, defining it is somewhat tricky. The nature of space has been a debate for a long time (at least since antiquity), and no clear and uncontroversial definition exists because different disciplines (e.g. mathematics, physics, geography and geography) have different definitions that are context-related (Couclelis, 1999). Different typologies of space, based on different criteria, have nevertheless been proposed. For example, Zubin (1989) classifies space into four categories according to the human's perception, and Mark (1992) has three categories, based on people's interaction with space; Freundschuh and Egenhofer (1997) offer a thorough summary of the different typologies related to GIS. Regardless of all the classifications, the type of space that is most relevant in the context of this thesis is the space that is usually referred to as *geographical space*. Cities, provinces and countries are all example of such a space because they are too large to be manipulated, and they are related to entities that lie on or near the Earth's surface. Small objects such as a bicycle or an airplane are not included because they are not of geographical scale; note that systems to model such objects will usually be called 'spatial' and not 'geographical'.

As Frank (1992) explains, the concepts used by humans to organise and structure space can not necessarily be implemented in a computer "either for lack of formal definition or for lack of discretization". A representation of geographical reality that can be implemented in a computer is referred to as a *spatial model* (which is also sometimes called a spatial data model, or simply a data model). A model is "a simplified representation of an object of investigation for purposes of description, explanation, forecasting or planning" (Fotheringham and Wegener, 2000). In the case of a spatial model, we are interested in representing the geographical entities, their

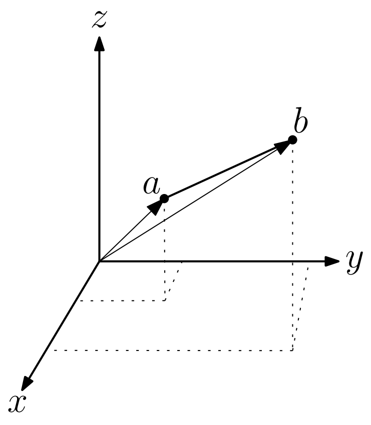


Figure 2.1: Euclidean space in three dimensions.

relationships, and the phenomena that are in a certain region. The entities, which will be abstracted and mapped into the model, will become objects and the (spatial) relationships between objects will have to be explicitly defined in the model. The two following sections, Sections 2.2 and 2.3, discuss models for geographical data.

2.1.1 Representations of Space

The representation of geographical space that best abstracts people's perception of small- and medium-scale space is the *Euclidean space* (Couclelis, 1999; Mark et al., 1989; Worboys and Duckham, 2004). This is a metric space, i.e. a space where the notions of distance between two points and of angle between two vectors can be defined. The Euclidean space in three dimensions, denoted $\mathbb{R}^3$, has a coordinate system that has a fixed origin, and three mutually orthogonal axes, usually named x, y and z (as shown in Figure 2.1). Note that in two dimensions, the Euclidean space (denoted $\mathbb{R}^2$) is the well-known *Cartesian plane*. In $\mathbb{R}^3$, a point a is defined by a tuple of real numbers (a_x, a_y, a_z), which defines a vector from the origin $(0,0,0)$ to (a_x, a_y, a_z). The length, or norm, of vector $\vec{a}$ is

$$||\vec{a}|| = \sqrt{a_x^2 + a_y^2 + a_z^2}\,, \tag{2.1}$$

and the distance between any given points a and b, denoted $|ab|$, is the straight line joining the two points:

$$|ab| = ||b - a|| = \sqrt{(b_x - a_x)^2 + (b_y - a_y)^2 + (b_z - a_z)^2}\,, \tag{2.2}$$

which is obtained by using the Pythagorean distance. Also, the (interior) angle θ between vectors $\vec{a}$ and $\vec{b}$ is obtained by simple geometry:

$$\cos\theta = \frac{a_x b_x + a_y b_y + a_z b_z}{||\vec{a}||\,||\vec{b}||}\,. \tag{2.3}$$

Another important concept when discussing space in a GIS context is *topology*. This is a branch of mathematics concerned with spatial properties preserved under *topological transformations* (or *homeomorphism*, which basically means stretching and twisting, but not tearing, puncturing or folding). Take for example a map of Europe made of rubber, if one stretches or twists the map, certain properties will be preserved, such as the fact that France and Spain are adjacent, or that Berlin is situated inside Germany. No matter how one stretches the rubber map, these properties will remain true because they are not based on distance or size, but rather on how things are 'connected together'. The study of topology is a central theme in GIS because it permits us to formalise how geographical entities are connected or relate to each other in space, especially when some transformations, e.g. a change of cartographic projection, are applied to a map. It should be noticed that what is usually referred to as topology in the GIS-related literature is a fairly reduced view of a much broader subject.

2.1.2 Conceptualisations of Space

Space can be conceptualised according to two different approaches: the *object* and the *field* views (Couclelis, 1992; Goodchild, 1992a; Peuquet, 1984). The former view considers space as being 'empty' and populated with discrete entities embedded in space. The entities can be for example roads, cups of tea, churches, etc., and they have certain properties. The latter model considers the space as being continuous, and every location in space has a certain property (there is *something* at every location). Entities or things are formed by clusters of properties.

The field-object dichotomy is a debate that has been going on for years in different disciplines, and is still considered open (Couclelis, 1992). In philosophy for example, it dates back to Kant who considered space not to exist by itself and being formed by the entities that populate it, and to Descartes who argued that space exists by itself. In physics, it is analogous to the dichotomy between the plenum and atomic perspectives. The work of Newton is based on what he called the absolute space, where the space is a container of things and event. By contradiction, his rival Leibnitz assumed a relative space where there is no emptiness. Also, Couclelis (1992) investigates the dichotomy from a human cognition point of view, to understand how the geographical world is perceived and categorised by humans. She states that "cognitive geography will not side with either the object nor the field side of the controversy, but supports a little of each, and much beyond". In brief, humans use both approaches, but for different purposes.

The dichotomy also has serious practical consequences as it influences the way geographical entities and phenomena are represented in a GIS. Although typical commercial GISs usually offer one or the other representation, some offer the possibility to have both, with operations to convert representations back and forth. Even when both representations are supported, a distinction between the two is always present as many spatial analysis operations are based on one representation or the other (Albrecht, 1996).

2.1.3 Confusion between Spatial Models and Data Structures

In the GIS community, there exists a confusion between the terms 'spatial model' and 'data structure'. The confusion originates from the fact that object and field views of space are usually implemented in a GIS with respectively vector and raster models (Frank, 1992; Goodchild, 1992a). A vector model represents individually each object with primitives such as points, lines and polygons (and polyhedra in 3D), and have hence been linked to the object view. The confusion mainly comes from raster representations because they are more or less synonymous with fields in the GIS community. However, as Goodchild (1992a) points out, and as explained in Section 2.2, fields can also be represented by vector models (e.g. if a triangulation is used).

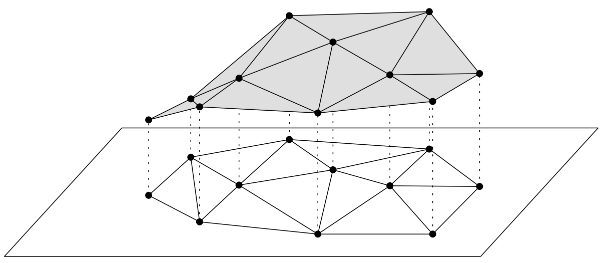


Figure 2.2: A 2D surface obtained from a set of points in the plane having a height as an attribute.

According to Frank (1992), a spatial model offers an abstract view of a data structure, which is defined as “the specific implementation of a geometric data model, which fixes the storage structure, utilization, and performance”. The same spatial model can therefore be implemented with different data structures.

In this thesis, the terms ‘object view’ and ‘field view’ are used to refer to spatial models, while ‘vector’ and ‘raster’ are used to refer to the implementation details of a spatial model.

2.1.4 Dimensionality of a Spatial Model

The term ‘three-dimensional’ is misleading in a GIS context because it might refer to two different concepts:

1. Each location in space has two independent coordinates (usually x and y), and one dependent variable z that is usually the height (or elevation) of a terrain.

2. Each location in space has three independent coordinates (x, y and z); z is thus not an attribute. One dependent variable a can be assigned to each location.

The first kind of ‘third dimension’ is when a two-dimensional surface is embedded in three-dimensional space. Because each location (x, y) is assigned to one and only one height z, the surface can be projected unto the plane (x, y), as shown in Figure 2.2. The surface can be seen as geometrically three-dimensional, but topologically two-dimensional, for when it is embedded in a two-dimensional space, it keeps the same structure (if it was a triangulation, the adjacency relationships between triangles would stay the same). It should be noticed that the modelling such surfaces is sometimes referred to as ‘2.5D GIS’.

The second concept can refer to two different situations. In the first one, the three independent coordinates (x, y, z) represent the exterior of an object (as in a boundary representation). One example is the work of Tse and Gold (2004) where a triangulation is used to represent the surface of the Earth, with holes (for tunnels), cliffs and caves. This extension to the TIN structure is possible with the help of computer-aided design (CAD) functions. The second situation refers to the modelling of not only the boundaries of an object, but also of its interior (the whole volume covered by an object).

The state-of-the-art in 3D GIS developed by major commercial companies usually means adding a 3D visualisation package to a 2D GIS to give the impression of a 3D environment. Although texture of terrain, roads and buildings are added over the terrain, and 3D bird’s view navigation is available, this adds nothing to the two-dimensional structure of the terrain. The buildings are usually not 'topologically connected' to the ground and hence no 3D spatial analysis is possible.

2.2 Field-based Models

A field is a concept rather difficult to define because it is not tangible and not part of our intuitive knowledge. It is easy for us to see and describe entities such as houses or chairs, but, although we can imagine fields, they are somewhat an abstract concept. The consequences of that are firstly that formalising a field is difficult, and secondly that many definitions exist in different disciplines (Peuquet et al., 1999). The definition usually used in a GIS context, and used in this thesis, is borrowed and adapted from physics. Physicists in the 19th century developed the concept of a *force field* to model the magnetic or the gravitational force, where a force (a vector with an orientation and a length) has a value everywhere in space, and changes from location to location. In this thesis, the vector assigned to each point of the Euclidean space is replaced by a scalar value, and we obtain *scalar fields*; note that, in this thesis, unless explicitly stated, the word 'field' always refers to a scalar field.

Because each point in space possesses a value, a field can be represented mathematically. It is a model of the spatial variation of a given attribute a over a spatial domain, and it is modelled by a function, from $\mathbb{R}^d$ to $\mathbb{R}$ in a d-dimensional Euclidean space, mapping the location to the value of a, thus

$$a = f(location).$$

The function can theoretically have any number of independent variables (i.e. the spatial domain can have any dimensions), but in the context of geographical phenomena the function is usually bivariate (x, y) or trivariate (x, y, z). Note that the domain can also incorporate time as a new dimension, and *dynamic fields*, such that $a = f(location, time)$, are thus obtained (Kemp, 1993)—see Section 2.6 for more details.

2.2.1 Properties

2.2.1.1 Scale of Measurement

The value of the attribute a of a field can be measured according to different scales, and depending on the scale used different types of fields will be obtained. It is important to discuss the different scales because the mathematical operations possible on fields will be different when different scales are used. For geographical data, the scale of measurement usually used is that of Stevens (1946), in which four scales are defined:

Interval: the values of an attribute can be any real number $\mathbb{R}$, but the zero point on the scale is arbitrarily defined. Operations such as addition and subtraction are meaningful, but division is not. The Celsius and Fahrenheit scales are two obvious examples: 30℃ is 20℃ warmer than 10℃, but it is not three times warmer. Only the magnitude of the difference between two values is meaningful.

Ratio: the values of an attribute have all the features of interval measurement, but here the ratio between two values is meaningful. In other words, a ratio scale has a fixed zero point. Obvious examples are when temperatures are measured in Kelvin (instead of Celsius), or the amount of snow that has fallen in a certain region.

Nominal: the values of an attribute are simply labels, and they are meaningless. An example is a map of Europe where each location contains the name of the country. The only comparison that can be made between two values is if they are the same or not.

Ordinal: the values are also labels, but they can be ordered, e.g. a certain region can be categorised according to its suitability to agriculture from 1 to 5: 1 being poor, and 5 very good. Here comparisons such as 'greater than' and 'less than' can be made, but no arithmetic operations are possible.

The first two scales are considered as being *continuous*, and most fields studied in the geosciences fall into this category. Temperature, precipitation or salinity are examples because they can be measured precisely. I refer to this type of field as a *continuous field*. The last two scales are less common, but as shown in Chapter 7, are useful in some applications. I refer to this kind of field as a *discrete field*.

2.2.1.2 Continuity

The use of the terms 'continuous' and 'discrete' can be misleading because in mathematics continuity has a slightly different meaning. The terms refer here to the scale of measurement, and not to the spatial continuity of a field. Indeed, both types of fields are spatially continuous, as they are represented by a function and there exists a value at every location in space.

2.2.1.3 Anisotropy

Anisotropy is the property of being directionally dependent, and it is the opposite of isotropy. An anisotropic field is thus a field for which the values of the attribute varies depending on the direction. Most fields in the geosciences are anisotropic.

2.2.2 Digital Representations of Fields

The representation of a (geographical) field in a computer faces many problems. Firstly, fields are continuous functions, and, by contrast, computers are discrete machines. Secondly, in the context of the geosciences, we rarely have a complete coverage of a geographical phenomenon, unless a dataset comes from photogrammetry or remote sensing (in three-dimensions, it is almost impossible to have complete representation). We must therefore rely on *sampling* a field, and reconstructing the field from the samples. The reconstruction of a field from a set of discrete samples is based on the notion of *spatial autocorrelation*, which states that the similarity between a given attribute at two locations x_i and x_j increases as the two locations converges. That follows Tobler's First Law of Geography: "everything is related to everything else, but near things are more related than distant things" (Tobler, 1970). Therefore, to represent a field—which, I should stress, is usually obtained from a set of discrete samples—in a computer, we need to:

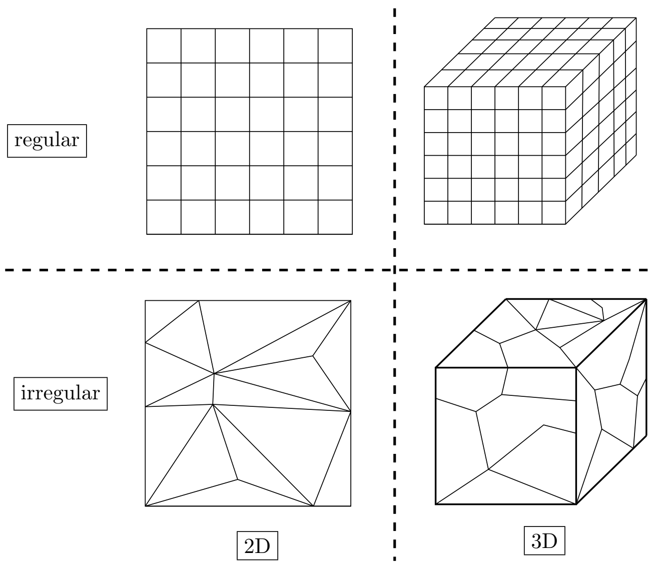


Figure 2.3: Examples of regular and irregular tessellations in two and three dimensions.

1. tessellate into finite elements the space covered by the field. A tessellation is a partition of the space by a set of disjoint elements, such that there are no empty parts (the union of all the elements completely fills the space).

2. define a set of rules to obtain the values of the attribute studied at any location x. This is akin to interpolation methods, which are further covered in Section 5.2.

In the GIS literature, there are usually six different representations of 2D fields (Kemp, 1993; Longley et al., 2001): (1) regularly spaced sample points, (2) irregularly spaced sample points, (3) TIN, (4) rectangular cells (raster), (5) polygons, (6) contour lines. Theoretically, these six representations generalise trivially to three dimensions, except perhaps the contour lines that become contour surfaces, which could be tricky to implement and visualise. Although the first two representations are commonly used, they are *incomplete*, as the set of rules to reconstruct the fields between the samples is not explicitly defined, and should not, in my opinion, be considered valid representations. These six usual representations are not exhaustive, and some are conceptually the same, e.g. TIN and polygons.

I categorise the representations based on the type of tessellation, and also on the function used to reconstruct the field. The space covered by a field can be tessellated *regularly*, *irregularly*, or *hierarchically*. Once partitioned, the field function becomes *piecewise*: to each region is assigned a function describing the spatial variation in its interior.

2.2.2.1 Regular Tessellations

Regular regions all have the same shape and size. The most common regular tessellation in GIS is by far the grid (or raster representation), where the elements are squares in 2D (usually called *pixels* as an analogy to digital images) and cubes in 3D (called *voxels*, a portmanteau of the words 'volume' and 'pixel'), see Figure 2.3. Notice that other regular shapes are possible, such as hexagons.

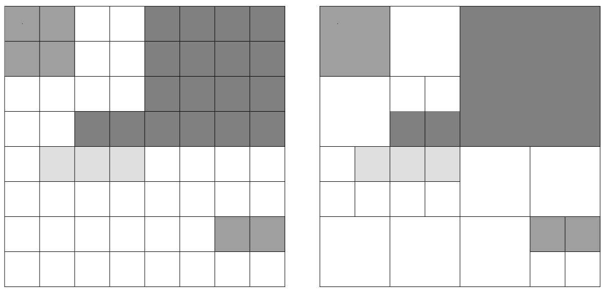


Figure 2.4: Examples of a raster representation (left), and its compression with the region quadtree (right).

The wide popularity of raster representations in GIS applications is probably due to the fact that they permit us to integrate remote sensing images and fields, and also due to simplicity. Indeed, a grid is naturally stored in a computer as an array (each grid cell is addressed by its position in the array, and only the value of the grid cell is stored), and thus the spatial relationships between cells are implicit. The algorithms to analyse and manipulate (boolean operations such as intersection or containment) are also trivially implemented in a computer. On the other hand, raster representations have been severely criticised, both from a theoretical and a technical point of view. The issues involved when using pixels as the main element for geographical analysis are summarised in Fisher (1997). One problem is that a grid arbitrarily tessellates the *space*, without taking into consideration the objects embedded in it or the phenomena modelled. Kemp (1993) states that "[a raster representation] requires us to enforce a structure on reality rather than allowing reality to suggest a more appropriate structure for our analysis". On the technical side, the problems most often cited are: (1) the meaning of a grid is unclear (are the values at the centre of each grid cell, or at the intersections of grid lines?), (2) the size of a grid (if a fine resolution is wished, then the size of a grid can become huge, especially in three dimensions), (3) grids scale badly and are not rotationally invariant, (4) an entity is not represented explicitly, but by a group of grid cells with similar values, which means that its boundaries will be jagged. Also, unless a grid comes from an image sensor (remote sensing or photogrammetry), we can assume that it was constructed from a set of samples. Converting samples to grids is dangerous because the original samples, which could be meaningful points such as the summits, valleys or ridges or a terrain, are not present in the resulting grid. The samples are in a way lost. Also, when a user only has access to a grid, he often does not know how it was constructed and what interpolation method was used, unless meta-data are available.

A few implementations of a voxel-based GIS exist, notably the open-source system GRASS[1] in which diverse operations such as interpolation and visualisation are possible. Also, most of the earlier attempts at building 3D GISs were using voxels, as explained in Raper (1989).

2.2.2.2 Hierarchical Tessellations

To solve the problem with massive raster files, hierarchical tessellations can be used. A commonly used structure in two dimensions is the *region quadtree*, which indexes and merges adjacent grid cells to save space. As shown in Figure 2.4, it recursively subdivides the space

[1]Geographic Resources Analysis Support System (`http://grass.itc.it`).

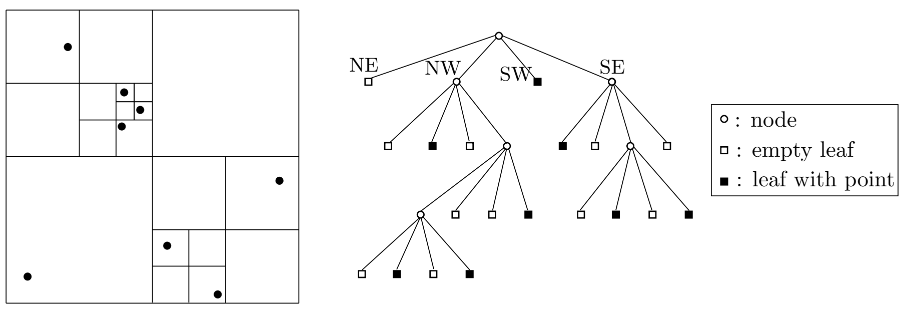


Figure 2.5: Quadtree of a set of points in the plane.

into four squares of equal size, until every square contains one homogeneous region (based on the attribute of every grid cell).

A quadtree is a generic term for a family of tessellations that recursively subdivide the plane into four quadrants, labelled NE, NW, SW and SE (NE being north-east, SW being south-west, and so on). It is easily implemented in a computer because it is a tree (see Section 3.1.1), where each node has exactly four children, if any. Different variant of quadtree have been proposed (see Samet (1984) for an excellent survey), but I focus here on the most relevant type for modelling fields, the *PR quadtree* (P stands for point, and R for region). This assumes that the quadtree is based on a set of points, e.g. the samples collected to study a field. A PR quadtree recursively subdivides the plane into squares of equal size until each square contains a maximum of one point, and each node of the tree therefore corresponds to a square (see Figure 2.5). Notice that some squares do not contain any points, and also that a quadtree offers an adaptive solution as more squares are present to capture more details where there are more points. The quadtree easily generalises to higher dimensions. For instance, in three dimensions, the space is recursively subdivided into eight octants, and thus the tree is called an *octree* for each node has eight children.

The shortcomings of octrees are similar to those of rasters: the rotation and scaling operations are difficult to handle, and only a rough approximation of the boundaries of an object is possible (e.g. a surface separating two different attributes). Worboys and Duckham (2004) also note that the structure is highly dependent on the position of the points, i.e. a small rotation of the set of points, or a small change in the location of a few points may modify the octree considerably.

The main advantage of using the structure—saving memory space—is actually true only when the distribution of the points is more or less regular. Indeed, the size and depth of an octree is not dependent on the number n of points it stores, but rather on their distribution in space (see de Berg et al. (2000) for an analysis). When the point distribution is irregular (e.g. contains clusters), the tree can be unbalanced (and have arbitrary depth), and therefore the query and the manipulation (insertion or deletion of points) of an octree can be extremely slow. The *compressed quad/octree* improves this situation by eliminating all the non-leaf node having only one non-empty child, but the depth of an octree can still be $\mathcal{O}(n)$ in the worst case (Clarkson, 1983). In spite of its shortcomings, it should also be noticed that the PR quad/octree is a structure that can be easily modified dynamically (insertion and deletion of a single point), and is thus attractive for the modelling of fields in an interactive context (see Section 2.5 for more details about dynamic modelling).

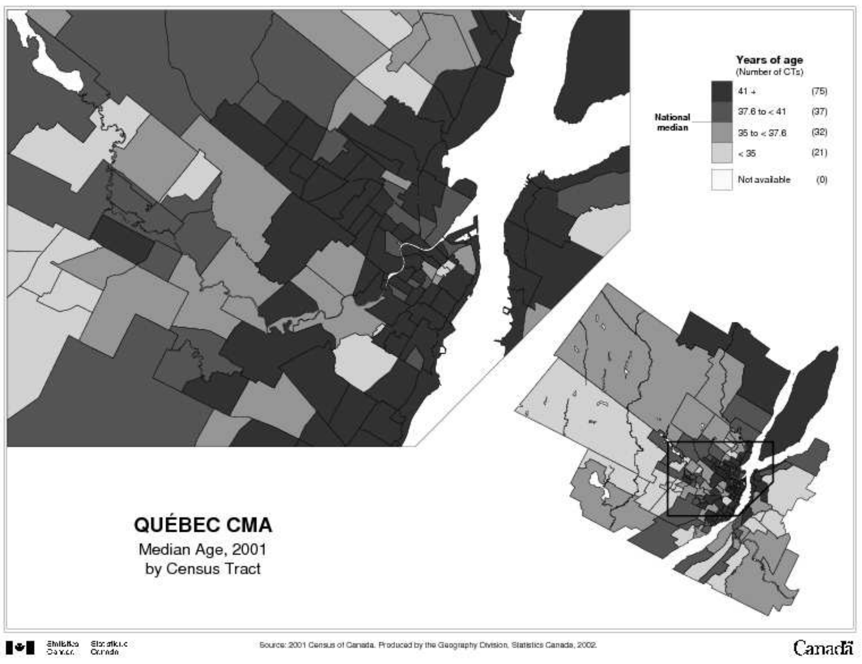


Figure 2.6: An example of a choropleth map (figure from Statistics Canada).

Examples of the use of octrees in GIS projects are plenty: Bak and Mill (1989) and Jones (1989) use it for modelling geological structures; Li and Saxena (1993) use it for modelling the three-dimensional objects in their prototype marine GIS; and O'Conaill et al. (1992) and Mason et al. (1994) show its advantages for many applications in the geosciences.

2.2.2.3 Irregular Tessellations

The elements of an irregular tessellation can be of any shape and size. In the plane, each element forming the tessellation is a polygon, while in three dimensions it is a polyhedron. Irregular tessellations usually follow the outline of the data points (the samples that were collected to study the field), albeit this is not a requirement. Subdividing the space based on the samples has the main advantage of producing a tessellation that is adaptive to the sample distribution and to the complexity of the phenomenon studied. The subdivision is potentially better than that obtained with quad/octree because there are no constraints involved when subdividing the space.

The most known examples of the use of irregular tessellations in GIS are arguably the choropleth map and the TIN. The former subdivides the plane into arbitrary polygons where all the locations within a given polygon have the same values. They are thus discrete fields and used mostly in social sciences for the visualisation of statistical variables such as densities, rates or proportions. Figure 2.6 shows one example where the median age of the population for different districts of Québec City is shown. A TIN tessellates the plane into triangles formed by data points representing the elevation (or other attributes) at a certain location (x, y). Figure 2.2 shows an example of a TIN.

Another example of an irregular tessellation is the Voronoi diagram of a set of points in a d-dimensional space, where each point is mapped in a one-to-one way to a cell representing the locations in space that are closer to the point than to any other in the set. This is an advantage over raster and quad/octree structures where some cells do not contain any data points. See Figure 1.2(a) for a two-dimensional example, and the properties of the structure are extensively discussed in the next chapter. Perhaps the main advantage of irregular tessellations in the context of field modelling is that they permit us to preserve the samples, which are the only 'ground truth' of the field studied, and have even been referred to as the *meta-field* (Kemp and Vckovski, 1998).

A number of spatial models based on irregular tessellations have been proposed in the literature. Although the authors were not referring directly to fields, they are still subdividing the space covered by a three-dimensional object. Because the implementation of these are theoretically and technically much more difficult than for regular and hierarchical tessellations, many of these are simply theoretical discussion. Carlson (1987) proposes using triangles and tetrahedra for modelling subsurface structures, but does not consider many cases that would restrict the implementation, such as the presence of over-folds. Pigot (1992) also uses arbitrary cells of any shapes to tessellate 3D objects. Despite the flexibility of this model, it can be very difficult to implement from a data structure point of view and many spatial analysis operations are complicated by the fact that the cells are not convex. His work is based on a rather powerful data structure developed by Brisson (1989), which is described further in the section about data structures (Section 6.3). Tetrahedralizations, i.e. the decomposition of objects into tetrahedra, have also been used. Examples include Lattuada (1998) who uses tetrahedra in geological applications, and Pilouk (1996) who proposes using constrained tetrahedralizations to tessellate 3D objects. Implementing Pilouk's solution might prove to be difficult, as constrained tetrahedralizations are still at the research level, and are not possible for every 3D object (see Section 4.4.1).

2.2.2.4 Piecewise Functions

The function representing a field is piecewise when it is formed by different functions assigned to the different elements of a tessellation. The function within each element is usually a simple mathematical function, and as Goodchild (1992a) points out, this function can be constant, linear, or of a higher order. A constant function means that the value of the attribute is constant within one region, for example to represent a discrete field, as in a choropleth map. An example of the use of a linear function is a TIN: the spatial variation within each region (a triangle) is described by the linear function (a plane) defined by the three vertices lifted to their respective elevation. The value of the attribute of a field at an unsampled location x is thus obtained by linearly interpolating on the plane passing through the three vertices of the triangle containing x. Akima (1978) shows the advantages of using higher order functions in each region of a TIN—the main one being that the slope of the terrain is continuous everywhere, i.e. there are no discontinuities at the border of two triangles.

The same functions can obviously be used within each element in three dimensions.

2.2.3 Field Operations

Although fields are continuous in space and contrast with the object-view of space, the concept of object is still important as we can 'extract' discrete objects from a field. Indeed, implicit lines or surfaces representing the border between two different attributes can be extracted to help for visualisation or some other purpose. An example in two dimensions is a set of contours lines created from a field representing elevation, and in three dimensions a surface representing the front of a depression in atmospheric sciences.

Many arithmetic operations are also possible on fields, as demonstrated in the section about map algebra (Section 2.4.2).

2.3 Object-based Models

Although this thesis is concerned with the modelling of fields, it is still worth discussing the methods used in GIS to represent objects for several concepts are related (e.g. data structures). In an object-based view of the world, the space itself is empty and 'littered' with individual objects having their own attributes and location. Objects will 'interact' with each other (e.g. the location of each object will cause some to perhaps overlap, or be adjacent, or be contained within one another), and a GIS must represent each object, and also offer mechanisms to extract and represent the interactions (the topological relationships). In two dimensions, the representation of objects is usually done with three primitives: points, lines and polygons. To store all the objects in a certain portion of the plane, a variety of data structures with different properties have been developed and implemented. For instance, several GISs explicitly store the adjacency relationships between objects (e.g. TIGER (Boudriault, 1987) and ARC/INFO (Morehouse, 1985)), while some others use non-topological structures (e.g. the so-called spaghetti models) and reconstruct on the fly the spatial relationships when needed. Theobald (2001) examines the merits and drawbacks of both approaches.

In three dimensions, the same ideas can be applied, but one more primitive is needed: the polyhedron. Note that not all primitives need to be explicitly stored, as we can represent a 3D object only by its exterior surface, and thus use points, lines and polygons only. Most of the work on three-dimensional vector GIS is concerned with the modelling of cities, with for instance many buildings, roads and bridges. When designing three-dimensional spatial models for such applications, two issues must be tackled: (1) how each object should be represented; (2) how the topological relationships between objects are detected and stored. The former issue is usually solved by using a boundary representation (*b-rep*) of each object, and many data structures exist for that. I discuss in Section 6.3 several issues concerning data structures for storing three-dimensional subdivisions. The latter issue implies that two or more datasets containing objects are being 'overlaid' (or superimposed), and one needs to build the relationships between all the objects. Zlatanova et al. (2004) give an excellent summary of the spatial models that have been proposed for handling the geometry, the semantic and the spatial relationships between 3D objects. They also give an overview of the several frameworks used to detect relationships; most of them are similar to the 9-intersection model of Egenhofer and Herring (1990) and to its extension to three dimension (Egenhofer, 1995).

2.4 GIS Operations and Spatial Analysis

Although many programs can handle spatial data (e.g. CAD, scientific visualisation tools, statistical packages and database management systems), what sets GISs apart is their capabilities to perform spatial analysis (Goodchild, 1987). Spatial analysis operations include any geometrical, topological or statistical operations performed on spatial data (and their attributes) to produce information for decision-making. The result of a spatial operation is dependent on the locations of the objects studied; if the objects involved in a query moved, then the result of query would be different.

The GIS operations in two dimensions are well-known, and many are available in commercial systems. A few taxonomies of GIS operations exist, e.g. see Aronoff (1991) or Goodchild (1987), but are limited by the fact that they are data structure dependent (vector or raster). Admittedly, not every operation is possible on any kind of data (e.g. interpolation is meaningless without a field, so is a buffer zone without discrete objects), but, in my opinion, the data structure used to represent geographical reality should not drive the taxonomy of operations.

Albrecht (1996) recognises that and proposes a taxonomy that is, to my knowledge, the most relevant one in the context of this thesis, albeit it is for the two-dimensional case only. The other set of operations relevant to this thesis is the *map algebra* framework (Tomlin, 1983), which comprises tools to model and analyse different fields stored in a raster format.

2.4.1 Albrecht's Classification

Albrecht (1996) proposes a taxonomy of GIS operations mainly to help in the integration of environmental modelling within GIS. He classifies "20 universal GIS operators that allow to build all but the most exotic GIS applications" and separates them into six categories:

1. **Search**: interpolation, (re)classification, thematic and spatial search;
2. **Location Analysis**: buffer zone, overlay, and creation of Voronoi diagram;
3. **Terrain Analysis**: slope/aspect, viewshed, catchments/basins, and drainage;
4. **Distribution/Neighbourhood**: cost, proximity, and nearest neighbour;
5. **Spatial Analysis**: statistical and pattern analysis, shape;
6. **Measurements**: distance, area and centroid.

All of these are relevant in the context of modelling fields because, as mentioned in Section 2.2.3, discrete objects can be extracted from fields. It should be noticed that most of these operators take into consideration the attributes attached to each object or location, but that some operators consider only the location information. Interpolation, classification, thematic search, terrain analysis, and distribution belong to the former category; while buffer zone, pattern analysis, shape and measurement belong to the latter. Notice also that new objects are created by some operators, most notably the Voronoi diagram and buffer zone operations. The overlay operator encompasses all the usual 'topological operators', as defined by Egenhofer and Herring (1990): union, intersection, containment, adjacent, etc.

All the operations theoretically generalise to three dimensions, except the ones concerned with terrain analysis. Many of these are discussed in Chapter 5.

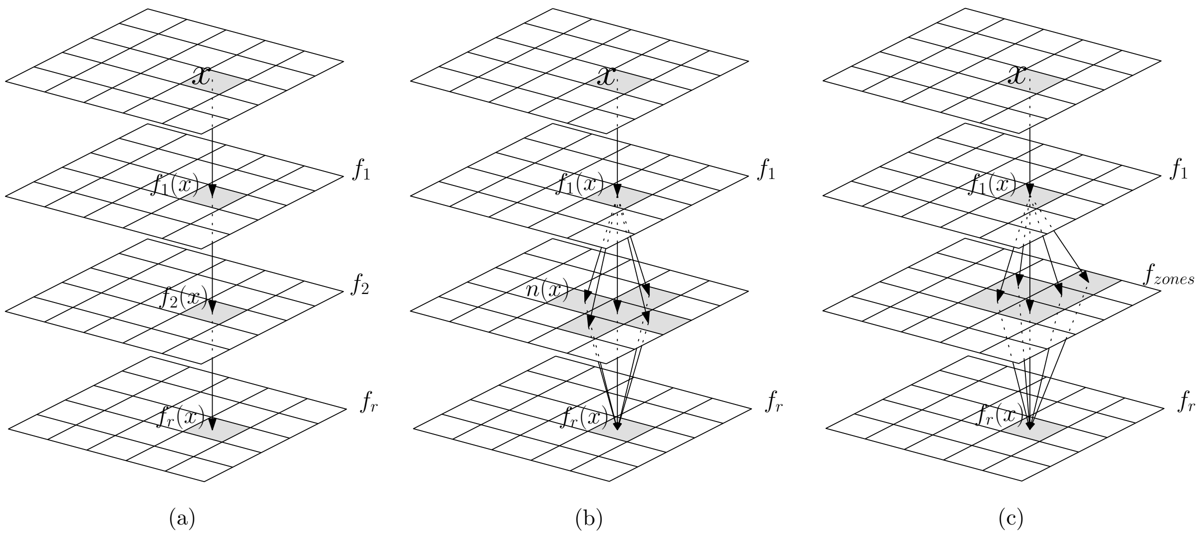


Figure 2.7: The map algebra operations with a raster structure. **(a)** A binary local operation. **(b)** An unary focal operation. **(c)** A zonal operation that uses a set of zones (f_{zones}) stored as a grid.

2.4.2 Map Algebra[2]

Map algebra refers to the framework, first developed and formalised by Tomlin (1983), to model and manipulate fields stored as raster structures. It is called an algebra because each field (also called a map) is treated as a variable, and complex operations on fields are formed by a sequence of primitive operations, like in an equation (Berry, 1993). A field operation always takes a field (or many fields) as input and returns a new field as output (the values of the new field are computed location by location). Operations can be unary (input is a single field), binary (two fields) or n-ary (n fields); I describe here only the unary and binary cases because n-ary operations can be obtained with a series of binary operations. Tomlin (1983) describes three categories of operations:

Local operation: (Figure 2.7(a)) the value of the new field at location x is based on the value(s) of the input field(s) at location x. An unary example is the conversion of a field representing the elevation of a terrain from feet to metres. For the binary case, the operation is based on the overlay in GIS: the two fields f_1 and f_2 are superimposed, and the result field f_r is pointwise constructed. Its value at location x, defined $f_r(x)$, is based on both $f_1(x)$ and $f_2(x)$. An example is when the maximum, the average or the sum of the values at each location x is sought.

Focal operation: (Figure 2.7(b)) the value of the new field at location x is computed as a function of the values in the input field(s) in the neighbourhood of x. As Worboys and Duckham (2004) describe, the neighbourhood function $n(x)$ at location x associates with each x a set of locations that are 'near' to x. The function $n(x)$ can be based on distance and/or direction, and in the case of raster it is usually the four or eight adjacent pixels. An unary example is the derivation of a field representing the slope of a terrain, from an elevation field.

[2]This section is based on Ledoux and Gold (2006a).

Zonal operation: (Figure 2.7(c)) given a field f_1 and a set of *zones*, a zonal operation creates a new field f_r for which every location x summarises or aggregates the values in f_1 that are in a given zone. The set of zones is usually also represented as a field, and a zone is a collection of locations that have the same value (e.g. in a grid file, all the adjacent cells having the same attribute). For example, given a field representing the temperature of a given day across Europe and a map of all the countries (each country is a zone), a zonal operation constructs a new field such that each location contains the average temperature for the country.

Although the operations are arguably simple, the combination of many makes map algebra a rather powerful tool. It is indeed being used in many commercial GISs, albeit with slight variations in the implementations and the user interfaces (Bruns and Egenhofer, 1997).

It should also be noticed that the three categories of operations are not restricted to the plane, and are valid in any dimensions; Mennis et al. (2005) have recently implemented them with a voxel structure.

Despite its popularity, the biggest handicap to the use of map algebra is arguably that is was developed for regular tessellations only, although the concepts are theoretically valid with any tessellation of space (Takeyama, 1996; Worboys and Duckham, 2004). The shortcomings of raster structures to represent fields, as described earlier, all apply for map algebra. A further important consideration is that in order to perform binary operations, the two grids must 'correspond', i.e. the spatial extent, the resolution and the orientation of the two grids must be the same, so that when they are overlaid each pixel corresponds to one and only one pixel in the other grid. If the grids do not correspond, then *resampling* of one grid (or both) is needed. This involves the interpolation of values at regularly distributed locations with different methods such as nearest neighbour or bilinear interpolation, but unfortunately each resampling degrades the information represented by the grid (Gold and Edwards, 1992).

2.4.3 GIS Operations in Three Dimensions

The GIS operations are not as studied in three dimensions and no classification as complete as Albrecht's is known. Ideally, a 3D GIS should be able to perform the same operations as in two dimensions. As stated, most of the two-dimensional GIS operators generalise to three dimensions straightforwardly.

To my knowledge, the only attempt to classify the 3D GIS operations is a very generic classification by Raper (1992). Since it is notorious that volumetric data are difficult to visualise, he includes visualisation functions as one of the category. His other operators are: transformation (shearing); selection (based on intersection of spatial objects); relationships (metric and topological); characterisation (volume, surface, centre of mass); and modelling (building of complex objects from primitives).

I propose in Section 5.1 an extension to three dimensions of the classification of Albrecht (1996).

2.5 Dynamic and Interactive Modelling

The term 'dynamic' has different meanings depending in which discipline it is being used. In a GIS context, it usually refers to the movement of some objects, so it is closely related to the modelling of time. By contrast, a dynamic data structure in computer science is one that permits the insertion and the deletion of objects, without the need of reconstructing from scratch the whole data structure. A simple example is a linked list, because elements can be added or deleted anywhere in the list, and the updating is efficient. Data structures permitting the update of the data structure while objects are in motion, e.g. a set of points moving in the plane and we want to keep at all times the spatial relationships between them up-to-date, is referred to a *kinetic* structure in computer science.

Dynamism is essential in a context of *interactive modelling* (Anselin, 1999; Bailey and Gatrell, 1995; Gold, 1993). This is an approach where the user does not only use standard operations (e.g. GIS spatial analysis or statistical methods) on a dataset, but actually interacts with it by manipulating and transforming it and then looks at the consequences of his manipulations. The emphasis is put on the interaction between the user and the dataset through modification and visualisation tools. A simple example of this approach is a spreadsheet tool (e.g. Microsoft Excel) where numerical data can be explored with the help of different statistical tests and charts (histograms, scatter points, pie charts, etc.). These charts are linked to the data, i.e. if some numbers in the dataset are changed, the charts are automatically updated.

In the context of geoscientific modelling, one can think of an environment where the samples can be inserted, deleted or even moved at will by the user, and he gains insight about the spatial variation of the field studied by manipulating the dataset and observing the results. A key factor in interactive analysis is the speed with which each function is performed, to ensure that the user gets an (almost) instantaneous result from a query or an operation, so that he is not disturbed by long waits.

2.6 Handling Time within a GIS

2.6.1 Is Time Another Dimension?

As in the case of integrating the third dimension in a GIS, the integration of time is rather complex. Although several solutions have been proposed over the years (see Langran (1992) and Peuquet (2001) for excellent surveys), the representation of both space and time in the same system is still problematic, and no fully functional systems are known. As Couclelis (1999) points out, time has been historically linked to space, to the extent that it is often considered as another dimension, one that is orthogonal to the three dimensions of space. This approach, which yields a four-dimensional structure, is supported both by physics and mathematics: it is called a Minkowski space and was used by Einstein to formulate the special theory of relativity. However, as Galton (2004) argues, the special theory of relativity tries to explain the workings of the universe and is somewhat irrelevant to the context of modelling temporal data at the surface of the Earth. The full integration of time within a spatio-temporal coordinate system is mostly a theoretical discussion as in practice, no matter if time is fully integrated or not, objects will be located in three-dimensional space and in time. By contrast, Peuquet (1994b) warns that a homogeneous four-dimensional representation might not be wished since space and time exhibit important differences. The main differences being that the scale (or units) of time is totally different from the scale of space, and also that discretising time is not a simple issue. These problems can probably be solved for specific applications, but generic rules are difficult to devise.

In brief, space and time can theoretically be integrated together, but in practice in a GIS context, things are more complicated. Adding a time dimension to a 2D GIS does not necessarily make it 3D, in the sense that the geometry and topology of a 2D problem does not necessarily become 3D when the factor time is added. It should be noticed that in a GIS context, the terms 'four-dimensional' and 'multi-dimensional' are nevertheless often being used. Indeed, attributes are often considered as an another dimension (as in a TIN), and scale as even been considered as another dimension (Li, 1994).

2.6.2 Past or Future?

In the discussion about the integration of time in a GIS, two concepts contrast:

1. One is interested in analysing the state of a certain region at a given time, in the past. This presupposes that data have been collected at different times, and one wants for instance to analyse changes and identify trends.
2. One possesses a set of data at a certain time and would like to predict what will happen in the future, given a certain number of hypotheses. This has been referred to as 'What-if? analysis', or simulation.

Both concepts can use similar techniques, but there can also be important differences. Representing changes to be able to answer questions such as 'Where was such-and-such object and what was its attributes at time t?' is mostly a database problem. On the other hand, the second concept involves the movement of objects or the simulation of a process, such as a forest fire or the tides.

2.6.3 Representing Changes

Although GISs were essentially developed for the storage, representation and manipulation of spatial information, different methods have been proposed in recent years to represent changes. To integrate time, most of the methods developed are based on the extension of the existing spatial models in commercial GISs, as it has been the case when integrating the third dimension. Since there were no mechanisms to represent time, the first method used was rather trivial: time was simulated by showing a series of separate *snapshots*, where each snapshot is a representation of a certain region (a GIS layer) at a certain time (Armstrong, 1988). Every layer has a date associated with it and every object on this layer is affected. This model has many problems: the same object can be duplicated in two layers—which causes data redundancy and possibly data inconsistency—, enormous data storage can be needed, and it is impossible to know when a change occurred, we just know that it was between the times associated with two layers. To solve these problems, the idea of *date-stamping* each piece of information instead of the whole layer was developed. The attributes of every location (in a raster) or the objects themselves (in a vector representation) are linked to a list that contains the chronological changes that happened (the list can be thought of being perpendicular to the two spatial dimensions). Examples of this method are the models of Langran (1992), Peuquet (1994a) and Worboys (1992).

The previous models add the time factor to a GIS, but none of them can maintain the topological relationships valid after a change. The emphasis is put on ways to maintain consistently the database, but how the spatial relationships among objects are maintained is usually not covered. Processes like the movement of an object over time can not be modelled with snapshots because it is simply too expensive. It has been argued that the problem of adding dynamic processes to a GIS will not be solved by simply modifying the database linked to the system since the biggest handicap of commercial GIS is the definition of topology (Burrough, 1992; Gold, 1991; Peuquet, 1994b). Gold (1996) and van Oosterom (1997) both propose models where the updating of the topology and the database is performed with models that are not based on the usual polygon-arc-node models of commercial GISs. Representing continuous changes is possible with these models: local updates are performed as soon as an event occurs. van Oosterom achieves this with a topological data structure, similar to the *winged-edge* (Baumgart, 1975), where insertion and deletion of objects are possible. Gold's spatial model is based on the Voronoi diagram and it permits local updates when objects are inserted, deleted or even moved. It is possible to rebuild every topological state of a map since all the operations are reversible. I give more details about the algorithms to modify a VD in Chapter 4.

Models for handling time within a 3D GIS are rare probably because 3D spatial models are still in their infancy and are useful only in a few application disciplines, and also because of the technical difficulties arising with the implementation of 3D algorithms. Pigot and Hazelton (1992) propose a theoretical method, based on the spatial model of Pigot (1992), for representing space and time. Since the time is represented as an extra dimension, the spatio-temporal objects are embedded in four-dimensional space. Different snapshots of a dataset are kept and the spatio-temporal objects in it are formed by joining the cells between different times, thus creating four-dimensional cells. They state that these spatio-temporal objects can be stored with the data structure developed by Brisson (1989) and that the topological framework of Egenhofer (1995) can be extended to four dimensions. Also, Mason et al. (1994) and O'Conaill et al. (1992) describe two similar systems for three-dimensional oceanographic and atmospheric data, and recognise the importance of integrating time in a GIS to improve environmental simulations. Their system treats the four dimensions equally but instead of using cells, they use a raster representation that is implemented as a four-dimensional hypercube; the data structure used is called a *bintree*, which is a hierarchical structure similar to the octree. Varma (1999) also propose a similar model, based on the helical hyperspatial code, which is also a hierarchical tessellations. Finally, Raper and Livingstone (1995) discuss a 4D spatial model based on object-oriented techniques for a specific problem in geomorphology, but the concepts can be used for other applications where processes need to be modelled. The spatial model is vector-based as different objects are modelled. Their discussion is mostly theoretical, and details concerning the implementation (how the spatial relationships are defined) are not covered.

A further important consideration is that the map algebra framework, as described in Section 2.4.2, can also be considered as a mean to model changes as fields collected at different times can be analysed, and information extracted.

2.6.4 Simulations

Many disciplines need to simulate real-world processes, and the methods they use obviously differ. This section is not intended to be a thorough review of the methods used, but rather a short summary of the most relevant for the kind of data this thesis is concerned with.

Simulations in engineering or in geoscience 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) (Burrough et al., 1988). On the other hand, researches oriented more towards social sciences (e.g. urban growth or transportation) have used mostly *cellular automata* (CA) (Wolfram, 1986). Both methods are related as they require the partitioning of the space into discrete cells. PDEs are solved, or rather approximated, with mainly two numerical methods: the *finite difference method* (FDM), which was developed for regular tessellations; and the *finite element method* (FEM) (Strang and Fix, 1973), which is possible on any tessellations, regular or irregular. The solution of a PDE is obtained by firstly approximating the behaviour of the process studied for each element of the tessellation, and the final solution is obtained by accumulating all the results. By contrast, a cellular automaton is a discrete model of a process (both in space and in time), and it consists of a tessellation (usually regular, although it is possible to use CA on irregular tessellation) where each cell has a *state* (the attribute given to each cell). Dynamic phenomena are modelled discretely, i.e. the time is discrete, and the state of a cell at time t is defined uniquely by the states of its neighbouring cells at time $(t - 1)$. CA are theoretically valid in any dimensions, but most of the GIS-related work is in two dimensions, see for example Batty et al. (1997), Couclelis (1985), Takeyama and Couclelis (1997) and De Vasconcelos et al. (2002).

For the modelling of fluid flow, there exists an alternative approach to using a fixed/rigid tessellation: the *Free-Lagrange method* (FLM), where the flow is approximated by a set of discrete points (called particles) that are allowed to move freely. A tessellation of the space is still required (based on the particles), but of course it will be modified as the particles are moving. As already explained briefly, the Voronoi diagram 'naturally' tessellates the space based on a set of points, and has therefore been used (Erlebacher, 1985). Although the FLM theoretically can represent better a physical process (Fritts et al., 1985), it is handicapped by the difficulties encountered when implementing it. As Mostafavi and Gold (2004) 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. They propose using the VD of a set of moving points (on the sphere), and the tessellation does not have to be reconstructed after each move as all the operations are local, as in the work of Roos (1991); more details about the movement of points in a VD are given in Section 4.5.

The integration of process modelling with a GIS is a source of problems because GISs have not been designed to handle time, even less processes which involves continual movement (Sui and Maggio, 1999). Full integration is almost not heard 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), but the simulation itself is done using a specialised tool. Many argue for 'full integration', as both tools would ultimately gain (Freda, 1993; Parks, 1993).

2.7 Visualisation

This section briefly presents the basics of three-dimensional visualisation. More details can be found in any good computer graphics book, such as the one by Hill (2000).

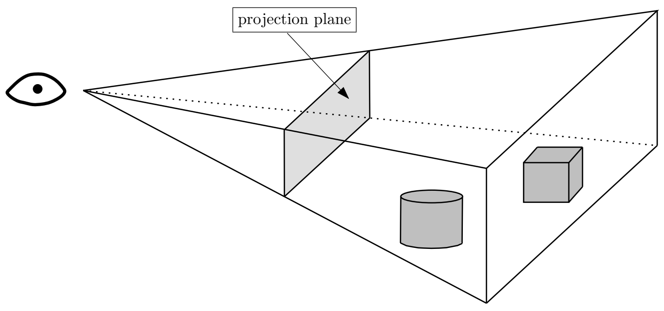


Figure 2.8: Geometry of a perspective projection.

The basic idea of 3D visualisation is to project three-dimensional objects—formed by points, line segments and planes—in a two-dimensional plane, the computer screen. Computer screens have a raster display, that is a grid of pixels where each pixel can be assigned one and only one colour. The process of projecting the objects is therefore split into two steps: the projection in the plane and the rasterisation to assign one colour to each pixel. There exist many kinds of projection, but a perspective projection (see Figure 2.8) will produce an image that mimics how the eyes see the objects, i.e. the closer to the viewpoint an object is, the bigger it will be in the plane. A perspective view is defined by a *viewing volume*, that is a truncated rectangular pyramid that defines what portion of a scene should be projected in the 2D plane. There exist different topics related to the rendering of 3D objects, for example *level of detail* algorithms will vary the details in a scene to accelerate the rendering of the objects; hidden-surface removal algorithms will determine which object in a scene is the closest to the viewpoint so the colour of this one will be assign to the corresponding pixel; and algorithms for computing the shading and the illumination of a scene are also necessary to have realistic views. The `OpenGL` library, an interface to the graphics cards, greatly helps the process of rendering a scene by providing the basics tools to perform the operations previously mentioned—`OpenGL` is not the only graphical library, but a popular one since it is platform independent.

A three-dimensional perspective view with ‘navigation tools’ (zoom in/out, translation and rotation) is sufficient to visualise surfaces, but not enough to explore a volumetric dataset (data in three dimensions plus an attribute). Several extra techniques can help, as explained in Section 5.4.

2.8 Why Do GISs Lack What Is Needed to Model Trivariate Fields?

Most of the techniques presented in this chapter have drawbacks for the representation and the modelling of three-dimensional fields, as found in the Earth sciences. The object-based models are clearly not appropriate, and what is needed is a tessellation. The wide popularity of regular (voxels) and hierarchical tessellations (octrees) is probably due to their simplicity: a simple data structure can be used to store them, and their manipulation is also simple. Unfortunately this simplicity has a hefty price. Rasters can become huge, do not scale well, force an unnatural discretisation of continuous phenomena and imply a fair amount of preprocessing of datasets, which is usually hidden to the user. Octrees improve that but still have many shortcomings. Arbitrary tessellations into polyhedra are an attractive solution, but the construction and the manipulation of such a spatial model are cumbersome and rather slow.

What is needed is a three-dimensional spatial model that adapts, or takes into consideration, the phenomena being modelled, and also that permits us to modify it efficiently in order to do interactive and temporal modelling. As briefly mentioned in this chapter, a tessellation of the space into Voronoi regions seems like a perfect solution in our case. VDs in two dimensions have been used in different GIS-related projects, most notably for a prototype of a Marine GIS (Gold and Condal, 1995; Gold, 1999) where the needs are similar to the modelling of fields in geoscience. A spatial model based on the three-dimensional VD would permit us to accurately and efficient reconstruct a field from the samples that were collected to study it. By assigning different functions to each cell, it would be possible to model both discrete and continuous fields.

I argue in this thesis that the Voronoi tessellation has many advantages over other tessellations for representing fields. First, it gives a unique and consistent definition of the spatial relationships between unconnected points (the samples). As every point is mapped in a one-to-one way to a Voronoi cell, the relationships are based on the relations of adjacency between the cells. Second, the size and the shape of Voronoi cells are determined by the distribution of the samples of the phenomenon studied, thus the VD adapts to the distribution of points. This is particularly interesting because geoscientific datasets usually have highly anisotropic distribution. Third, as demonstrated in Chapter 4, we can add, delete and even move points in a VD without reconstructing the structure from scratch. Fourth, the dual of the VD, the DT, can also be of great help for the manipulation and the analysis of fields.

Although the algorithms to manipulate and analyse a three-dimensional VD are admittedly more complex than the ones for raster structures, I argue that the benefits of using the VD compensate largely, as the rest of this thesis demonstrates.

Chapter 3

State-of-the-art Computational Geometry

Computational Geometry is a branch of computer sciences that involves the design and analysis of efficient algorithms for solving geometric problems. It emerged in the late 1970s from the field of algorithmic design, and is now a discipline on its own with conferences and journals. It is popular because a great many problems are geometric by nature, and because the results of computational geometry can be applied in a variety of disciplines, for instance computer graphics, CAD, engineering, GIS, robotics, etc.

Arguably the most important structures in computational geometry are the Voronoi diagram (VD) and the Delaunay triangulation (DT). The two structures have been discovered, rediscovered and studied many times and in many different fields (see Okabe et al. (2000) for a complete history). The VD can be traced back to 1644, where Descartes used Voronoi-like structures in Part III of his *Principia Philosophiæ*. The VD in two and three dimensions was used by Dirichlet (1850) to study quadratic forms—this is why the VD is sometimes referred to as *Dirichlet tessellation*—but was formalised and defined for the d-dimensional case by Voronoi (1908). The first use of the VD in a geographical context is due to Thiessen (1911), who used it in climatology to better estimate the precipitation average around observations sites. The DT, which is the dual of the VD, was formalised for the d-dimensional case by Delaunay (1934); notice that his name is sometimes spelled 'Delone'. Many computer scientists and mathematicians consider the VD as being the most fundamental spatial structure (or spatial model) because it is very simple, and yet is so powerful that it helps in solving many theoretical problems, and helps in many real-world application; Aurenhammer (1991) and Okabe et al. (2000) offer exhaustive surveys.

This chapter formally defines the concepts of VD and DT in three dimensions, and introduces several concepts in computational geometry and combinatorial topology that are needed to understand the next two chapters about respectively the manipulation of a three-dimensional VD and the implementation of GIS operations based on the VD/DT. The definitions of most of these concepts can be found in any good computational geometry book, such as Preparata and Shamos (1985), Boissonnat and Yvinec (1998), de Berg et al. (2000), or Edelsbrunner (2001).

3.1 Mathematical Preliminaries

3.1.1 Graphs

In mathematics and computer science, a graph $G = (V, E)$ is defined as a set of vertices V (also called nodes) together with a set of edges E joining two distinct vertices. Depending on

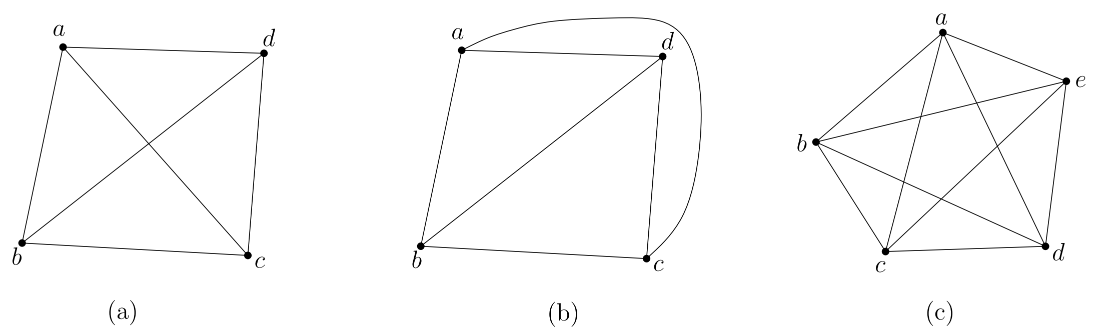


Figure 3.1: (a) The graph has two intersecting edges, but it is still a planar graph since it can be drawn without any intersection, as in **(b)**. **(c)** An example of a non-planar graph.

the application, an edge $e = \{a, b\}$ joining two distinct vertices $a, b \in V$ can be *directed* or not. Every edge in an *undirected graph* is formed by an unordered pair of vertices a and b, so that $e = \{a, b\}$ and $e = \{b, a\}$ represent the same edge. Weights can also be assigned to each edge of a graph, i.e. a value is associated to an edge. This permits many real-world applications, for instance the calculation of the optimal route between two locations (the weight can be the distance between the locations, the time it takes to drive from one to the other, etc.). Notice that the graphs used in this thesis are always unordered and unweighted. To simplify the notation, I will simply use ab to refer to an edge e formed by the vertices a and b.

An important concept here is that of a *planar graph*: a graph is said planar when it can be embedded in the Euclidean plane, and preserve its structure, so that no edges intersect (except at nodes). Figure 3.1 shows examples of planar and non-planar graphs. Notice that the graph in Figure 3.1(a), although having intersecting edges, is planar for it can be drawn without the intersection (Figure 3.1(b)). Also, observe that a graph $G = (V, E)$ can be embedded in higher dimensions, if for example V is a set of vertices having three-dimensional coordinates, and the edges in E are formed, like in the two-dimensional case, by pairs of vertices.

Another important concept is that of the *degree* of a vertex a, which is equal to the number of edges formed by a and any other vertices in V (the number of incident edges to a).

To traverse or search a graph, mainly two algorithms are used: *breadth-first search* (BFS) and *depth-first search* (DFS). The former algorithm starts at a given vertex v, and explores all the adjacent vertices. Then, for these, it explores all their adjacent vertices until all the vertices have been explored, or the goal is found. The BFS is similar, although it will start with one adjacent vertex, and continue exploring another adjacent vertex until it is not possible to explore further. Then another adjacent vertex to the starting vertex will be explored.

A *tree* is a special case of a graph in which any two vertices are connected by exactly one path (there are no cycles). Figure 2.5 shows one example of a tree.

3.1.2 Simplices and Simplicial Complexes

Given a set S of points in the plane, if the *maximum planar graph* is constructed, a triangulation of S is obtained. A maximum planar graph is a graph that contains the maximum possible number of edges, if another edge connecting any pairs of nodes would be inserted, then it would intersect an already existing edge. A maximum planar graph can therefore be seen as

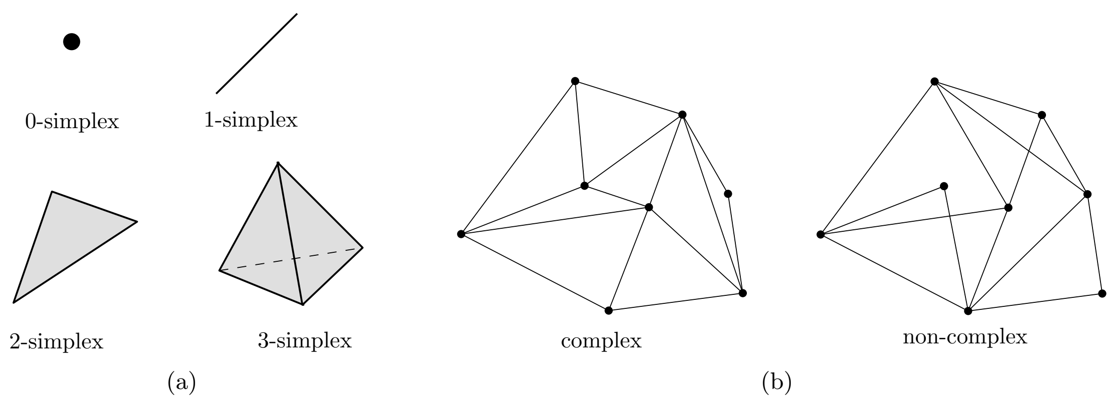


Figure 3.2: (a) Examples of simplices in different dimensions. **(b)** A simplicial complex on the left, and a structure that is not a complex on the right.

a *simplicial complex*. Informally, this is a structure that is built from a set of simple elements called *simplices*; in the case of a triangulation, the elements are nodes, edges and triangles (notice that the triangles are represented *implicitly* by the graph). A simplex is the simplest element in a given dimension. Specifically, a k-dimensional simplex is called a k-simplex and it is the convex hull (see Section 3.1.4) of a set of $(k + 1)$ linearly independent points in $\mathbb{R}^k$. As shown in Figure 3.2(a), a 0-simplex is a node, a 1-simplex an edge, a 2-simplex a triangle, a 3-simplex a tetrahedron, and so on. Observe that in $\mathbb{R}^d$, k-simplices (where $0 \leq k \leq d$) can exist. Also to simplify the notation, a k-simplex σ formed by the vertices a, b and c, is simply denoted abc.

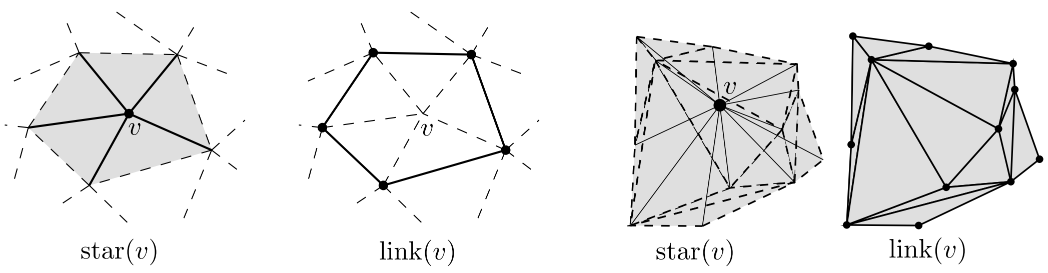
Notice that a k-simplex can be constructed by simplices of lower dimensionality. Let σ be a k-simplex, then the *facets* of σ are the $(k - 1)$-simplices forming it. Two distinct simplices σ_1 and σ_2 (of the same dimensionality) are said to be *adjacent* if they share a facet. Also, two simplices σ_1 and σ_2 are said to be *incident* if one is a facet of the other (thus σ_1 and σ_2 are not of the same dimensionality).

A simplicial complex C is a finite set of simplices having the following two conditions:

1. Any facet of a simplex in C is also in C;

2. The intersection of two simplices σ_1 and σ_2 in C, denoted $\sigma_1 \cap \sigma_2$, is either empty or is a facet of both σ_1 and σ_2.

In brief, a simplicial complex is a space-filling structure, where its constituent simplices are not allowed to puncture the interior of others simplices (see Figure 3.2(b)). The dimension of a simplicial complex is given by the maximum dimension of its simplices. A two-dimensional simplicial complex is a triangulation, and in three dimensions it is formed by tetrahedra, and it is called a tetrahedralization (also called a tetrahedrization sometimes). Notice that the term “triangulation” is sometimes used as the general term for a simplicial complex in $\mathbb{R}^d$, where $d \geq 2$. A simplicial complex in $\mathbb{R}^d$ can be represented by a graph where the vertices and edges are embedded in $\mathbb{R}^d$ such that a set of vertices and edges implicitly represent a k-simplex.



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

Figure 3.3: The star and the link of a vertex v.

3.1.2.1 Stars and Links

The *star* and the *link* of a vertex in a d-dimensional triangulation are important concepts that will be used later in this thesis. Let v be a vertex in a d-dimensional triangulation (where $d \geq 2$). Referring to Figure 3.3, 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.

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 $(d-1)$-dimensional triangulation (if $d = 2$ then the link is a 1-dimensional triangulation, or simplicial complex, which is a closed polygon). 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.

3.1.2.2 Cells

The notion of simplex can be slightly modified to obtain a *cell*: a 0-cell is a vertex, a 1-cell an edge, a 2-cell a polygon, and a 3-cell a polyhedron. Thus, a k-cell is a shape in $\mathbb{R}^k$, but not necessarily the simplest possible.

The same concept applies to simplicial complexes: *cell complexes* are obtained, and the complex formed must respect the same two conditions as a simplicial complex. The notions of incidence and adjacency are also the same as for the simplicial case.

3.1.3 Manifolds

A manifold is a topological space that is *locally* like a Euclidean space. The neighbourhood of every point of a d-dimensional manifold (a d-manifold) resembles $\mathbb{R}^d$; in mathematics terms, the neighbourhood of a point is said to be *homeomorphic* to a ball in $\mathbb{R}^d$. A 2-manifold is therefore a surface which, it should be pointed out, can be embedded in three-dimensional space (see Figure 3.4(a)); an obvious example is the surface of the Earth, for which near to every point the environment is equivalent to a plane. An example of a 3-manifold is the entire

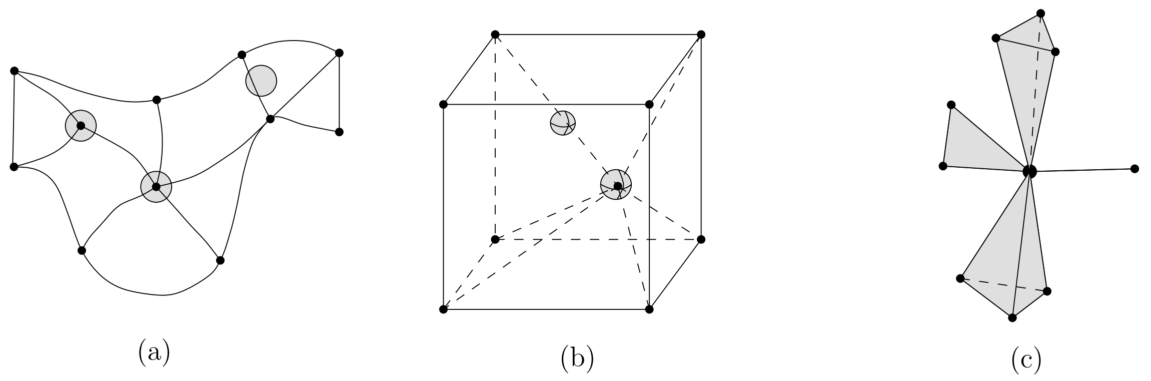


Figure 3.4: **(a)** The neighbourhood of every point in a 2-manifold is equivalent to a plane. **(b)** The neighbourhood of every point in a 3-manifold is equivalent to a 3D space. **(c)** A non-manifold object. The neighbourhood of the vertex where the two tetrahedra, the triangular face and the edge meet does not resemble $\mathbb{R}^3$, i.e. a coordinate system can not be defined locally.

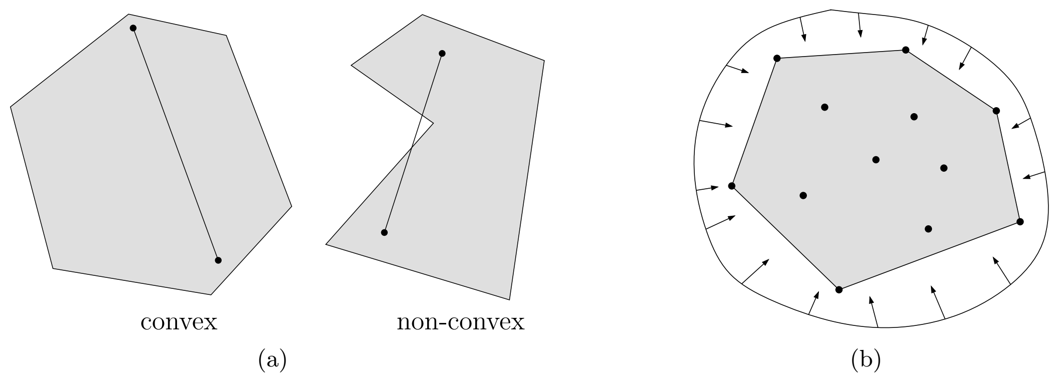


Figure 3.5: **(a)** A convex and a non-convex object. **(b)** The convex hull of a set of points in $\mathbb{R}^2$.

Earth (its interior) because the neighbourhood of every point is equivalent to a sphere (see Figure 3.4(b)). The concept of neighbourhood, or locality, is such that a manifold can actually be constructed by 'gluing' separate Euclidean spaces together. A case where an object is not a manifold is shown in Figure 3.4(c).

In mathematics and topology, there exist different classes of manifolds, and their definitions are fairly technical. In the context of this thesis, the definition given here is sufficient. Manifolds will be discussed mostly in the section about the possible data structures to store the VD/DT (Section 6.3).

3.1.4 Convexity

An object is *convex* if for any pair of points within the object the line segment joining the two points is completely contained in the object (see Figure 3.5(a)). Let S be a set of points in $\mathbb{R}^d$, the *convex hull* of S, denoted conv(S), is the minimal convex set containing S. It is best understood with the elastic band analogy: imagine each point in $\mathbb{R}^2$ being a nail sticking out of the plane, and a rubber band stretched to contain all the nails, as shown in Figure 3.5(b). When released, the rubber band will assume the shape of the convex hull of the nails. Notice

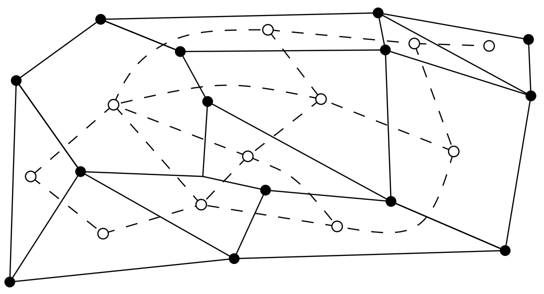


Figure 3.6: A graph G (black lines), and its dual graph $G^\star$ (dashed lines).

that conv(S) is not only formed by the edges connecting the points (the rubber band), but all the points of $\mathbb{R}^2$ that are contained within these edges.

The three-dimensional case is analogous, but somewhat more complicated to visualise. The convex hull of a set of points in $\mathbb{R}^3$ is a polyhedron formed by triangular faces.

Convexity is a simple but useful concept, and that in a variety of applications. The convex hull of a set of points helps to compute other structures, and manipulating objects that are convex simplifies tremendously many operations: the intersection of two convex objects yields a convex object, the calculation of areas/volumes is straightforward and so is the "point-in-polygon" problem, etc.

3.1.5 Duality

Duality can have many different meanings in mathematics, but it always refers to the translation or mapping in a one-to-one fashion of concepts or structures. I use it here in the sense of the dual graph of a given graph. Let G be a planar graph, as illustrated in Figure 3.6 (black edges). Observe that G can also be seen as a cell complex in $\mathbb{R}^2$. The duality mapping is defined as follows. The dual graph $G^\star$ has a vertex for each face (polygon) in G, and the vertices in $G^\star$ are linked by an edge if and only if the two corresponding dual faces in G are adjacent (in Figure 3.6, $G^\star$ is represented with dashed lines). Notice also that each polygon in $G^\star$ corresponds to a vertex in G, and that each edge of G is actually dual to one edge (an arc in Figure 3.6) of $G^\star$ (for the sake of simplicity the dual edges to the edges on the boundary of G are not drawn).

The concept of duality is valid in any dimensions, if we consider a graph embedded in $\mathbb{R}^d$ as a d-dimensional cell complex. The mapping between the elements of a cell complex in $\mathbb{R}^d$ is simple: let C be a k-cell, the dual cell of C in $\mathbb{R}^d$ is denoted by $C^\star$ and is a $(d-k)$-cell. As as result, in three dimensions, a point becomes a polyhedron, and vice versa; and an edge becomes a face, and vice versa.

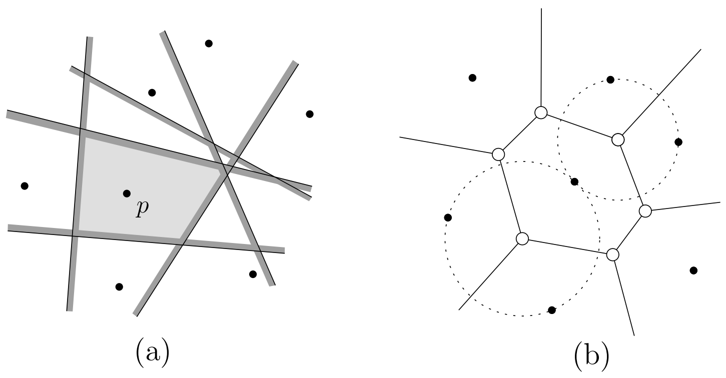


Figure 3.7: (a) The Voronoi cell $\mathcal{V}_p$ is formed by the intersection of all the half-planes between p and the other points. **(b)** The VD for a set S of points in the plane (the black points). The Voronoi vertices (white points) are located at the centre of the circle passing through three points in S, provided that this circle contains no other points in S in its interior.

3.2 The Voronoi Diagram and the Delaunay Triangulation

3.2.1 The Voronoi Diagram

Let S be a set of points in $\mathbb{R}^d$. The Voronoi cell of a point $p \in S$, defined $\mathcal{V}_p$, is the set of points $x \in \mathbb{R}^d$ that are closer to p than to any other point in S; that is:

$$\mathcal{V}_p = \{x \in \mathbb{R}^d \mid \|x - p\| \leq \|x - q\|, \; \forall \, q \in S\}. \tag{3.1}$$

The union of the Voronoi cells of all generating points $p \in S$ form the Voronoi diagram of S, defined VD(S). If S contains only two points p and q, then VD(S) is formed by a single hyperplane defined by all the points $x \in \mathbb{R}^d$ that are equidistant from p and q. This hyperplane is the perpendicular bisector of the line segment from p to q, and splits the space into two (open) half-spaces. $\mathcal{V}_p$ is formed by the half-space containing p, and $\mathcal{V}_q$ by the one containing q. As shown in Figure 3.7(a), when S contains more than two points (let us say it contains n points), the Voronoi cell of a given point $p \in S$ is obtained by the intersection of $n - 1$ half-spaces defined by p and the other points $q \in S$. That means that $\mathcal{V}_p$ is always convex, in any dimensions. Notice also that every point $x \in \mathbb{R}^d$ has at least one nearest point in S, which means that VD(S) covers the entire space.

As shown in Figures 3.7(b), the VD of a set S of points in $\mathbb{R}^2$ is a planar graph. Its edges are the perpendicular bisectors of the line segments of pairs of points in S, and its vertices are located at the centres of the circles passing through three points in S. The VD in $\mathbb{R}^2$ can also be seen as a two-dimensional cell complex where each 2-cell is a (convex) polygon (see Figure 3.8(a)). Two Voronoi cells, $\mathcal{V}_p$ and $\mathcal{V}_q$, lie on the opposite sides of the perpendicular bisector separating the points p and q.

In $\mathbb{R}^3$, VD(S) is a three-dimensional cell complex. The Voronoi cell of a point p is formed by the intersection of all the half-spaces (three-dimensional planes) between p and the other points in S. Drawing a picture of the three-dimensional case is not easy, thus Figure 3.8(b) shows only two Voronoi cells (which are convex polyhedra).

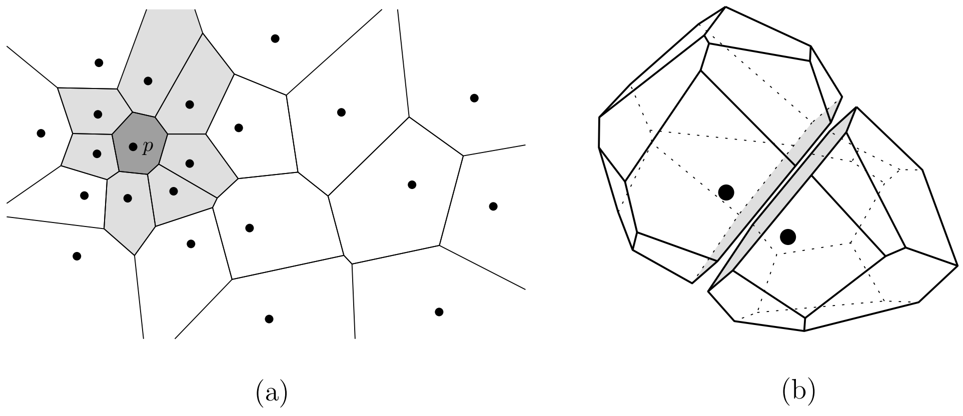


Figure 3.8: (a) VD of a set of points in the plane. The point p (whose Voronoi cell is dark grey) has seven neighbouring cells (light grey). **(b)** Two Voronoi cells adjacent to each other in $\mathbb{R}^3$ (they share the grey face).

The VD has many properties, and most of them are valid in any dimensions. Note that most of these properties are valid only when the set S of points is in *general position*, that is when for example in two dimensions no four points are cocircular, and no three points are collinear. Details concerning the possible degeneracies are given in Section 3.2.5. What follows is a list of the most relevant properties in the context of this thesis.

Size: if S has n points, then VD(S) has exactly n Voronoi cells since there is a one-to-one mapping between the points and the cells.

Voronoi vertices: in $\mathbb{R}^d$, a Voronoi vertex is equidistant from $(d + 1)$ points. Observe for instance in Figure 3.7(b) that the Voronoi vertices are at the centre of circles. In $\mathbb{R}^3$, a Voronoi vertex is at the centre of a sphere defined by 4 points in S.

Voronoi edges: in $\mathbb{R}^d$, a Voronoi edge is equidistant from d points.

Voronoi faces: in $\mathbb{R}^d$, a Voronoi face is equidistant from $(d - 1)$ points. Hence, in $\mathbb{R}^3$, it is the bisector plane perpendicular to the line segment joining two points.

Convex hull: let S be a set of points in $\mathbb{R}^d$, and p one of its points. $\mathcal{V}_p$ is unbounded if p bounds conv(S). Otherwise, $\mathcal{V}_p$ is the convex hull of its Voronoi vertices. Observe that in Figure 3.7(b), only the point in the middle has a bounded Voronoi cell.

Although the Voronoi diagram can be defined for other metrics than Euclidean (e.g. Manhattan, Laguerre, etc.), this thesis is restricted to the Voronoi diagram in the Euclidean space.

3.2.2 The Delaunay Triangulation

Let $\mathcal{D}$ be the VD of a set S of points in $\mathbb{R}^2$. Since VD(S) is a planar graph, it has a dual graph, and let $\mathcal{T}$ be this dual graph obtained by drawing straight edges between two points $p, q \in S$ if and only if $\mathcal{V}_p$ and $\mathcal{V}_q$ are adjacent in $\mathcal{D}$. Because the vertices in $\mathcal{D}$ are of degree 3, the graph $\mathcal{T}$ is a triangulation. $\mathcal{T}$ is actually called the Delaunay triangulation (DT) of S, and, as shown in Figure 3.9(a), partitions the plane into triangles—where the vertices of the triangles are the points in S generating each Voronoi cell—that satisfy the *empty circumcircle* test (a circle is

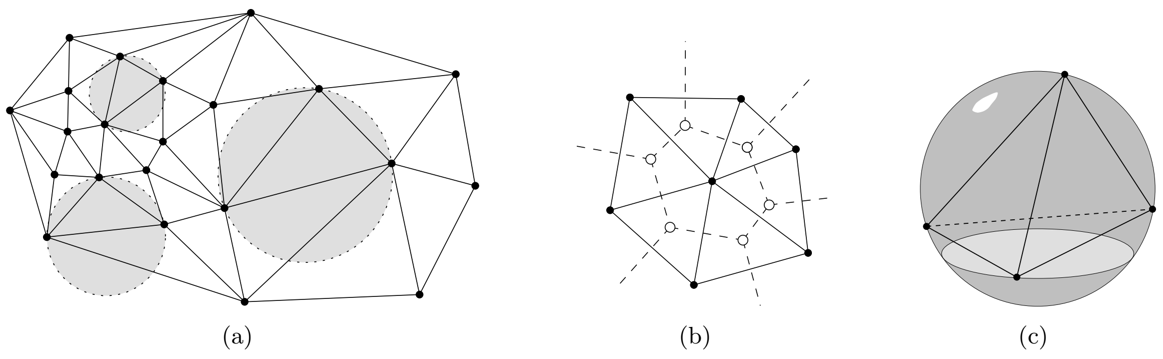


Figure 3.9: **(a)** The DT of a set of points in the plane (same point set as Figure 3.8(a)). **(b)** Both the DT (black lines) and the VD (dashed lines) of a set of points in the plane. **(c)** A Delaunay tetrahedron has an empty circumsphere.

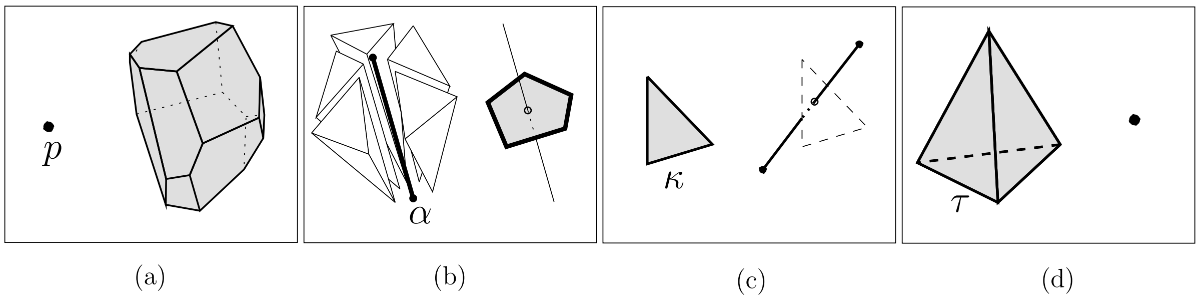


Figure 3.10: Duality in 3D between the elements of the VD and the DT.

said to be *empty* when no points are in its interior). If S is in general position, then DT(S) is unique.

The duality between the Voronoi diagram and the Delaunay triangulation is actually valid in any dimensions. The Delaunay triangulation of a set S of points in $\mathbb{R}^d$ is a simplicial complex where each d-simplex σ, formed by $d+1$ vertices in S, has an empty *circumball*. For $\mathbb{R}^3$, it is called the *Delaunay tetrahedralization*: the space is tessellation into non-overlapping tetrahedra having an empty *circumsphere*. Notice that in this thesis the acronym 'DT' can refer to the Delaunay triangulation in any dimensions, and that the dimensionality is usually implied in a section. The duality between the VD and the DT in $\mathbb{R}^3$ follows the rule given in Section 3.1.5: a Delaunay vertex p becomes a Voronoi cell (Figure 3.10(a)), a Delaunay edge α becomes a Voronoi face (Figure 3.10(b)), a Delaunay triangular face κ becomes a Voronoi edge (Figure 3.10(c)), and a Delaunay tetrahedron τ becomes a Voronoi vertex (Figure 3.10(d)). A Voronoi vertex is located at the centre of the sphere circumscribed to its dual tetrahedron, and two vertices in S have a Delaunay edge connecting them if and only if their two respective dual Voronoi cells are adjacent.

3.2.2.1 Convex Hull

In any dimensions, the DT of set S of points subdivides completely conv(S), i.e. the union of all the simplices in DT(S) is conv(S).

3.2.2.2 Local Optimality

Let $\mathcal{T}$ be a triangulation of S in $\mathbb{R}^d$. A facet σ (a $(d-1)$-simplex) is said to be *locally* Delaunay if it either

(i) belongs to only one d-simplex, and thus bounds conv(S), or

(ii) belongs to two d-simplices σ_a and σ_b, formed by the vertices of σ and respectively the vertices a and b, and b is outside of the circumball of σ_a.

The second case is illustrated in two dimensions in Figure 3.11(a). In an arbitrary triangulation, not every facet that is locally Delaunay is necessarily a facet of DT(S), but, as Delaunay (1934) himself proved, local optimality implies globally optimality in the case of the DT:

Lemma 3.1. *Let $\mathcal{T}$ be a triangulation of a point set S in $\mathbb{R}^d$. If every facet of $\mathcal{T}$ is locally Delaunay, then $\mathcal{T}$ is the Delaunay triangulation of S.*

This has serious implications as the DT—and its dual—are locally modifiable, i.e. we can theoretically insert, delete or move a points in S without recomputing DT(S) from scratch.

3.2.2.3 Angle Optimality

The DT in two dimensions has a very important property that is useful in applications such as finite element meshing or interpolation: the *max-min angle optimality*, firstly observed by Sibson (1978). Among all the possible triangulations of a set S of points in $\mathbb{R}^2$, DT(S) maximises the minimum angle (max-min property), and also minimises the maximum circumradii. In other words, it creates triangles that are as equilateral as possible. Notice here that maximising the minimum angle is not the same as minimising the maximum, and the DT only guarantees the former.

Finding 'good' tetrahedra, i.e. nicely shaped, is however more difficult than finding good triangles because the max-min property of Delaunay triangles does not generalise to three dimensions. A DT in $\mathbb{R}^3$ can indeed contain some tetrahedra, called *slivers*, whose four vertices are almost coplanar (see Figure 3.11(b)). Note that slivers do not have two-dimensional counterparts.

For many applications where the Delaunay tetrahedralization is used, e.g. in the finite element method in engineering or when the tetrahedra are used to perform interpolation directly, these tetrahedra are bad and must be removed. In the context of this thesis, one might wonder why use them if their properties are not good? First, it should be said that in most cases Delaunay tetrahedra have a more desirable shape than arbitrary tetrahedra. Rajan (1991) shows that the smallest sphere containing a Delaunay tetrahedron is smaller than the one of any other tetrahedron, i.e. the Delaunay criterion favours 'round' tetrahedra. Second, the VD is not affected by them—Voronoi cells in three dimensions will still be 'relatively spherical' even if the DT has many slivers. Third, many GIS operations on fields use the properties of the VD (see Chapter 5), and if only one tetrahedron does not have an empty circumsphere, then the VD is corrupted.

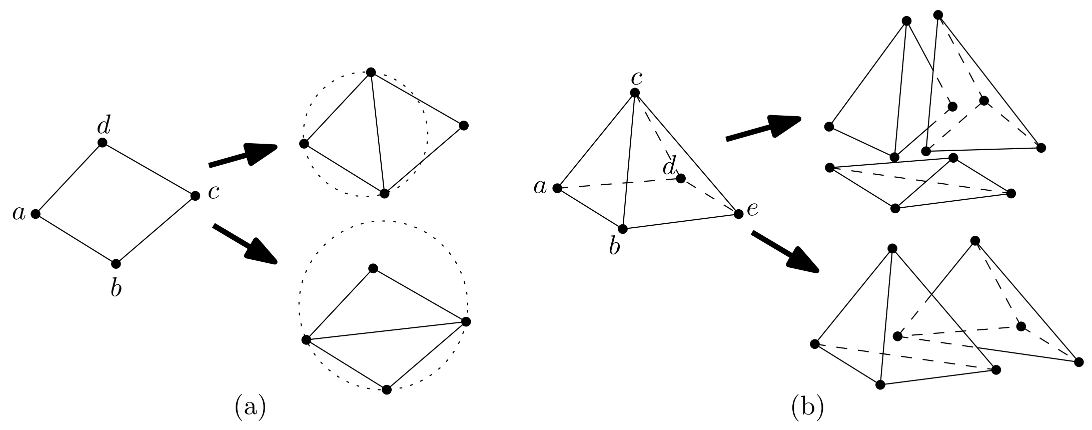


Figure 3.11: (a) A four-sided convex polygon $abcd$ can be triangulated in two different ways, but the empty circumcircle criterion guarantees that the triangles are as equilateral as possible. Notice that the edge ac is not locally Delaunay, but bd is. **(b)** In three dimensions, five vertices can be triangulated with either two or three tetrahedra. Although the tetrahedralization at the bottom has two nicely shaped tetrahedra, they are not Delaunay (the point d is inside the sphere $abce$, which also implies that b is inside the sphere $acde$). The tetrahedralization at the top respects the Delaunay criterion, but contains one very thin tetrahedron spanned by the points a, b, d and e.

3.2.3 Lifting on the Paraboloid

There exists a close relationship between DTs/VDs in $\mathbb{R}^d$ and convex polytopes in $\mathbb{R}^{d+1}$. This relationship was first observed by Brown (1979) (who used a spherical transformation), and I describe here the well-known *parabolic lifting map* (Seidel, 1982; Edelsbrunner and Seidel, 1986).

Let S be a set of points in $\mathbb{R}^d$, and let $x_1, x_2, \dots, x_d$ be the coordinates axes. The parabolic lifting map projects each vertex $v(v_{x1}, v_{x2}, \dots, v_{xd})$ to a vertex $v^+(v_{x1}, v_{x2}, \dots, v_{xd}, v_{x1}^2 + v_{x2}^2 + \dots + v_{xd}^2)$ on the paraboloid of revolution in $\mathbb{R}^{d+1}$. The set of points thus obtained is denoted S^+. Observe that, for the two-dimensional case, the paraboloid in three dimensions defines a surface whose vertical cross sections are parabolas, and whose horizontal cross sections are circles; the same ideas are valid in higher dimensions. The relationship is the following: every facet (a d-dimensional simplex) of the lower envelope of conv(S^+) projects to a d-simplex of the Delaunay triangulation of S. This is illustrated in Figure 3.12 for the construction of the DT in $\mathbb{R}^2$.

Since DTs are dual to VDs, the connection using the paraboloid is also valid for the VD in $\mathbb{R}^d$. For this case, the transformation maps each vertex $v(v_{x1}, v_{x2}, \dots, v_{xd})$ in S to a hyperplane that is tangent to the paraboloid at the point v^+. Consider P the polytope in $\mathbb{R}^{d+1}$ that is formed by the intersection of all the positive half-spaces defined by the hyperplanes. The VD(S) in $\mathbb{R}^d$ is the projection of the simplices of P.

In short, the construction of the d-dimensional DT can be transformed into the construction of the convex hull of the lifted set of points in $(d + 1)$ dimensions.

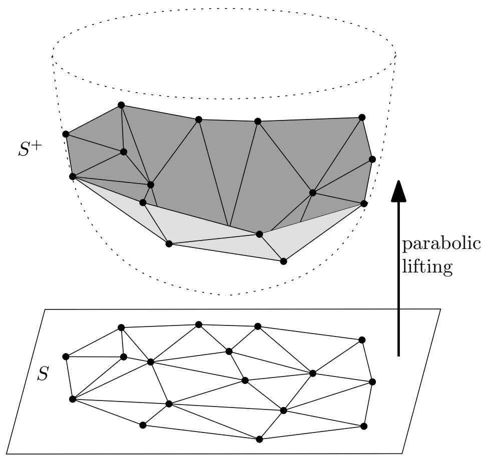


Figure 3.12: The parabolic lifting map for a set S of points in the plane.

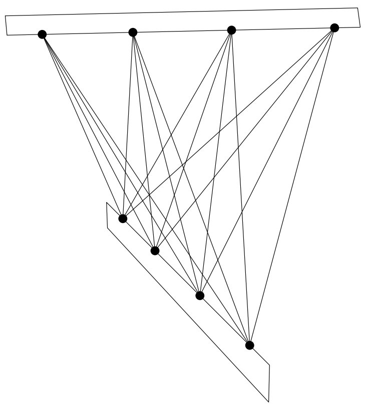


Figure 3.13: Example of a set of points for which the complexity of the DT is quadratic.

3.2.4 Space Complexity

The worst-case combinatorial complexity, i.e. the number of vertices, edges and faces, of VD(S) for a set S of n points in $\mathbb{R}^3$ is $\Theta(n^2)$ (Okabe et al., 2000). In other words, the number can be at most some constant times n^2. The same bound is valid for the DT in $\mathbb{R}^3$ because there exist some sets for which the number of tetrahedra in DT(S) is $\Theta(n^2)$; Figure 3.13 illustrates a simple example. Consider two skew lines, and distribute the points in S such that each lines has $n/2$ points. Observe that the sphere passing through any two contiguous points on the upper line and any two contiguous and the lower line is empty. Therefore, the number of tetrahedra for S will be around $n^2/4$.

Notice that these results are consistent with the worst-case combinatorial complexity of convex hulls in $\mathbb{R}^d$, since it is $\Theta(n^{\lfloor d/2 \rfloor})$ (de Berg et al., 2000). Once S is projected in $\mathbb{R}^4$ with the parabolic lifting map for instance, the complexity of conv(S^+) becomes $\Theta(n^2)$.

It should nevertheless be stated that these bounds are for extreme distributions of points, and that in practice the combinatorial complexity of the VD/DT in three dimensions is expected to be linear. Indeed, Dwyer (1991) shows that when the points in S are distributed uniformly in a sphere, the *expected* complexity is $\Theta(n)$. Attali and Boissonnat (2002) prove the same

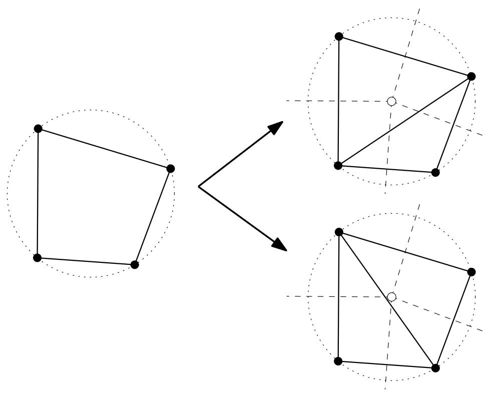


Figure 3.14: The DT for four cocircular points in two dimensions is not unique (but the VD is).

result when the points of S are distributed on a polyhedral surface. The datasets used for the experiments in Section 6.4 corroborate these theoretical results.

3.2.5 Degeneracies

The previous definitions of the VD and the DT assumed that the set S of points is in general position, i.e. the distribution of points does not create any ambiguity in the two structures. For the VD/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 two dimensions, when four or more points in S are cocircular there is an ambiguity in the definition of DT(S). As shown in Figure 3.14, the quadrilateral can be triangulated with two different diagonals, and an arbitrary choice must be made since both respect the Delaunay criterion (points should not be on the interior of a circumcircle, but more than three can lie directly on the circumcircle).

This implies that in the presence of four or more cocircular points, DT(S) is not unique. Notice that even in the presence of cocircular points, VD(S) is still unique, but it has different properties. For example, in Figure 3.14, the Voronoi vertex in the middle has degree 4 (remember that when S is in general position, every vertex in VD(S) has degree 3). When three or more points are collinear, DT(S) and VD(S) are unique, but problems with the computation of the structures can arise, as explained in the next chapter. All the previous observations generalise straightforwardly to higher dimensions.

3.3 Implementation of Geometric Algorithms

The authors of most papers in the computational geometry literature assume that they are working in a 'perfect world', i.e. they state that their algorithms are valid if and only if the input is in general position (which can have different meanings depending on the geometric problem at hand). When developing an algorithm and analysing its complexity, this assumption usually makes sense because one wants to avoid being overwhelmed with technical details (that are usually seen as implementation details), and also wants to simplify the complexity analysis (to be able to compare with other algorithms). Moreover, as de Berg et al. (2000, page 9) state, "degenerate cases can almost always be handled without increasing the asymptotic complexity of the algorithm".

The result is that the handling of degenerate cases is usually left to the programmer. However, modifying an algorithm to make it robust for any inputs can be in some cases an intricate, time-consuming and error-prone task because degeneracies have the effect of increasing considerably the number of special cases that must be handled. For non-geometric algorithms such as a sorting algorithm, it can be easy since only the case of two keys being equal must be handled, but geometric algorithms can have dozens, or even hundreds of special cases (Hoffmann, 1989).

In some cases, modifying the original algorithm so that it is robust against all input can be even more time-consuming and tedious than the design of the original solution in general position. A simple GIS-related example is that of Douglas (1974) who describes his early attempts to implement a routine to determine where two line segments in a plane intersect. What seemed like a simple problem at first, turned out to be an implementation nightmare because of the problems caused by vertical lines, tolerance, segments that are close to parallel, etc. (his routine finally had 36 different cases). He states: "All of these inconsistencies eventually drag the programmer down from his high level math (i.e. algebra), through computer language (i.e. FORTRAN), into the realm of the machine methods actually used to perform arithmetic operations and their restrictions". Admittedly, this problem can now be solved rather easily if vector representations are used instead of mathematical equations (Saalfeld, 1987), but I believe it gives a good example of the difficulties of dealing with special cases.

A popular solution among computational geometers for dealing with degeneracies is the use of *perturbation schemes*. Perturbing a set S of points means moving by an infinitesimal amount the points to ensure that S is in general position. Since there are no special cases to deal with, both the design and implementation are greatly simplified. However, as Seidel (1998) points out, although the method is certainly useful for a number of problems, it has been abused by many because it offers an easy solution to a difficult problem. The first problem is that after perturbing a problem, the algorithm will not solve the problem itself, but a closely related one. This can have serious drawbacks, for instance, in a DT, valid tetrahedra in the perturbed set of points can become degenerate (e.g. have a volume of zero) when the points are put back to their original positions.

The use of perturbations in geosciences is probably justified since there are always errors associated with each samples collected, and moving some a bit will not change anything for a given application. The problem is that geoscientific datasets can be riddled with degeneracies (e.g. with datasets collected by water-columns, many data points are collinear) and detecting and correcting them can be costly because the whole dataset must be visited. The second problem is that exact arithmetic is required for all the operations involved. I discuss further perturbation schemes in the next chapter, and I show that they can be very useful for solving some problems when manipulating a DT.

Another problem that arises during the implementation of geometric algorithms is that algorithms are usually designed to work with real numbers ($\mathbb{R}$), which are difficult to represent in a computer and are thus usually approximated with floating-point arithmetic. A floating-point number can not represent exactly every number and the roundoff error can create problems for geometric algorithms. For instance, if two line segments are nearly intersecting (but do not), a routine based on floating-point arithmetic could report that they do. For a single test like this one, the error might not be problematic. But objects like cell complexes have a combinatorial structure and geometric information is attached to each vertex. The combinatorial part depends on the results of several geometric tests: one wrong result and the whole combinatorial structure could become inconsistent. I discuss in Section 6.2 a few solutions for dealing with floating-point arithmetic, including the one that was chosen to implement the prototype GIS developed for this thesis.

In brief, to ensure that a geometric algorithm is robust against all input, i.e. it will recover from error conditions or special cases in the input, these special cases must first be identified, and then a means of dealing with the roundoff error of floating-point arithmetic must also be devised.

Chapter 4

Manipulating Three-dimensional Voronoi Diagrams

This chapter is about the manipulation of a Voronoi diagram in three dimensions, and of its dual. I discuss three main operations that are necessary to build a dynamic/kinetic spatial model, namely the insertion, the deletion and the movement of a point. The three operations are based on a single local operation—the bistellar flip—which permits us to modify locally a VD/DT. I also present the basic supporting operations needed, such as the traversal of the structures and the basic geometric tests.

As explained in the previous chapter, the concepts and definitions of the VD and the DT in two dimensions are rather simple, algorithms to manipulate them are well-known and several implementations exist. The concepts of the VD/DT generalise rather easily to three dimensions, but unfortunately the algorithms and their implementations are not straightforward. This is due to the many degenerate cases that arise in three dimensions.

In this chapter, I describe in detail, by taking into account the degenerate cases, the algorithms to manipulate a three-dimensional VD. Since the VD and the DT are dual structures, 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. Because it is easier, from an algorithmic and data structure point of view, to manage tetrahedra over arbitrary polyhedra (they have a constant number of vertices and adjacent cells), I construct and manipulate a VD by working only on its dual structure. When the VD is needed, it is extracted from the DT. This has the additional advantage of speeding up algorithms because when the VD is used directly intermediate Voronoi vertices—that will not necessarily exist in the final diagram—need to be computed and stored.

The algorithms described in this chapter are not necessarily asymptotically optimal, as conceptual simplicity and robustness were favoured over theoretical results and solutions that would be intricate to implement. Still, as demonstrated in Chapter 6, they take in practice a reasonable time for the problems that need to be solved, and they are also relatively easy to implement.

The first part of the chapter (Sections 4.1 and 4.2) introduces concepts that have all been previously published in different papers or books; only Section 4.2.4 is original work I made. The algorithm to construct a DT is also based on algorithms previously published, although I summarise and give many insights for the treatment of degenerate cases. Sections 4.4 and 4.5 are original contributions.

4.1 Extracting the Voronoi Diagram from the Delaunay Tetrahedralization

Let $\mathcal{T}$ be the DT of a set S of points in $\mathbb{R}^3$. The simplices of the dual $\mathcal{D}$ of $\mathcal{T}$ can be computed as follows (all the examples refer to Figure 3.10 on page 39):

Vertex: a single Voronoi vertex is easily extracted—it is located at the centre of the sphere passing through the four vertices of its dual tetrahedron τ.

Edge: a Voronoi edge, which is dual to a triangular face κ, is formed by the two Voronoi vertices dual to the two tetrahedra sharing κ.

Face: a Voronoi face, which is dual to a Delaunay edge α, is formed by all the vertices that are dual to the Delaunay tetrahedra incident to α. The idea is simply to 'turn' around a Delaunay edge and extract all the Voronoi vertices. These are guaranteed to be coplanar, and the face is guaranteed to be convex.

Polyhedron: the construction of one Voronoi cell $\mathcal{V}_p$ dual to a vertex p is similar: it is formed by all the Voronoi vertices dual to the tetrahedra incident to p. Since a Voronoi cell is convex by definition, it is possible to collect all the Voronoi vertices and then compute the convex hull (e.g. see Barber et al. (1996) for an algorithm); the retrieval of all the tetrahedra incident to p can be done by performing a BFS-like algorithm on the graph dual to the tetrahedra. A simpler method consists of first identifying all the edges incident to p, and then extracting the dual face of each edge.

Given $\mathcal{T}$, we must obviously visit all its 3-simplices to be able to extract $\mathcal{D}$. This means that computing $\mathcal{D}$ from $\mathcal{T}$ has a complexity of $\Theta(n)$ when S contains n points.

4.2 Basic Delaunay Tetrahedralization Operations

4.2.1 Initialisation: Big Tetrahedron

All of the algorithms related to the VD/DT in this thesis assume that the set S of points is entirely contained in a *big tetrahedron* (τ_{big}) several times larger than the range of S; conv(S) therefore becomes τ_{big}. Figure 4.1 illustrates a two-dimensional example. The construction of DT(S) is for example always initialised by first constructing τ_{big}, and then the points in S are inserted.

Doing this has many advantages, and is being used by several implementations (Facello, 1995; Mücke, 1998; Boissonnat et al., 2002; Liu and Snoeyink, 2005b). First, when a single point p needs to be inserted in DT(S), this guarantees that p is always inside an existing tetrahedron. We do not have to deal explicitly with vertices added outside the convex hull, as in Joe (1991). Second, we do not have to deal with the (nasty) case of deleting a vertex that bounds conv(S). Third, since a triangular face is always guaranteed to be shared by two tetrahedra, point location algorithms never 'fall off' the convex hull. Fourth, the Voronoi cells of the points that bounds conv(S) will be bounded, since the only unbounded cells will be the ones of the four points of τ_{big}. This can help for some of the spatial analysis operations described in Chapter 5.

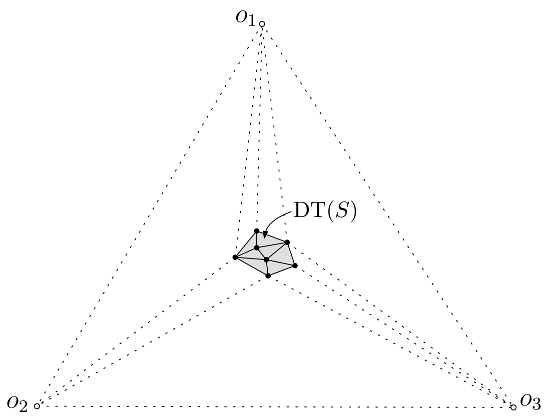


Figure 4.1: The set S of points is contained by a *big triangle* formed by the vertices o_1, o_2 and o_3. Many triangles outside conv(S) are created.

The main disadvantage is that more tetrahedra than needed are constructed. For example in Figure 4.1 only the shaded triangles would be part of DT(S). The extra triangles (tetrahedra in 3D) can nevertheless be easily marked as they are the only ones containing at least one of the four points forming τ_{big}.

4.2.2 Predicates

Constructing a DT and manipulating it essentially require two basic geometric tests (called *predicates*): Orient determines if a point p is over, under or lies on a plane defined by three points a, b and c; and InSphere determines if a point p is inside, outside or lies on a sphere defined by four points a, b, c and d. Both tests can be reduced to the computation of the determinant of a matrix (Guibas and Stolfi, 1985):

$$\textsc{Orient}(a,b,c,p) = \begin{vmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ p_x & p_y & p_z & 1 \end{vmatrix} \tag{4.1}$$

$$\textsc{InSphere}(a,b,c,d,p) = \begin{vmatrix} a_x & a_y & a_z & a_x^2+a_y^2+a_z^2 & 1 \\ b_x & b_y & b_z & b_x^2+b_y^2+b_z^2 & 1 \\ c_x & c_y & c_z & c_x^2+c_y^2+c_z^2 & 1 \\ d_x & d_y & d_z & d_x^2+d_y^2+d_z^2 & 1 \\ p_x & p_y & p_z & p_x^2+p_y^2+p_z^2 & 1 \end{vmatrix} \tag{4.2}$$

We can state that applying an identical translation to every point will not change the result of the determinant, and, for this reason, after translating all the points by $-p$ the two predicates are respectively implemented as the determinants of 3x3 and 4x4 matrices. I will not prove here the correctness of those predicates (see Guibas and Stolfi (1985) for the two-dimensional case) but simply state that Orient returns a positive value when the point p is above the plane defined by a, b and c; a negative value if p is under the plane; and exactly 0 if p is directly on the plane. Orient is consistent with the *left-hand rule*: when the ordering of a, b and c follows the direction of rotation of the curled fingers of the left hand, then the thumb points towards the positive side (the *above* side of the plane). In other words, if the three points defining a

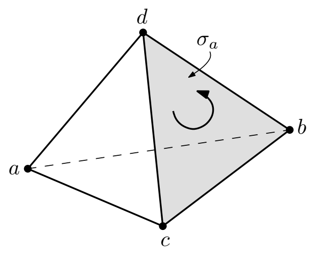


Figure 4.2: The tetrahedron *abcd* is correctly oriented since Orient (a, b, c, d) returns a positive result. The arrow indicates the correct orientation for the face σ_a, so that Orient (σ_a, a) returns a positive result.

plane are viewed clockwise from a viewpoint, then this viewpoint defines the positive side the plane. The predicate InSphere follows the same idea: a positive value is returned if p is inside the sphere; a negative if p is outside; and exactly 0 if p is directly on the sphere. Observe that to obtain these results, the points a, b, c and d in InSphere must be ordered such that Orient (a, b, c, d) returns a positive value.

It should be noticed that InSphere is derived from the parabolic lifting map (see Section 3.2.3). It is simply transformed into a four-dimensional Orient test: p is inside (outside) the sphere *abcd* if and only if p^+ lies under (above) the hyperplane $a^+b^+c^+d^+$, and directly on the sphere if p^+ lies on the hyperplane $a^+b^+c^+d^+$.

The 'orientation' of points in three dimensions is somewhat tricky because, unlike in two dimensions, we can not simply rely on the counter-clockwise orientation. In three dimensions, the orientation is always relative to another point of reference. When dealing with a single tetrahedron τ formed by the four vertices a, b, c and d (as in Figure 4.2), we say that τ is correctly oriented if Orient (a, b, c, d) returns a positive value. Notice that if two vertices are swapped in the order, then the result is the opposite (i.e. Orient (a, c, b, d) returns a negative value).

Vertices forming a face in a tetrahedron τ can also be ordered. As shown in Figure 4.2, a face σ_a, formed by the vertices b, c and d, is correctly oriented if Orient (σ_a, a) gives a positive result—in the case here, Orient (b, c, d, a) gives a negative result, therefore the correct orientation of σ_a is *cbd*. Observe that the face *bcd* is called σ_a because it is 'mapped' to the vertex a that is opposite; each of the four faces of a tetrahedron can be referred to in this way.

4.2.3 Three-dimensional Bistellar Flips

A bistellar flip is a local (topological) operation that modifies the configuration of some adjacent tetrahedra (Lawson, 1986; Edelsbrunner and Shah, 1996). Consider the set $S = \{a, b, c, d, e\}$ of points in general position in $\mathbb{R}^3$ and its convex hull conv(S). There exist two possible configurations, as shown in Figure 4.3:

1. the five points of S lie on the boundary of conv(S); see Figure 4.3(a). According to Lawson (1986), there are exactly two ways to tetrahedralize such a polyhedron: either with two or three tetrahedra. 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*.

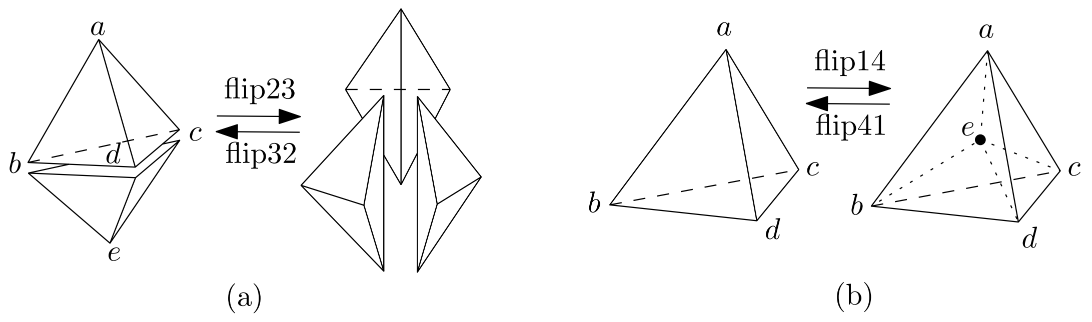


Figure 4.3: The three-dimensional bistellar flips.

2. one point e of S does not lie on the boundary of conv(S), thus conv(S) forms a tetrahedron; see Figure 4.3(b). The only way to tetrahedralize S is with four tetrahedra all incident to e.

Based on these two configurations, four types of flips in $\mathbb{R}^3$ can be described: *flip23*, *flip32*, *flip14* and *flip41* (the numbers refer to the number of tetrahedra before and after the flip). When S is in the first configuration, two types of flips are possible: a flip23 is the operation that transforms one tetrahedralization of two tetrahedra into another one with three tetrahedra; and a flip32 is the inverse operation. If S is tetrahedralized with two tetrahedra and the triangular face bcd is not locally Delaunay, then a flip23 will create three tetrahedra whose faces are locally Delaunay. A flip14 refers to the operation of inserting a vertex inside a tetrahedron, and splitting it into four tetrahedra; and a flip41 is the inverse operation that deletes a vertex.

Bistellar flips do not always apply to adjacent tetrahedra (Joe, 1989). For example, in Figure 4.3(a), 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 $bcde$.

Degenerate cases. To deal with degenerate cases, other flips need to be defined. Shewchuk (2003) defines and uses *degenerate flips* for the construction and the manipulation of constrained Delaunay triangulations in $\mathbb{R}^d$. A flip is said to be degenerate if it is a non-degenerate flip in a lower dimension. It is needed for special cases such as when a new point is inserted directly onto a simplex. For instance, Figure 4.4(a) shows the flip12 that splits a tetrahedron $abcd$ into two tetrahedra when a new point is inserted directly onto the edge ac, and the flip13 that splits $abcd$ into three tetrahedra when a new point is inserted directly onto the face abc. The flip12 is equivalent to a flip in one dimension, and the flip13 to the one needed to insert a new point in a triangle in two dimensions.

These flips are needed when dealing with constrained DT, but in our case they are not explicitly needed, as the operation they perform can be simulated, as shown in the next sections, with the flip23 and the flip32. The only degenerate flip needed for constructing a DT is the following. Consider the set $S = \{a, b, c, d, e, f\}$ of points configured as shown in Figure 4.4(b), with points b, c, d and e forming a plane. If S is tetrahedralized with four tetrahedra all incident to one edge—this configuration is called the *config44*—then a flip44 transforms one tetrahedralization into another one also having four tetrahedra. Note that the four tetrahedra are in config44 before and after the flip44. A flip44 is actually a combination in one step of a flip23 (that creates

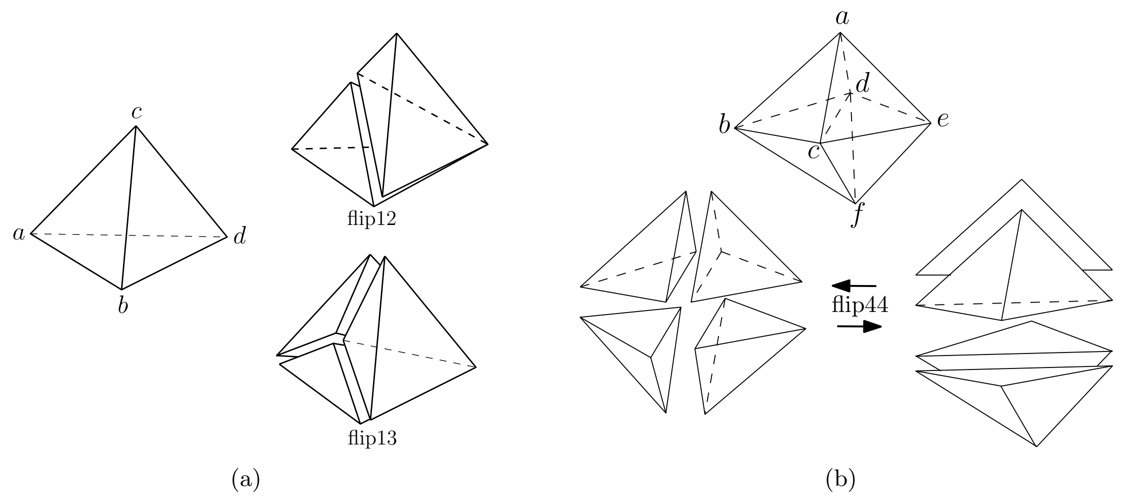


Figure 4.4: The degenerate flips **(a)** flip12 and flip13, and **(b)** flip44.

a flat tetrahedron *bcde*) followed immediately by a flip32 that deletes the flat tetrahedron; a flat tetrahedron is a tetrahedron spanned by four coplanar vertices (its volume is zero). The flip44 is conceptually equivalent to the well-known flip22 in two dimensions.

4.2.4 Traversal of a Subdivision

Given a cell complex $\mathcal{D}$ in $\mathbb{R}^d$, traversing it means to visit all its k-cells (where $0 \leq k \leq d$) in order to report them all (or only a few). This is a useful operation if one wants for instance to draw $\mathcal{D}$, or perform some operations that require all the cells (e.g. the extraction of isosurfaces, see Section 5.4.1). The traversal of $\mathcal{D}$ could be done rather easily with graph-search algorithms, but unfortunately, if one does not want to visit twice (or more) the same cell, these algorithms require to mark/flag which cells have already been visited. Mark bits have several disadvantages: (1) they require extra storage space; (2) it could be impossible to modify the data structure to store them; (3) but more importantly, resetting them after a complex has been traversed completely or partially (to perform another one later) is problematic.

There exists an old and simple algorithm to traverse a 2D DT (Gold et al., 1977; Gold and Maydell, 1978; Gold and Cormack, 1987). The method does not require the use of mark bits, and is based on the following observation.

Observation 4.1. *Any Delaunay triangulation in 2D may be ordered as a binary tree when considered from a fixed viewpoint.*

In that case, the elements of the tree are the triangles. Since a tree is built, it means that the parent (the predecessor during the traversal) of every triangle must be unique. Notice that the tree is binary because every triangle, after being entered, has two potential children. The method uses the geometry of the triangles—with respect to the viewpoint—to ensure that a triangle has only one parent; thus, only local information is necessary. As shown in Figure 4.5(a), the method starts at the triangle closest to the viewpoint and defines two types of triangle, based on the number of edges visible from the viewpoint. A triangle can only be entered from a *in* edge, and by applying a simple and consistent rule for each triangle having

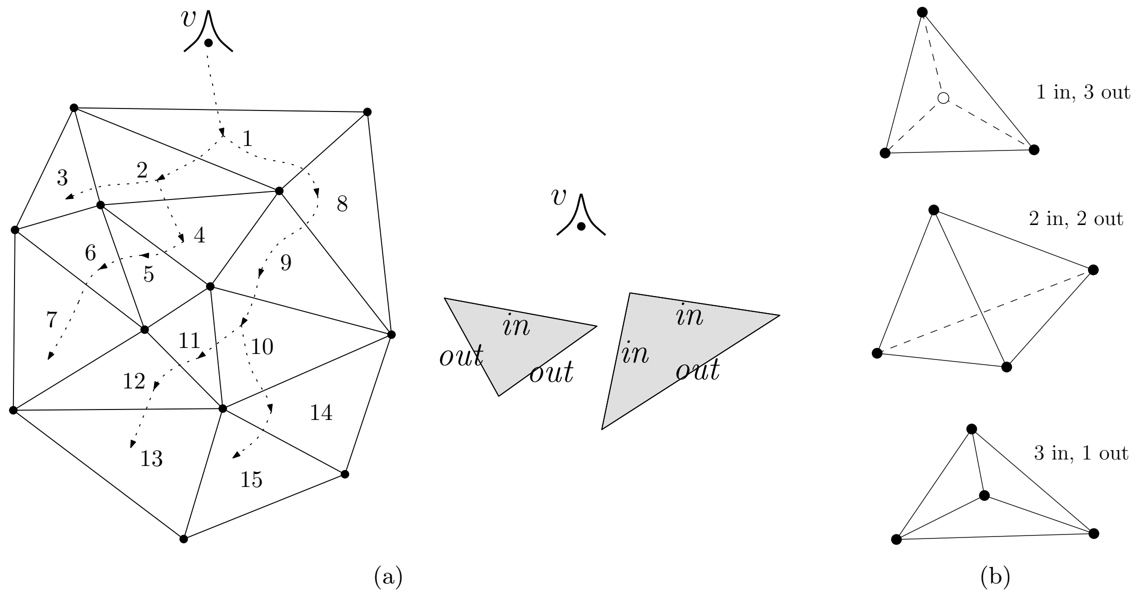


Figure 4.5: (a) The traversal of a triangulation (left), and the two types of triangles (right). The number refers to the order, viewed from the viewpoint v. Notice that a binary tree is built. **(b)** In three dimensions, three types of tetrahedra are possible, when viewed from a fixed viewpoint (the reader in this case).

two *in* edges, we can traverse a triangulation and guarantee that each triangle will be visited only once. Observe that the method reports the triangle in depth-first order, i.e. the ordering of the triangles is equivalent to a depth-first search on the graph dual to the triangulation.

Gold et al.'s algorithm has been extended for the traversal of any cell complexes in two dimensions, and for the traversal of convex cell complexes in three dimensions (de Berg et al., 1997). The idea is the same: define an order on the cells of the complex, and visit them in that order (a tree is also built, but it is obviously not binary since a cell can have more than two children). de Berg et al. (1997) show that if a unique predecessor to each cell can be defined, then a depth-first order can be obtained and each cell will be visited only once.

I present in the following a new algorithm, called TRAVERSE (see Algorithm 4.1), to traverse a Delaunay tetrahedralization. It uses ideas from Gold et al.'s and de Berg et al.'s algorithms, and is efficient and conceptually simple. As Figure 4.5(b) shows, three types of tetrahedra are possible when viewed from a viewpoint. Since we want to define a depth-first order on the tetrahedra (and build a tree), a tetrahedron τ must have one and only one predecessor; that means that only one face of τ can be its 'real' *in* face (this face is defined as the *entrance* face). When τ has two or three *in* faces, several rules could be used to define which one is the *entrance* face. The one I propose is as follows. (The notation for the faces is from Section 4.2.2, where a face σ_a is mapped to its opposite vertex a.) The *entrance* face can be the face mapped to the vertex in τ having the minimum distance to the viewpoint, or if a tetrahedron-based data structure is used (see Section 6.3), the vertex having the lowest lexicographic index in τ. Both of these rules are local, and can be computed in constant time. Notice that the function GETCHILD (line 6) in Algorithm 4.1 refers to a function that returns the children of a given tetrahedron, after it has been entered by one face.

Algorithm 4.1: TRAVERSE($\mathcal{T}, \tau_1, v$)

```
Input: A tetrahedralization $\mathcal{T}$, a starting tetrahedron $\tau_1$, and a viewpoint $v$
Output: $L$: a list containing all the tetrahedra in $\mathcal{T}$
1  push $\tau_1$ on stack
2  while stack is non-empty do
3      $\tau \leftarrow$ pop from stack
4      Add $\tau$ to $L$
5      for $i \leftarrow 0$ to $2$ do
6          $\tau_c \leftarrow$ GETCHILD($i$)
7          $\sigma \leftarrow \tau \cap \tau_c$
8          if $\sigma$ is entrance face of $\tau_c$ then
9              push $\tau_c$ on stack
10         end
11     end
12 end
```

Only one special case must be handled here: when the three vertices of a face *abc* are coplanar with the viewpoint v. To make sure that the face *abc*, shared by the tetrahedra τ_1 and τ_2, will be *in* for τ_1 and *out* for τ_2 (or vice versa), it suffices to define a constant rule. A simple solution is to define two vectors, $\vec{v_1}$ and $\vec{v_2}$, that are perpendicular to the face *abc*, and that are respectively pointing to the inside of τ_1 and τ_2, and test the angle between these and a fixed vector defined arbitrarily. The tetrahedron for which the angle between the vectors is positive is the *in* face.

4.2.5 Point Location

Given the Delaunay tetrahedralization $\mathcal{T}$ of a set S of n points and a query point p, the point location problem consists of determining inside which tetrahedron of $\mathcal{T}$ lies p. This is a necessary operation for the incremental constructing of a DT, for deleting a vertex from it (depending on the data structure used), and also for different spatial analysis operations.

Many algorithms solve the generic point location problem in two dimensions, i.e. for a cell complex, in the optimal time of $\mathcal{O}(\log n)$ (Snoeyink, 1997). For the special case of the DT, different optimal methods are linked to the construction of the DT: Guibas et al. (1992) describe one that comes with their incremental algorithm, and Edelsbrunner and Shah (1996) have generalised the method to higher dimensions. Devillers (2002b) also proposes a hierarchical structure to compute and store the DT in any dimensions where the point location is sped up because it is made at different levels of the hierarchy. The major problems with these algorithms are that they use additional storage (e.g. Edelsbrunner and Shah use a graph that needs $\mathcal{O}(n^2)$ additional storage to store the history of previously performed flips), they require preprocessing for creating the additional data structure, and they are also often very complicated to implement. As Mücke et al. (1999) note, optimal algorithms do not necessarily mean better results in practice because of the amount of preprocessing involved, the extra storage needed, and also because the optimal algorithms do not always consider the dynamic case, where points in the DT could be deleted. For many problems like this one, practitioners have been known to favour sub-optimal algorithms that are easier to implement.

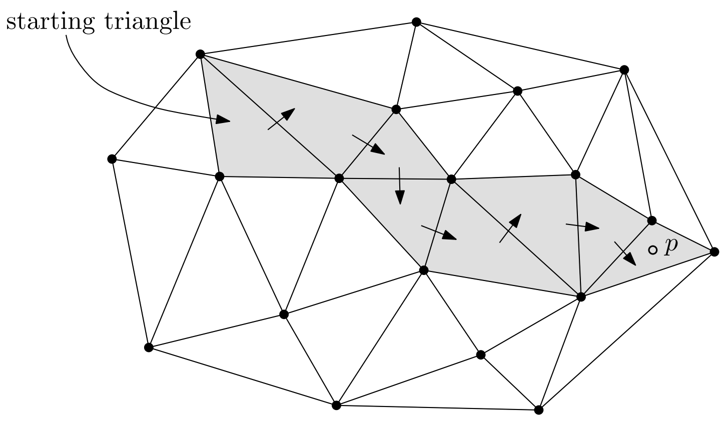


Figure 4.6: The Walk algorithm for a DT in two dimensions. The query point is p.

Mücke et al. (1999) and Devillers et al. (2002) analyse theoretically sub-optimal algorithms, for the DT in three dimensions, that yield fast practical performances. These algorithms are desirable in the case of a dynamic structure because they do not use any additional storage or preprocessing. The adjacency relationships between the simplices in a DT are used to perform the point location. They discuss and compare mainly two strategies: the *walking* and the *marching*. The first one was described in the earliest papers about the construction of the DT in two dimensions (Gold et al., 1977; Green and Sibson, 1978): in a DT in $\mathbb{R}^d$, starting from a d-simplex σ, we move to one of the neighbours of σ (σ has $(d+1)$-neighbours; we choose one neighbour such that the query point p and σ are on each side of the $(d-1)$-simplex shared by σ and its neighbour) until there is no such neighbour, then the simplex containing p is σ. The algorithm is illustrated in Figure 4.6 for the two-dimensional case, and Algorithm 4.2 gives the pseudo-code for the three-dimensional case (notice that at line 4, the face σ_i must be correctly oriented, as explained in Section 4.2.2).

The walking method has often been overlooked by theoreticians because not much can be said about the number of simplices that will be visited. As Figure 4.6 shows, the walk seems to go in the general direction of the final destination, but other routes could be followed if the order in which the edges are tested within a triangle was different.

An important consideration here is that the algorithm Walk as described in Algorithm 4.2 is valid only with a DT. Indeed, for some arbitrary triangulations, such as the one in Figure 4.7(a), it might never end. The walk is guaranteed to end is the case of a DT because it was proved that DTs, in any dimensions, are acyclic for any fixed viewpoint (Edelsbrunner, 1990). In other words, the in-front/behind relationship for the d-simplices of a DT, with respect to a viewpoint, is acyclic (see Figure 4.7(b)). Walk can nevertheless be modified easily for arbitrary tetrahedralizations: when entering a tetrahedron, if its faces are tested in a random order instead of always the same, then the walk will find its way to the query point (although it might take a long time!). Also, notice that Walk is not affected by degenerate cases, and that if an Orient test returns 0, then it is simply considered a positive result. This will ensure that if the query point p is located exactly that the same position as one point in S, then one tetrahedron incident to p will be returned.

The second point location strategy, marching, is also simple: only the d-simplices intersected by a line segment joining one vertex of the starting tetrahedron to p are visited. Although the

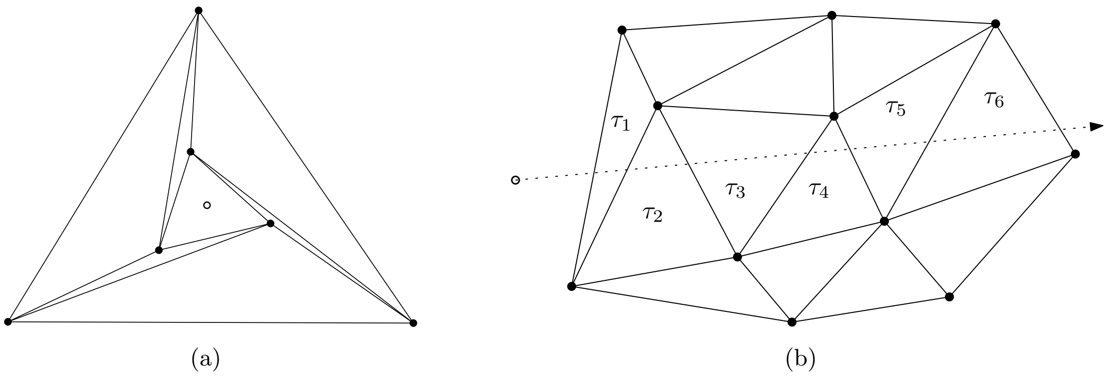


Figure 4.7: **(a)** The Walk algorithm might never end for this triangulation because the three skinny triangles form a cycle, when viewed from the viewpoint in the middle. **(b)** The triangles in a DT can be ordered in an in-front/behind manner when viewed from a viewpoint.

Algorithm 4.2: Walk($\mathcal{T}$, τ, p)

```
   Input: A Delaunay tetrahedralization $\mathcal{T}$, a starting tetrahedron $\tau$, and a query point $p$
   Output: $\tau_r$: the tetrahedron in $\mathcal{T}$ containing $p$
1  while $\tau_r$ not found do
2      for $i \leftarrow 0$ to 3 do
3          $\sigma_i \leftarrow$ get face opposite vertex $i$ in $\tau$
4          if Orient $(\sigma_i, p) < 0$ then
5              $\tau \leftarrow$ get neighbouring tetrahedron of $\tau$ incident to $\sigma_i$
6              break
7          end
8      end
9      if $i = 3$ then
           // all the faces of $\tau$ have been tested
10         Return ($\tau_r = \tau$)
11     end
12 end
```

number of simplices visited will obviously be less than with Walk, the intersection tests in three dimensions are much more computationally expensive than Orient tests.

Based on experimental results, both Devillers et al. (2002) and Mücke et al. (1999) conclude that the walking strategy is the fastest solution. Different improvements to this simple algorithm can be done to speed up the implementation. Firstly, remembering by which face we 'entered' a tetrahedron, in order not to test it again, can improve slightly the algorithm. Secondly, instead of starting the walk each time from a random tetrahedron, we can cleverly select one. One basic method involves always remembering the last tetrahedron visited and start from there for the next walk. One could argue that in practice the ordering of the points are rarely totally random and often the next point to be inserted in a mesh is close to the last inserted, and that when Walk is used for analysing the DT (e.g. interpolation) then we are most likely to need tetrahedra that are close to each other. Another way, similar to a 'bucketing' method, is to randomly select a certain number of tetrahedra in the DT, and to start each walk from the closest one (selected by a simple Euclidian distance test). Mücke et al. (1999) show that this method—they give empirical results to choose an appropriate number of tetrahedra with respect to the number n of points—has an expected running time close to $\mathcal{O}(n^{1/4})$.

4.3 Constructing a Delaunay Tetrahedralization

Mainly three paradigms of computational geometry can be used for computing a Delaunay triangulation in two and three dimensions: divide-and-conquer, sweep plane, and incremental insertion. Each one of these paradigms yields an optimal algorithm in two dimensions, i.e. a DT of n points is computed in $\Theta(n \log n)$. Examples of these are the divide-and-conquer and the incremental insertion algorithms of Guibas and Stolfi (1985), and the sweep line algorithm of Fortune (1987) that constructs directly the VD.

In three dimensions, things are a bit more complicated. Cignoni et al. (1998) developed an algorithm, called `DeWall` and based on the divide-and-conquer paradigm, for constructing the DT in any dimensions. Although the worst-time complexity of this algorithm is $\mathcal{O}(n^3)$ in three dimensions, they affirm that the speed of their implementation is comparable to the implementation of known incremental algorithms, and is subquadratic. Shewchuk (2000) proposes sweep algorithms for the construction of the *constrained* Delaunay triangulation in $\mathbb{R}^d$, and these can be used for the normal DT. As is the case with Cignoni et al. (1998), his algorithm is sub-optimal for the three-dimensional case.

In $\mathbb{R}^3$, only incremental insertion algorithms have a complexity that is worst-case optimal, i.e. $\Theta(n^2)$ since the complexity of the DT in $\mathbb{R}^3$ is quadratic. With these algorithms, each point is inserted one at a time in a valid DT and the tetrahedralization is 'updated', with respect to the Delaunay criterion, between each insertion. Observe that the insertion of a single point p in a DT only modifies locally the DT, i.e. only the tetrahedra whose circumsphere contains p need to be deleted and replaced by new one respecting the Delaunay criterion (see Figure 4.8(a) for a two-dimensional example). In sharp contrast to this, divide-and-conquer and plane sweep algorithms build a DT in *one* operation (this is a batch operation), and if another point needs to be inserted after this, the whole construction operation must be done again from scratch.

Incremental insertion algorithms are therefore mandatory for building a dynamic and kinetic spatial model. Consider $\mathcal{T}$, the DT(S) of a set S of n points in $\mathbb{R}^3$. The insertion of a single point p, thus getting $\mathcal{T}^p = \mathcal{T} \cup \{p\}$, can be done with two incremental insertion algorithms: the *Bowyer-Watson* algorithm, or one based on flipping in a tetrahedralization.

4.3.1 The Bowyer-Watson Algorithm

Bowyer (1981) and Watson (1981) both published independently a paper—in the same issue of the same journal—describing essentially the same algorithm, which is valid in any dimensions. The idea is relatively simple: all the d-simplices *conflicting* with p, i.e. whose circumball contains p, are deleted from $\mathcal{T}$, and the 'hole' thus created is filled by creating edges joining p to each vertex of the hole (see Figure 4.8(a)). The following lemma proves that these new edges are guaranteed to be locally Delaunay (this is for the two-dimensional case, but it generalises to three dimensions easily):

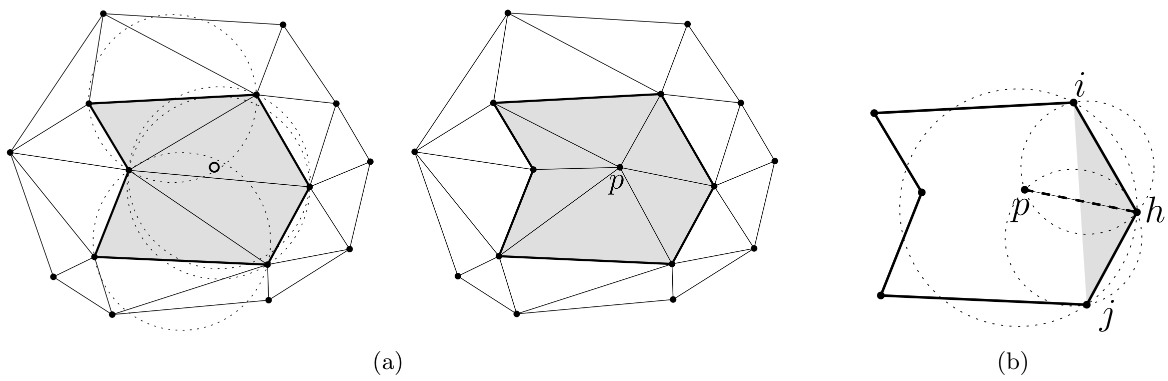


Figure 4.8: (a) The DT before and after a point p has been inserted. Notice that the DT is updated only locally (only the shaded part of the triangulation is affected). **(b)** After deleting the triangles whose circumcircle contain p, the edge ph is guaranteed to be Delaunay.

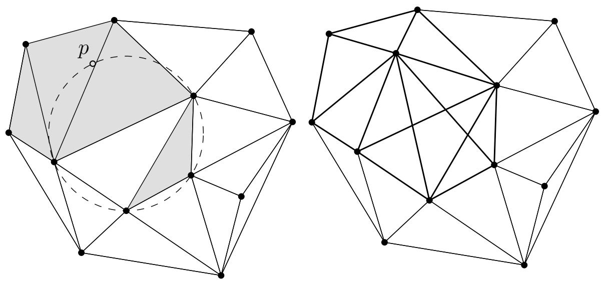


Figure 4.9: The insertion of p with the Bowyer-Watson algorithm causes the resulting triangulation to be invalid. In the left triangulation, the triangles whose circumcircle contain p are shaded. Notice that two triangles have exactly the same circumcircle, but one, because of a roundoff error, does not contain p.

Lemma 4.1. *Let p be a newly inserted point in $\mathcal{T}$ and h one of the vertices of a triangle hij deleted because its circumcircle contains p. The edge ph is locally Delaunay.*

Proof. There are only two ways of triangulating a convex quadrilateral, and only one respects the Delaunay criterion (see Figure 3.11). As shown in Figure 4.8(b), ij is not locally Delaunay since p is inside the circumcircle of hij. Therefore, ph is the only other possibility and is locally Delaunay. □

In its naive form, the Bowyer-Watson algorithm is prone to errors due to floating-point arithmetic and in some cases, as Field (1986) and Sugihara and Hiroshi (1995) demonstrate, the result of the insertion of a single point can be disastrous. Figure 4.9 illustrates one unfortunate example in two dimensions: a point p inserted on the circumcircle of two triangles yields something that is not even a triangulation. Sugihara and Hiroshi (1995) explain how to solve this problem in two and three dimensions: before retetrahedralizing the hole, they basically ensure that it is formed by connected simplices.

A further important consideration is that creating a hole in a tetrahedralization usually implies the use of an auxiliary data structure to keep track of the hole's boundary, and is intricate from

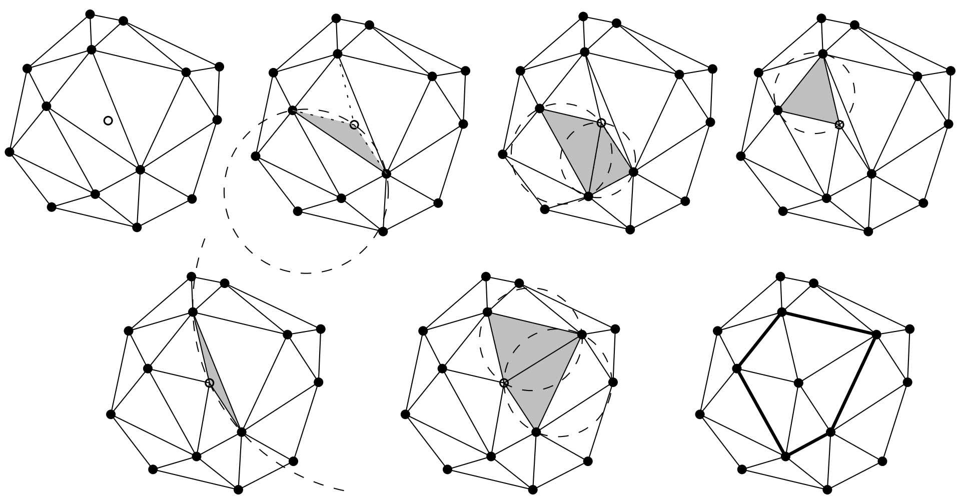


Figure 4.10: Step-by-step insertion, with flips, of a single point in a DT in two dimensions.

an algorithmic and data structure point of view as the geometric and topological relationships of the tetrahedralization are temporarily destroyed.

4.3.2 An Incremental Flip-based Algorithm

The alternative to creating a hole is to use bistellar flips (see Section 4.2.3) to modify the configuration of tetrahedra in the vicinity of p. It should first be noticed that flips are operations valid in any dimensions, and not only in two or three dimensions (Lawson, 1986). They permit us to keep a complete tetrahedralization during the insertion process, and hence algorithms are relatively simple to implement and also numerically more robust. Although a flip-based algorithm requires somewhat more work than an algorithm where a hole is created—Liu and Snoeyink (2005b) state that on average 1.5–2 times more tetrahedra are created and that flippability tests also slow down the whole process—its implementation is simplified and less error-prone since the maintenance of adjacency relationships in a DT is encapsulated in the flip operations.

Two dimensions. The first flip-based algorithm was designed to construct the DT in two dimensions. It was developed by Lawson (1972, 1977) who proved that, starting from an arbitrary triangulation of a set S of points in the plane, flipping edges (i.e. replacing the diagonal of a quadrilateral as in Figure 3.11(a)) can transform this triangulation into any other triangulation of S, including the Delaunay triangulation. Roundoff errors can still happen in this case but it is unlikely that they will cause the disastrous consequence shown in Figure 4.9; the most likely problem in this case is that the final triangulation will not respect the Delaunay criterion. The total running time of this algorithm is $\mathcal{O}(n^2)$, since there is $\mathcal{O}(n)$ triangles that must be tested against each other.

An incremental insertion algorithm based on edge flipping can improve this to $\mathcal{O}(n \log n)$ (Guibas and Stolfi, 1985). What follows are the main steps for the insertion of a single point p in a DT, and the process is shown in Figure 4.10. First, the triangle τ containing p is identified and then split into three new triangles by joining p to every vertex of τ. Second, each new triangle is tested—according to the Delaunay criterion—against its opposite neighbour (with respect

Algorithm 4.3: InsertOnePoint ($\mathcal{T}$, p)

```
Input: A DT $\mathcal{T}$ of a point set $S$ in $\mathbb{R}^3$, and a new point $p$ to insert
Output: $\mathcal{T}^p = \mathcal{T} \cup \{p\}$
1  $\tau \leftarrow$ Walk(DT($S$), $p$)
2  insert $p$ in $\tau$ with a flip14
3  push 4 new tetrahedra on stack
4  while stack is non-empty do
5      $\tau = \{p, a, b, c\} \leftarrow$ pop from stack
6      $\tau_a = \{a, b, c, d\} \leftarrow$ get adjacent tetrahedron of $\tau$ having $abc$ has a facet
7      if $d$ is inside circumsphere of $\tau$ then
8          Flip($\tau$, $\tau_a$)
9      end
10 end
```

to p); if it is not a Delaunay triangle then the edge shared by the two triangles is flipped and the two new triangles will also have to be tested later. This process stops when every triangle having p as one of its vertices respects the Delaunay criterion.

Three dimensions. While the concept of flipping generalises to three dimensions, Lawson's algorithm (Lawson, 1972, 1977) does not, as Joe (1989) proves. Indeed, starting from an arbitrary tetrahedralization, it is possible that during the process of flipping, a non-locally Delaunay facet be impossible to flip (this case happens when the union of two tetrahedra is concave). Joe nevertheless later circumvented this problem by proving the following result (Joe, 1991):

Lemma 4.2. *Given a DT(S) and a point p in $\mathbb{R}^3$, there always exists at least one sequence of bistellar flips to construct DT($S \cup \{p\}$).*

In this case, there will be non-Delaunay facets impossible to flip, but he proved that there will always be a flip possible somewhere else such that the algorithm progresses. This can form the basis of an incremental insertion algorithm for the construction of the Delaunay tetrahedralization, that is a straightforward generalisation of the two-dimensional case. Let S be a set of points in $\mathbb{R}^3$ and let $\mathcal{T}$ be DT(S), Algorithm 4.3 shows the algorithm, called InsertOnePoint, needed to restore the 'Delaunayness' in $\mathcal{T}$ when a single point p is inserted. The algorithm is adapted from Joe (1991), and is conceptually the same as Edelsbrunner and Shah (1996). As is the case with the two-dimensional algorithm, the point p is first inserted in $\mathcal{T}$ with a flip (flip14 in the case here), and the new tetrahedra created must be tested to make sure they are Delaunay. The sequence of flips needed is controlled by a stack containing all the tetrahedra that have not been tested yet. The stack starts with the four resulting tetrahedra of the flip14, and each time a flip is performed, the new tetrahedra created are added to the stack. The algorithms stops when all the tetrahedra incident to p are Delaunay, which also means that the stack is empty.

The Flip method in InsertOnePoint needs to be refined because, unlike in two dimensions, different geometric cases yield different flips. Assume that p was inserted in $\mathcal{T}$, and that a certain number of flips have been performed but that $\mathcal{T}^p = \mathcal{T} \cup \{p\}$ is not Delaunay yet. All the tetrahedra incident to p, which form star(p) (see Section 3.1.2.1), must be tested to ensure

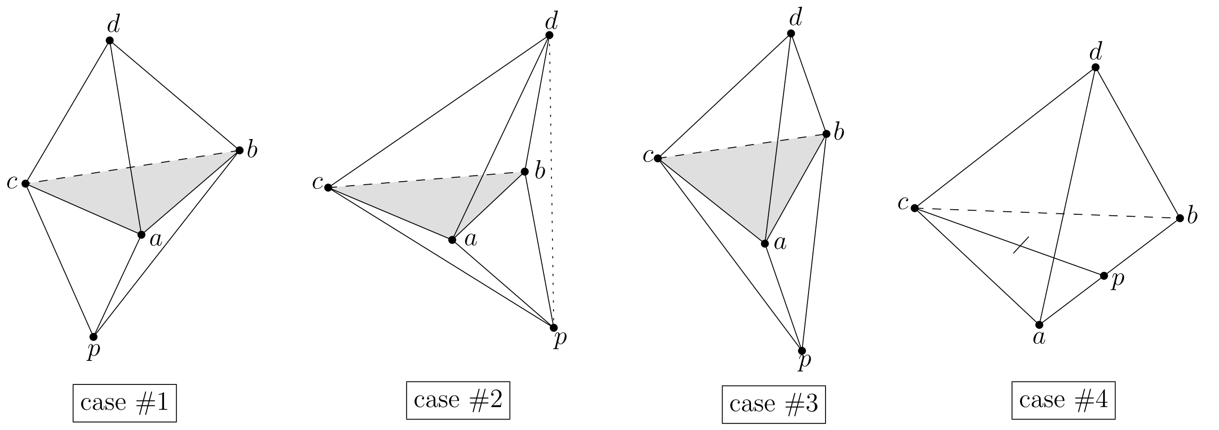


Figure 4.11: Four cases are possible for the FLIP algorithm. For case #3, vertices p, a, b and d are coplanar. For case #4, the tetrahedron $abcp$ is flat.

that they are Delaunay, and notice that the ones that have not been tested yet are in the stack. Let $\tau = \{p, a, b, c\}$ be the next tetrahedron popped from the stack. To ensure τ is Delaunay, we need only to test it with the tetrahedron $\tau_a = \{a, b, c, d\}$ outside star(p) and incident to the facet abc; we are actually testing if abc is locally Delaunay. If the circumsphere of τ contains d, different options for the flip required are possible, according to the geometry of τ and τ_a. Observe that if we look from p, we can see one, two or even three facets of τ_a (same idea as in Figure 4.5(b)). When S is in general position, two cases are possible (both cases refer to Figure 4.11):

Case #1: only one face of τ_a is visible, and therefore the union of τ and τ_a is a convex polyhedron. In this case, a flip23 is performed.

Case #2: two faces of τ_a are visible, and therefore the union of τ and τ_a is non-convex. If there exists a tetrahedron $\tau_b = abpd$ in $\mathcal{T}^p$ such that the edge ab is shared by τ, τ_a and τ_b, then a flip32 can be performed. If there exists no such tetrahedron, then no flip is performed. The non-locally Delaunay facet abc will be 'rectified' by another flip performed on adjacent tetrahedra, as Joe (1991) proves.

When three faces of τ_a are visible, no action is taken and the next tetrahedron in the stack is processed.

The implementation of the test that determines which flip should be applied for τ and τ_a does not test if their union is convex or concave. What is used instead is a test that determines if the edge joining the two apexes of τ and τ_a crosses the interior of their common facet. If the edge does, it means the union is convex, and if it does not the union is concave.

Time complexity. When an incremental insertion algorithm based on INSERTONEPOINT is used to compute DT(S) of a set S of n points in $\mathbb{R}^3$, the algorithm takes $\Theta(n^2)$, which is worst-case optimal (Edelsbrunner and Shah, 1996). However, in practice, the algorithm will most likely be faster. Edelsbrunner and Shah (1996) prove that if the n points are uniformly distributed in a unit cube, then the expected running time goes down to $\mathcal{O}(n \log n)$, provided that the history of the flips is used for point location.

```
Algorithm 4.4: FLIP($\tau$, $\tau_a$)
Input: Two adjacent tetrahedra $\tau$ and $\tau_a$
Output: A flip is performed, or not
// the 4 cases refer to Figure 4.11
1  if case #1 then
2      flip23($\tau$, $\tau_a$)
3      push tetrahedra $pabd$, $pbcd$ and $pacd$ on stack
4  else if case #2 and $\mathcal{T}^p$ has tetrahedron $pdab$ then
5      flip32($\tau$, $\tau_a$, $pdab$)
6      push $pacd$ and $pbcd$ on stack
7  else if case #3 and $\tau$ and $\tau_a$ are in config44 with $\tau_b$ and $\tau_c$ then
8      flip44($\tau$, $\tau_a$, $\tau_b$, $\tau_c$)
9      push on stack the 4 tetrahedra created
10 else if case #4 then
11     flip23($\tau$, $\tau_a$)
12     push tetrahedra $pabd$, $pbcd$ and $pacd$ on stack
13 end
```

Furthermore, the work needed to insert a single point p in $\mathcal{T}$ is proportional to s, the number of tetrahedra conflicting with p (Edelsbrunner and Shah, 1996). Indeed, notice that every flip in INSERTONEPOINT is actually deleting one and only one non-Delaunay tetrahedron, and replacing it with some new ones incident to p. The first flip14 removes only one tetrahedron and replaces it by four new ones; a flip23 deletes a conflicting tetrahedron τ_a and replaces τ (the tetrahedron incident to p in star(p)) and τ_a by three tetrahedra all incident to p; and a flip32 also removes one conflicting tetrahedron (the one outside star(p)) and creates two new tetrahedra incident to p. Also, once a tetrahedra is deleted after a flip, it is never re-introduced in $\mathcal{T}^p$. Therefore, if s tetrahedra in $\mathcal{T}$ are conflicting with p, then exactly s flips are needed to obtain $\mathcal{T}^p$.

Degenerate cases. Joe (1991) not only proved that a flip-based incremental insertion works in three dimensions, he proved it even when S has degeneracies. He detailed the different configurations possible when a flip is to be performed. I present in the following the many degeneracies that can arise and I describe how to solve them. These solutions are mainly taken from Joe (1991), Shewchuk (1997b) and Shewchuk (2003).

When S contains degenerate cases, the intersection tests between the two apexes of τ and τ_a can return other results (both cases refer to Figure 4.11):

Case #3: the line segment pd intersects directly one edge of abc (assume it is ab here). Thus, vertices p, d, a and b are coplanar. Observe that a flip23, that would create a flat tetrahedron $abdp$, is possible if and only if τ and τ_a are in config44 (see Figure 4.4(b)) with two other tetrahedra τ_b and τ_c, such that the edge ab is incident to τ, τ_a τ_b and τ_c. Since the four tetrahedra are in config44, a flip44 can be performed. If not, no flip is performed.

Case #4: the vertex p is directly on one edge of the face abc (assume it is ab here). Thus, p lies in the same plane as abc, but also as abd. The tetrahedron $abcp$ is obviously flat. The only reason for such a case is because p was inserted directly on one edge of $\mathcal{T}$ in the first place (the edge ab), and the first flip14 to split the tetrahedron containing p created two flat tetrahedra. Because p was inserted on the edge ab, all the tetrahedra in $\mathcal{T}$ incident to ab must be split into two tetrahedra. This could be done with the degenerate flip12, but it can also be done simply with a sequence of flip23 and flip32. When the case #4 happens, it suffices to perform a flip23 on τ and τ_a. Obviously, another flat tetrahedron $abdp$ will be created, but this one will deleted by another flip.

Observe that although flat tetrahedra are not allowed in a DT (the circumsphere of a flat tetrahedron is undefined and contains all the points in a set), they are allowed during the process of updating a tetrahedralization, as long as the combinatorial structure stays coherent.

The only degenerate cases remaining to handle are: (1) there exists a vertex at the exact location where p was supposed to be inserted; (2) p lies directly on the circumsphere of a tetrahedron in $\mathcal{T}$; (3) p was inserted directly on a face of a tetrahedron in $\mathcal{T}$. The first case can be handled trivially: simply test the distance from p to each vertex of the tetrahedron returned by WALK against a tolerance. If one distance is smaller than the tolerance, do not insert p at all. The second case is also easy to handle: do not perform a flip since this is an unnecessary operation that will slow down the algorithm. Since p is directly on the circumsphere and not inside, no operation will still result in a correct DT. The last case does not require any special treatment: the flip14 that inserts p in a tetrahedron will create one flat tetrahedron, and this tetrahedron will be deleted when tested against the tetrahedron outside star(p) sharing the same face.

4.4 Deleting a Vertex in a Delaunay Tetrahedralization[1]

As we saw in the precedent section, the construction of the DT of a set S of points in $\mathbb{R}^3$ is a rather well-known problem and many efficient algorithms exist (at least when S is in general position). The 'inverse' problem—the deletion of a vertex v in a DT(S), thus obtaining DT($S \setminus \{v\}$)—is however much less documented and is still a problem in practice. Most of the work on this topic has been done for the two-dimensional case and very little can be found for the three- and higher-dimensional cases. The problem has been tackled mostly by removing from the triangulation all the simplices incident to v and retriangulating the 'hole' thus formed (see Figure 4.8(a) for the two-dimensional case).

In two dimensions, when the vertex v to be deleted has degree k, the hole created in DT(S) is a polygon that needs to be retriangulated with $(k-2)$ triangles. Although Chew (1990) proves that this problem can be solved in the optimal time of $\Theta(k)$, sub-optimal algorithms are usually preferred for an implementation because of their simplicity and because the average degree of a vertex in a 2D DT is only 6. The most elegant algorithms are based on the concept of the *ears* of a polygon (Devillers, 2002a; Heller, 1990). As shown in Figure 4.12, an ear is an 'imaginary' triangle (defined by three consecutive vertices on the hole) that could be used to fill the hole. Ear algorithms fill the hole gradually by inserting at each step an ear; this guarantees that the hole stays a simple polygon at all times. They exploit the fact that a DT is only modified locally by the insertion/deletion of a single vertex, and isolate the hole and its

[1]This section is based two papers: Ledoux et al. (2005) and Ledoux et al. (n.d.). The second paper has been under review for the *International Journal of Computational Geometry and Applications* for nearly a year; Franz Aurenhammer and Jack Snoeyink made informal reviews of a preliminary version.

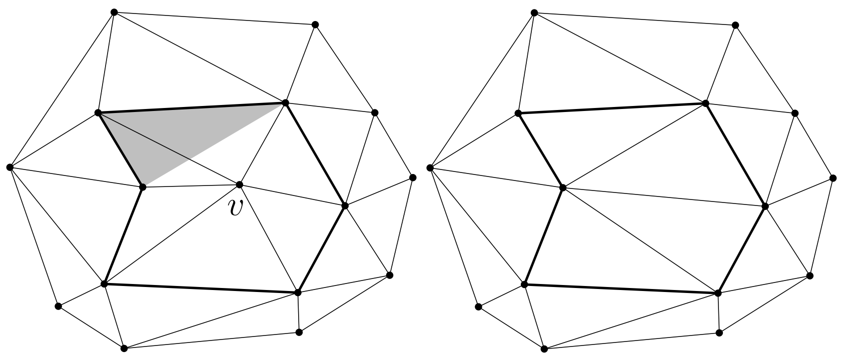


Figure 4.12: Left: DT(S) before v is deleted. The shaded triangle reprensents an ear of star(v). Right: DT($S \setminus \{v\}$.

vertices from the rest of the DT. Devillers (2002a) transforms the problem into the construction of the convex hull of the vertices of the hole lifted onto the paraboloid in three dimensions, and shows that the 'lifted ear' having the lowest intersection with the vertical axis is guaranteed to be Delaunay; he therefore proposes to process always that ear first (this is further explained in Section 4.4.5.1). Mostafavi et al. (2003) propose a simpler criterion for processing the ears: an ear is Delaunay if no vertices forming the hole is inside the circumcircle of the ear. These two algorithms have respectively a time complexity of $\mathcal{O}(k \log k)$ and $\mathcal{O}(k^2)$.

In three dimensions, the problem is similar: the hole, created by the deletion of all the tetrahedra incident to v, is a polyhedron. The major difference is that the number of tetrahedra needed to tetrahedralize the hole is not related to k, because the number of tetrahedra is not a function of the number of points in S (see Section 3.2.4).

As described in Section 3.3, most algorithms in computational geometry assume that inputs are in general position, and the handling of degeneracies are usually left to the programmers. Luckily, the deletion of a single vertex in a 2D DT does not have many special cases and its robust implementation is quite simple. However, the numerous degeneracies make the implementation of Devillers's and Mostafavi et al.'s algorithms impossible in three dimensions (impossible without the use of an extra mechanism that is), despite the facts that the former proved that his algorithm is valid in any dimensions, and that common sense suggests the latter algorithm generalises easily. The problems are caused by the fact that not every polyhedron can be tetrahedralized, as explained in Section 4.4.1. Devillers and Teillaud (2003) recognised that and used perturbations to solve the problem. Unfortunately, the major part of their paper is devoted to explaining the perturbation scheme used in `CGAL`[2] (Boissonnat et al., 2002), and very few details about the algorithm—a generalisation to three dimensions of Mostafavi et al.'s—are given. Shewchuk (2000) also proposes a sweep algorithm for constructing the constrained Delaunay triangulation in $\mathbb{R}^d$ of a star-shaped polytope, which can be used for deleting a vertex in a Delaunay tetrahedralization. However, his algorithm is only guaranteed to work correctly if a perturbation scheme is used to ensure that no five vertices lie on a common sphere.

In this section, I propose a new algorithm to delete a vertex v in a Delaunay tetrahedralization, and I show that, instead of creating a 'hole' in the tetrahedralization, it is possible to tackle the problem differently and use bistellar flips to modify the configuration of tetrahedra

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

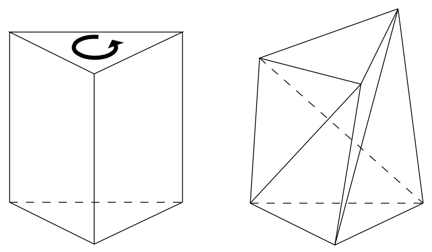


Figure 4.13: The Schönhardt polyhedron is impossible to tetrahedralize.

incident to v. As for the construction of a DT, flipping has many advantages since a complete tetrahedralization is kept during the whole deletion process. More importantly, I present two different methods to ensure the algorithm is robust against every degenerate case. The first one perturbs symbolically cospherical vertices, and the second one, called *unflipping*, involves going backward and modifying locally some tetrahedra to ensure that DT$(S \setminus \{v\})$ has a tetrahedralization.

4.4.1 Tetrahedralization of a Polyhedron

While any polygon in two dimensions can be triangulated, some arbitrary polyhedra, even if they are star-shaped, cannot be tetrahedralized without the addition of extra vertices, the so-called Steiner points. Figure 4.13 shows an example, as it was first illustrated by Schönhardt (1928). This polyhedron is formed by twisting the top face of a triangular prism to form a 6-vertex polyhedron having eight triangular faces (each one of the three quadrilateral faces adjacent to the top face will fold into two triangles). It is impossible to select four vertices of the polyhedron such that a tetrahedron is totally contained inside the polyhedron, as none of the vertices of the bottom face can directly 'see' the three vertices of the top triangular face.

For the deletion problem, we are interested in a special case of this problem: the polyhedron is formed by the union of tetrahedra in a DT, and is star-shaped (but not necessarily convex).

Lemma 4.3. *Let P be a polyhedron formed by the union of all the tetrahedra incident to a vertex v in a DT. The tetrahedralization of P is always possible, and moreover with locally Delaunay tetrahedra.*

Proof. Let $\mathcal{T}$ be a DT(S). If v is added to $\mathcal{T}$, thus getting $\mathcal{T}^v = \mathcal{T} \cup \{v\}$, with an incremental insertion algorithm (Watson, 1981; Joe, 1991), all the tetrahedra in $\mathcal{T}$ whose circumsphere contain v will be deleted; the union of these tetrahedra forms a polyhedron H. Then, H will be retetrahedralized with many tetrahedra all incident to v. Now consider the deletion of v from $\mathcal{T}^v$. All the tetrahedra incident to v must be deleted, their union is the polyhedron P. Notice that v could actually be any vertices in S, as a DT is unique and not affected by the order of insertion. The polyhedron P is exactly the same as H, therefore $\mathcal{T}$ tetrahedralizes P with Delaunay tetrahedra. □

4.4.2 Ears of a Polyhedron

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

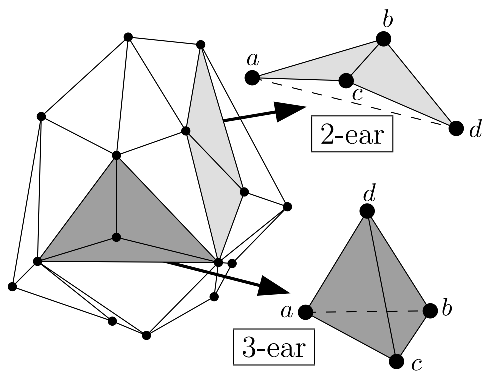


Figure 4.14: Perspective view of the outside of a polyhedron. Two adjacent triangular faces (e.g. in light grey) form a 2-ear, and three triangular faces incident to the same vertex (e.g. in dark grey) form a 3-ear.

Figure 4.14, 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 identify 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 4.14, 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.

4.4.3 ‘Flipping’ an Ear

In the case of the deletion of a vertex v in a DT, the polyhedron we are concerned with is star(v), and the ears of star(v) are defined by the triangular faces of link(v). Flipping an ear σ means creating the tetrahedron spanned by the four vertices of σ, by using a bistellar flip to modify the configuration of the two or three tetrahedra inside star(v) that are incident to σ. As a result, the newly created tetrahedron τ will not be incident to v anymore, and both star(v) and link(v) will be modified.

As shown in Figure 4.15, different flips are applied to different types of ears:

1. let σ be a 2-ear defined by the four vertices a, b, c and d. A flip23 performed on the two tetrahedra inside star(v) and incident to σ ($abcv$ and $bcdv$) creates the tetrahedra $abcd$, $abvd$ and $acvd$. The results of that flip23 are that, first, the tetrahedron $abcd$ is not part of star(v) anymore; and second, link(v) is modified as σ is replaced by a new ear σ' formed by the triangular faces abd and acd. The modification made to link(v) is equivalent to a two-dimensional flip22 of one edge. A 2-ear σ is said to be *flippable* if and only if the union of the two tetrahedra inside star(v) and incident to σ is a convex polyhedron.

2. let σ be a 3-ear defined by the four vertices a, b, c and d. A flip32 performed on the three tetrahedra inside star(v) and incident to σ ($abdv$, $bcdv$ and $acdv$) creates the two tetrahedra $abcd$ and $abcv$. After the flip32, the tetrahedron $abcd$ is not part of star(v)

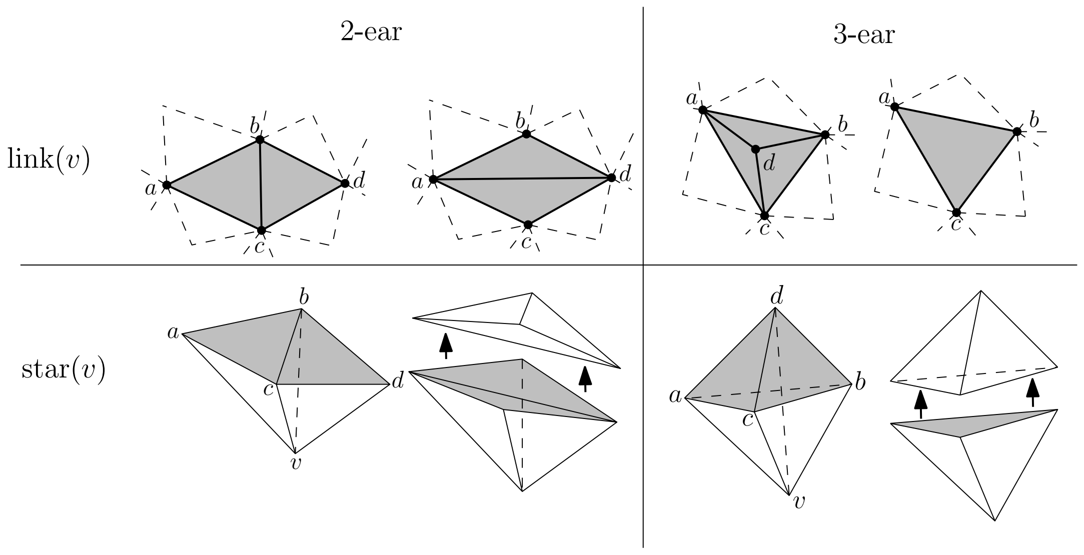


Figure 4.15: Flipping of an ear. For both 2- and 3-ears, star(v) and link(v) are modified by the flip.

anymore, and link(v) has now only one face abc instead of three. The modification made to link(v) is equivalent to a two-dimensional flip31. Also, the degree of v is reduced by 1 by such a flip. A 3-ear is *flippable* if and only if vertices d and v are on each side of the face abc, i.e. v should be 'outside' σ.

4.4.4 A Flip-based Deletion Algorithm

This section describes a new algorithm for deleting a vertex v in a DT(S) of a set S of points in $\mathbb{R}^3$ in general position. The algorithm is an extension to three dimensions of an ear algorithm (Devillers, 2002a; Heller, 1990), which means that the updates to the tetrahedralization are done incrementally by creating one Delaunay tetrahedron at each step of the process. Unlike other known approaches, the tetrahedra incident to v are not deleted from DT(S); DT($S \setminus \{v\}$) is obtained by restructuring the configuration of tetrahedra incident to v with a sequence of bistellar flips.

The algorithm, called DeleteInSphere and detailed in Algorithm 4.5, proceeds as follows. First, all the ears of star(v) are built and stored in a simple dynamic list; note that both valid and non-valid ears must be stored since as flips will be performed, link(v) and star(v) will be modified and non-valid ears will become valid, and vice versa. The general idea of the algorithm is to take an ear σ from the list (any ear, there is no particular order in which they should be processed) and flip it if all these three conditions are respected: σ is valid, σ is flippable and σ is locally Delaunay. An ear σ is locally Delaunay if its circumsphere does not contain any other vertices in link(v); this is simply tested with one InSphere test for each vertex in link(v). If one or more of the three conditions are not respected, the ear is simply not processed and the algorithm continues with the next ear in the list. As flips are performed, star(v) 'shrinks' and the configuration of the tetrahedra inside it changes; at each step of the process star(v) contains all the tetrahedra currently incident to v. The process stops when star(v) contains only four tetrahedra, and a flip41 deletes v.

Algorithm 4.5: DeleteInSphere (DT(S), v)

```
Input: DT($S$); the vertex $v$ to delete
Output: DT($S \setminus \{v\}$)
// let $L$ be a simple dynamic list
1  $L \leftarrow$ build a list of ears of star($v$), valid or not
2  while degree of $v > 4$ do
3      $\sigma \leftarrow$ get an ear from $L$ (anyone, order is not important)
4      if $\sigma$ is valid and $\sigma$ is flippable and $\sigma$ is locally Delaunay then
5          if $\sigma$ is a 2-ear then
6              flip23($\sigma$)
7              delete $\sigma$ from $L$
8              add new ear created $\sigma'$ to $L$
9          end
10         if $\sigma$ is a 3-ear then
11             flip32($\sigma$)
12             delete $\sigma$ from $L$
13         end
14     end
15 end
16 remove $v$ with a flip41
```

Time complexity. The time complexity of the algorithm is $\mathcal{O}(f\,k)$, where f is the number of ears of star(v), and k its degree (number of edges incident to v). Each ear needs to be tested against the k vertices in link(v). Notice that as flips are performed, star(v) shrinks and the number of vertices in link(v) decreases. A flip is assumed to be performed in constant time because only a finite number of adjacent tetrahedra are involved.

Data structure. The data structure to store the ears also needs to be discussed because of two difficulties: a 3-ear is formed by three overlapping 2-ears, and after a flip 2-ears can become 3-ears, and vice versa. For simplicity and ease of implementation, I propose storing only the 2-ears of star(v) and identifying 3-ears 'on the fly'. Before processing a 2-ear σ, a simple test, that takes constant time, can tell if σ is part of a 3-ear: if the union of the two tetrahedra defining σ is non-convex and if exactly three tetrahedra inside star(v) are incident to an edge of the common face between the two tetrahedra, then σ is one of the three 2-ears forming a 3-ear. For example, referring to the bottom right of Figure 4.15, assume we are processing the 2-ear *abcd*. Because the union of tetrahedra *acdv* and *bcdv* is non-convex, we check if there exists one tetrahedron (one and only one) that is adjacent to both the tetrahedron *acdv* and the tetrahedron *bcdv*; in that case, the ear *abcd* is actually a 3-ear because of the tetrahedron *abdv*.

Also, observe that a 2-ear can be identified by an edge in link(v) (only two triangular faces in link(v) are incident to the edge), and that each edge in link(v) is incident to exactly one triangular face in star(v). For the algorithm DeleteInSphere, we can therefore conceptually store the ears by simply storing all the triangular faces incident to v in star(v); because a complete tetrahedralization is kept at all time, the topological relationships permit us to have access to the two incident tetrahedra.

Correctness. The correctness of DeleteInSphere is proved by using results from the inverse operation: the insertion by flips of a single point v in a Delaunay tetrahedralization.

Lemma 4.4. *There always exists at least one sequence of bistellar flips to delete a vertex v in a Delaunay tetrahedralization.*

Proof. Let $\mathcal{T}$ be a DT in $\mathbb{R}^3$, Joe (1991) proves that there exists at least one sequence l of flips to construct $\mathcal{T}^v = \mathcal{T} \cup \{v\}$ (see Lemma 4.2). Section 4.2.3 shows that every flip has an inverse. Therefore, performing the inverse flips of l in reverse order trivially deletes v. □

We must now show that DeleteInSphere will always find one sequence of flips and that it will not get stuck or enter an infinite loop. Again, consider a DT $\mathcal{T}$, and let $\mathcal{T}^v = \mathcal{T} \cup \{v\}$ be the DT after the insertion of v, and $\mathcal{T}' = \mathcal{T}^v \setminus \{v\}$ the DT after the deletion of v from $\mathcal{T}'$. Notice that $\mathcal{T}'$ must be equal to $\mathcal{T}$ since we assume general position in the set S. With DeleteInSphere, if an ear σ of star(v) in $\mathcal{T}^v$ is Delaunay, that implies that the tetrahedron τ spanned by the four vertices of σ was present in $\mathcal{T}$. It also implies that τ was destroyed after the insertion of v in $\mathcal{T}$ since it does not exist in $\mathcal{T}^v$. The inverse flip that was used to destroy τ can trivially reintroduce it. The m tetrahedra that were destroyed by m flips after the insertion of v in $\mathcal{T}$ will have to be rebuilt with exactly m inverse flips when v is deleted from $\mathcal{T}^v$.

Consider now $\mathcal{T}^v$ after certain flips have been performed to insert v. Joe (1991) proves that if $\mathcal{T}^v$ is not Delaunay yet, then there exists at least one triangular face of link(v) that is flippable. This case is geometrically equivalent to a Delaunay ear that is flippable during the deletion algorithm, as the two processes are exactly inverse. The list of ears in DeleteInSphere contains all the potential tetrahedra that could be used to retetrahedralize star(v). That means that there will always be at least one Delaunay ear that is flippable until v is deleted. Also, if a Delaunay ear is flipped, the tetrahedron τ that is reintroduced in $\mathcal{T}'$ is not incident to v. Because the future flips will only affect the tetrahedra incident to v, τ cannot be deleted again. Which means that a cycle is not possible during the deletion algorithm.

4.4.5 Degeneracies

Degenerate cases occur when one of the following two conditions arises:

1. the set of points S is not in general position.
2. the vertex v to be deleted lies on the boundary of conv(S).

The algorithm DeleteInSphere as detailed in Algorithm 4.5 is not valid for deleting a vertex on the boundary of conv(S). This case can nevertheless be easily avoided by using a big tetrahedron, as described in Section 4.2.1. If one wants to delete a vertex that lies on the boundary of conv(S) but wants to avoid the use of a big tetrahedron, a sweep algorithm described in Shewchuk (2000) for deleting a point from a constrained DT can be used.

The coplanarity of four or more points in S leads to the use of the degenerate bistellar flip44; it is explained below how to use them.

When five or more points in S are cospherical, DT(S) is not unique. Let $\mathcal{T}$ be a DT(S) containing five or more cospherical points, and let v be a point located somewhere 'inside' the cospherical points. Consider the insertion of v in $\mathcal{T}$, thus getting $\mathcal{T}^v$, followed immediately by its deletion to get the tetrahedralization $\mathcal{T}'$. The tetrahedralization $\mathcal{T}'$, although being a valid DT, will not necessarily be the same as $\mathcal{T}$. Cospherical points will introduce an ambiguity as to which flips should be performed to delete v in $\mathcal{T}^v$. A flip used to delete v from $\mathcal{T}^v$, although possible and performed on a Delaunay ear, is not necessarily the inverse of a flip that was used to construct $\mathcal{T}^v$. This is not a problem in itself, except when some points in S are both cospherical and coplanar at the same time. In that case, unfortunately, one or more of these 'wrong' flips can lead to a polyhedron star(v) that is impossible to tetrahedralize. I propose two solutions to deal with this problem. The first one is to *prevent* an untetrahedralizable polyhedron by perturbing vertices to ensure that a DT is unique even for degenerate inputs. I describe in Section 4.4.5.3 how symbolic perturbations of cospherical points can be implemented. The main disadvantage of this method is that the same perturbation scheme must be used for all the operations performed on a DT. As a result, if one only has a DT and does not know what perturbation scheme was used to create it, then this method cannot be used. It is of course always possible to modify a DT so that it is consistent with a given perturbation scheme, but that could require a lot of work in some cases. Also, for some applications, using perturbations is not always possible. An example is a modelling system where points are moving while the topological relationships in the DT are maintained. It would be quite involved to ensure that the DT is consistent at all times with a perturbation scheme. The alternative solution consists of *recovering* from an untetrahedralizable star(v) by modifying the configuration of some triangular faces on its boundary; such an operation requires the modification of some tetrahedra outside star(v). This new method is called 'unflipping', as it goes backward and undo some of the work previously done, and is described below.

4.4.5.1 Deletion with Power Function Does Not Work with some Degenerate Inputs

Devillers (2002a) shows that the deletion problem can be solved by using the parabolic lifting map described in Section 3.2.3. His algorithm is theoretically valid in any dimensions, but consider here the case for a set S of points in $\mathbb{R}^3$ and a vertex v to be deleted. Let H be a subset of S formed by the vertices in link(v), and let σ be a valid ear of star(v). The problem of retetrahedralizing star(v) is transformed into the construction of conv(H^+), H^+ being the set H lifted onto the 4D paraboloid (as explained in Section 3.2.3 on 41). To simplify the visualisation in the following, let us translate the coordinates of every points in H by $-v$ and lift them to H^+ again (see Figure 4.16(a) for a one-dimensional example). First, notice that every σ_i lifts to a hyperplane σ_i^+, and that v lies under every σ_i^+ since it is inside the circumsphere of every σ_i. Observe also that the hyperplane σ_i^+ having the lowest intersection with the axis orthogonal to $\mathbb{R}^3$ is guaranteed to be part of conv(H^+). Indeed, the hyperplane σ_i^+ with the lowest intersection guarantees that every other point involved in the deletion is above σ_i^+, hence outside the circumsphere of σ_i. The construction of conv(H^+) can therefore be done by adding at each step the 3-simplex σ_i^+ whose intersection is the lowest. This order is a *shelling* order of conv(H^+) (Bruggesser and Mani, 1971), and gives an enumeration of the faces of a polytope in an order such that all the faces remain simply connected. For a polytope P formed by the faces of conv(H^+), the shelling order at point v is defined by the order in which an observer, starting at v and travelling along the axis orthogonal to $\mathbb{R}^3$, sees the faces disappearing. This actually enumerates only half of the faces of P but here we are interested in only the lower part of conv(H^+). Devillers's algorithms can theoretically be implemented with flips because there exists a relation between the ordering of flips when inserting a vertex v in a d-dimensional DT of a set S of points and the shelling order of a $(d+1)$-dimensional polytope. Indeed, Rajan (1991) proves that, in order to insert a new point v in a d-dimensional

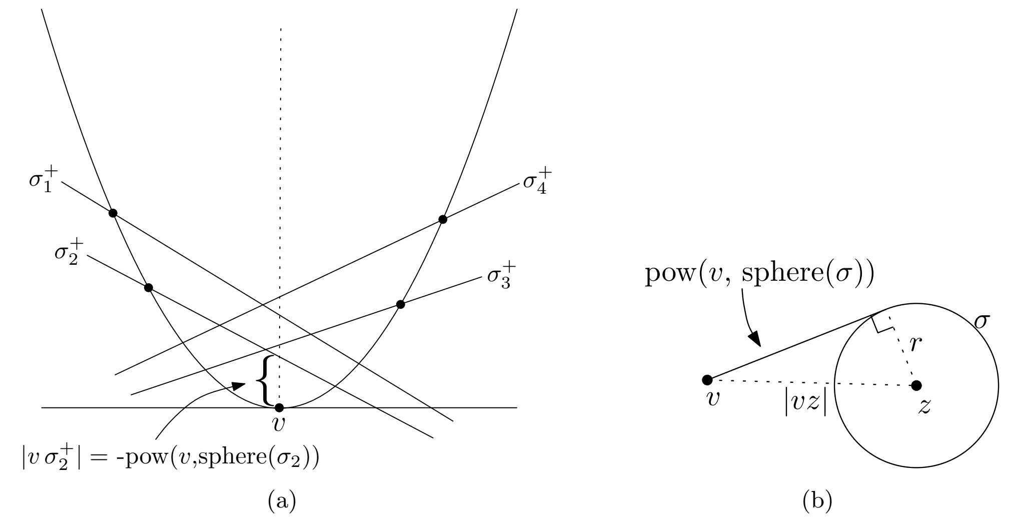


Figure 4.16: (a) The relation between power distance and 'vertical distance' shown for the one-dimensional case. **(b)** The power distance of a point v with respect to a circle σ.

DT, the shelling order at v of the lifted points S^+ gives the ordering of flips, that is an order where all the flips will be possible without having unflippable adjacent tetrahedra. Therefore, reversing the shelling order at v gives a flipping order (by using the inverse flip at each step) to delete v from DT(S); Devillers (2002a) simply describes the data structure that must be used to obtain this order.

Devillers's algorithm keeps a priority queue of all the ears σ_i sorted according to the distance, along the axis orthogonal to $\mathbb{R}^3$, between v and σ_i^+. This distance is equal to -pow(v, sphere(σ_i)), the *power distance* of v with respect to the circumsphere of σ_i. As shown in Figure 4.16(b), it is defined by

$$pow(v, sphere(\sigma)) = |vz|^2 - r^2 \,, \tag{4.3}$$

where $|vz|$ is the Euclidian distance between v and the centre z of the circumsphere of σ (denoted sphere(σ)), and where r is the radius of sphere(σ). As shown in Aurenhammer (1987), its value in three dimensions is equal to

$$pow(v, sphere(\sigma)) = \frac{-InSphere(\sigma, v)}{Orient(\sigma)} \,, \tag{4.4}$$

and is negative (positive) if v is inside (outside) sphere(σ), and zero if v is directly on sphere(σ).

The queue of ears eliminates the need to test each ear with all the vertices in link(v) with InSphere tests, and therefore speeds up the algorithm. A vertex v is deleted in $\mathcal{O}(t \log h)$ time, where t is the number of tetrahedra needed to retetrahedralize star(v) and h is the number of ears of link(v).

In two dimensions, this algorithm is elegant and simple to implement, but unfortunately it is not the case in three dimensions. Indeed, consider many cospherical points and a vertex v directly in the middle of the sphere. If v is deleted with Deviller's algorithm, then pow(v, sphere(σ_i)) of all the ears σ_i will be exactly the same but only some ears will be flippable (some configurations of tetrahedra will be non-convex). It is possible in that case to 'hold' some flips and process the

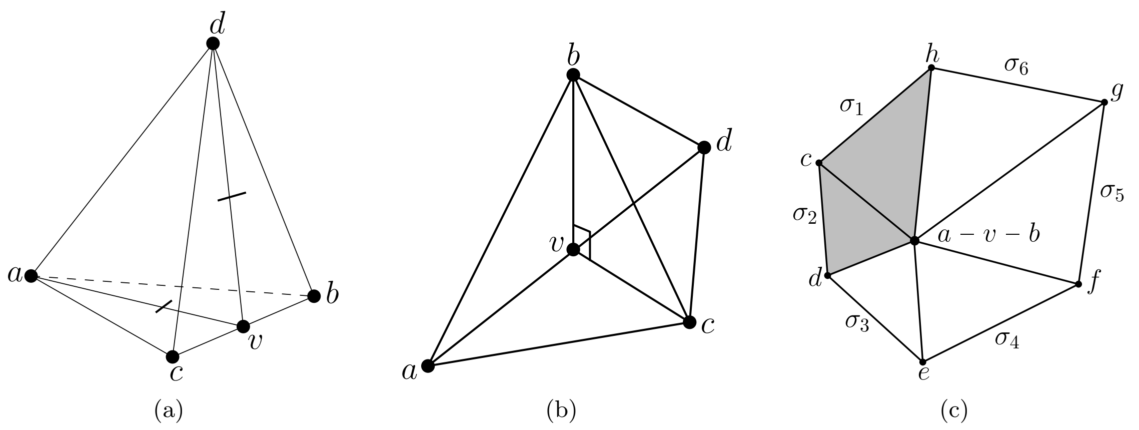


Figure 4.17: Coplanarity causes degenerate configurations of tetrahedra. **(a)** A flip41 can be performed with two flat tetrahedra. Tetrahedra *abcv* and *bcdv* are flat, a flip41 deletes v and creates the tetrahedron *abcd*. **(b)** The ear *abcd* is coplanar according to two planes (a, b, v and d are coplanar; and so are a, c, v and d). A flip23 on the ear *abcd* creates two flat tetrahedra *abvd* and *acvd*. **(c)** Possible configuration (viewed from the top) of star(v) after some flips if an oceanographic dataset is used. Every ear σ is in config44 according to two planes.

ear for which pow(v, sphere(σ_i)) is the second highest (or third, or fourth. . .). Unfortunately, holding flips does not always work because of untetrahedralizable polyhedra. Consider for example star(v), after some flips, whose shape is formed by an untetrahedralizable polyhedron 'attached' to an arbitrary polyhedron. All the ears on the untetrahedralizable polyhedron will be on 'hold', and it is possible that an ear σ_i, although not Delaunay, be flippable and be the one for which pow(v, sphere(σ_i)) is the highest. Such a flip would corrupt the algorithm and it is impossible to recover from that situation. The only solution is to prevent this case by using a perturbation scheme. However, the perturbation of the power function is more involved than the ones for InSphere or Orient because it has a higher algebraic degree (Devillers and Teillaud, 2003).

4.4.5.2 Handling Coplanarity with the Degenerate flip44

Let σ be a 2-ear of a polyhedron star(v), and let τ_1 and τ_2 be the two tetrahedra in star(v) incident to σ. Consider τ_1 and τ_2 to have four coplanar vertices and σ to be Delaunay (e.g. in Figure 4.15, *abdv* would be coplanar). A simple flip23 on σ is not always possible since it creates a flat tetrahedron; actually the flip23 is possible if and only if τ_1 and τ_2 are in config44 with two tetrahedra τ_3 and τ_4 that are in star(v) and incident to an ear σ_2 that is Delaunay. In that case, a flip44 will flip in one step both σ and σ_2. If τ_1 and τ_2 are not in config44, then σ cannot be flipped.

If four or more coplanar vertices are present in a DT, one must be aware of the presence of flat tetrahedra. They can be created during the process of updating a DT (after a flip), but no flat tetrahedron can exist in a DT. In DeleteInSphere, flat tetrahedra are permitted only if they are incident to v; an ear clearly cannot be flat because after being processed it becomes a tetrahedron of DT. Also, flat tetrahedra are allowed until the very end of the process because the last operation needed to delete v in a DT, the flip41, can be performed with two flat tetrahedra incident to v (see Figure 4.17(a)). Another way of eliminating a flat tetrahedron is with a flip32 performed on a 3-ear containing a flat tetrahedron.

A 2-ear σ can also be in config44 according to two different planes, that is when τ_1 and τ_2 are configured such that two subsets of their five vertices are coplanar (see Figure 4.17(b)). Such an ear must be handled with great care because flipping it creates two flat tetrahedra (one for each subset of coplanar vertices). For a flip44 to be performed on σ, τ_1 and τ_2 must be in config44 with tetrahedra according to both planes and the neighbouring ears must both be locally Delaunay.

For some degenerate set of points in which many points are coplanar, deleting a vertex is still a problem even if the flip44 is used. Examples of such degenerate sets are datasets collected in geology or in oceanography, where samples are respectively gathered from boreholes and water columns. The resulting datasets have many samples with the same $x - y$ coordinates and different depths. The deletion of a vertex v in the DT of such a dataset is tricky because many flat tetrahedra can potentially be created. Consider star(v), of a vertex v belonging to a borehole, after some ears have been flipped. It will be formed by many tetrahedra spanned by v, one vertex 'above' (or 'under') v in the same borehole and two vertices belonging to a neighbouring borehole, and most of the ears will be in config44 according to two planes. Viewed from above, star(v) will look like an umbrella. Referring to Figure 4.17(c), flipping σ_1 with a flip44 (with σ_2) creates the flat tetrahedra $vabd$ and $vabh$. If later σ_4 and σ_5 are flipped with a flip44, then star(v) does not stay 'simply connected', that is σ_6 is only connected to the rest of star(v) by the face avb having three collinear vertices. It is then impossible to flip σ_6 without creating another flat tetrahedron, which will be created 'over' one already present (both will be spanned by the same four vertices).

It is possible to recover from this situation by removing from the DT the two flat tetrahedra and updating the topological relationships between their adjacent tetrahedra, but this requires to destroy momentarily the tetrahedralization and could be tricky to implement. A simpler solution consists of preventing this situation by ensuring that star(v) remains a 'simple polyhedron' at all times. This is done simply by flipping an ear that is in config44 according to two planes if and only if it is the only possible flip among all the ears of star(v). For example, in Figure 4.17(c), if σ_1 and σ_2 are flipped first with a flip44, then σ_4 will not be flipped. But σ_3 will be flippable with a flip32 involving the flat tetrahedron $vabd$ and the tetrahedra $vade$ and $vbde$.

4.4.5.3 Symbolic Perturbations to Handle Cospherical Points

Perturbing a set S of points means moving by an infinitesimal amount the points to ensure that S is in general position. This is supposed to avoid many problems with the implementation of geometric algorithms; the construction of the DT is for example greatly simplified if there are no coplanar and no cospherical points. Unfortunately, moving points in $\mathbb{R}^3$ can have serious drawbacks: valid tetrahedra in the perturbed set of points can become degenerate (e.g. flat) when the points are put back to their original position (Seidel, 1998; Shewchuk, 2000).

In the case of the deletion of a vertex v in a DT in $\mathbb{R}^3$, only cospherical points really cause problems—they can lead to an untetrahedralizable polyhedron—since coplanarity can be handled with the flip44. I therefore describe in the following a method to perturb only cospherical points without actually moving them. The idea, which was first proposed in Edelsbrunner and Mücke (1990, Section 5.4), involves perturbing the points in $\mathbb{R}^{d+1}$ by using the parabolic lifting map. In $\mathbb{R}^3$, five points are cospherical if and only if the five lifted points are coplanar on the paraboloid in $\mathbb{R}^4$. Thus, each point of S is perturbed by a (very) small amount so that no five points in $\mathbb{R}^4$ lie on the same hyperplane. The method cannot be applied just for the deletion of a single vertex since the resulting tetrahedralization of star(v) would not necessarily be consistent with the tetrahedralization outside star(v). The main goal of using this method is having a unique DT even when five or more points in S are cospherical, so that there is a clear ordering of the flips to perform to delete v. The same perturbation scheme must therefore be used for every operation performed on the DT, including its construction.

The perturbation scheme used here to delete a vertex in a DT is consistent with the intuitive method a programmer would use to implement efficiently an algorithm to construct a DT (to avoid unnecessary operations), and is also consistent with the construction of the DT with InsertOnePoint previously described. When a new point v is added to a valid DT(S), if it lies directly on the circumsphere of an already present tetrahedron, then v is considered outside and the tetrahedron is not destroyed. This translates to perturbing sequentially the points in S and lifting them upward; the last point in S is perturbed the most by an infinitesimal amount, and the ones before by a smaller amount. Shewchuk (2000) adapted the scheme of Edelsbrunner and Mücke (1990) and simplified greatly its implementation. With his method, the amount by which each point is moved does not have to be calculated because the perturbations are implemented symbolically in the InSphere test. My implementation of the perturbed InSphere predicate is a slight modification of Shewchuk's method, and goes as follows. First, the points are perturbed only when they are cospherical; if they are not then InSphere returns either a positive or negative result. When InSphere returns zero, simply move upward the lifted point with the highest index by adding an arbitrary amount ϵ (the value is not important) to its fourth coordinate (the fourth coordinate of p becomes for example $p_x^2 + p_y^2 + p_z^2 + \epsilon$, see Equation 4.2 on page 48), and recompute InSphere. If the four remaining points are affinely independent, then the new result will be nonzero; if not then the points with the second highest index must be moved upward by a smaller value, and InSphere recalculated. This continues until the result of InSphere is nonzero. The perturbed InSphere test does not slow down the construction algorithm as it is not required to explicitly perturb the points. Indeed, all InSphere tests will be done with the last inserted vertex v and therefore if InSphere returns zero then v is always considered to be outside the sphere. It should be noted that the CGAL library uses a similar scheme (Boissonnat et al., 2002; Devillers and Teillaud, 2003), although the implementation is different.

By using the perturbed InSphere with DeleteInSphere, the case of untetrahedralizable polyhedra will not occur. It is nevertheless necessary to use the flip44 to handle coplanar points, as described in the previous section. It should also be pointed out that the implementation of a perturbation scheme is effective only if exact arithmetic is used for all the predicates involved; see Section 6.2 for more details.

4.4.5.4 Unflipping

If cospherical vertices are not perturbed for the deletion of vertex v, it is possible, for very degenerate sets S of points, that an intermediate star(v) (after some flips) be an untetrahedralizable polyhedron (UP). To demonstrate the cause of this and the solution proposed, I first need to state that Lemma 4.2 was proved by Joe (1991) even when the set S of points has degeneracies. He showed how to handle coplanar points, and, as we just saw in the previous section, cospherical points are easily handled. We thus have the following lemma:

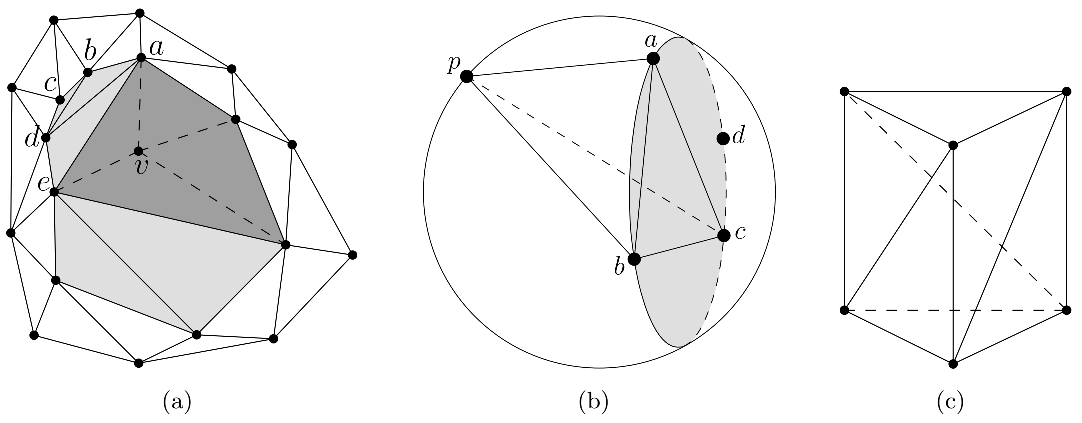


Figure 4.18: **(a)** Two-dimensional star(v) (dark grey) after some flips have been performed. The parts already retriangulated are in light grey. **(b)** Any four cocircular points a, b, c and d are cospherical with any other point p. **(c)** The smallest example of an untetrahedralizable polyhedron.

Lemma 4.5. *There always exists at least one sequence of bistellar flips to delete a vertex v in a Delaunay tetrahedralization of a set S of points in $\mathbb{R}^3$, even when S contains four or more coplanar points and/or five or more cospherical points.*

Although when S contains five or more cospherical points the DT is not unique, cospherical points are not the only culprit to produce a star(v) that is impossible to tetrahedralize.

Lemma 4.6. *Let v be a vertex in a Delaunay tetrahedralization $\mathcal{T}$ in which no four vertices are coplanar, but in which more than five vertices are cospherical. When deleting a vertex v, no intermediate star(v) will be an UP.*

Proof. The proof is by contradiction. Let star(v) be the polyhedron left to tetrahedralize after some flips have been performed (see Figure 4.18(a) for the 2D case, the 3D case follows the same idea). Suppose star(v) is an UP. It is impossible to tetrahedralize star(v) because some cospherical vertices (e.g. a, b, c, d and e) introduced an ambiguity in the sequence of flips that should be performed to delete v. Going backward to the original star(v) in $\mathcal{T}$ and performing another sequence of flips (to tetrahedralize the cospherical vertices differently) yields in every case the same intermediate star(v) that is impossible to tetrahedralize. This is because even in the presence of cospherical vertices, star(v) always stays a simplicial polyhedron. By Lemma 4.5, v can be deleted from $\mathcal{T}$, therefore star(v) cannot be an UP. □

Lemma 4.7. *A star(v) impossible to tetrahedralize can only happen when the set of points contains four or more cocircular points.*

Proof. First, notice that it is only in the presence of four or more coplanar vertices in link(v) that the polyhedron can become degenerate, i.e. a non-simplicial polyhedron having a face formed by four or more coplanar vertices. This face will be triangulated in link(v) during the process, but any triangulation would be valid.

Thus, to encounter an UP, link(v) must have five or more cospherical vertices (DT is not unique) and four or more coplanar vertices (degenerate star(v)/link(v)). Observe that cocircularity of some points implies that they are coplanar, and also that they are cospherical with some other point. Indeed, consider four cocircular points a, b, c and d (as in Figure 4.18(b)), these points are cospherical with any other points p because the circle they define will always be a cross-section of the sphere $abcdp$.

Now consider star(v), the polyhedron obtained after some flips when deleting v. If link(v) contains four or more cocircular vertices, the triangulation of these vertices in link(v) is not unique as it is the result of earlier flips where more than four points were cospherical. Note that any triangulation of the cocircular vertices would be valid, but only one was randomly picked. The impossibility to tetrahedralize some intermediate star(v) comes from this ambiguity: it can create an inconsistency between the possible tetrahedralization of star(v) and the tetrahedralization outside star(v) □

During the deletion of v, if an UP is encountered it will therefore contain four or more coplanar vertices, i.e. link(v) will have at least two coplanar triangular faces. Two incident coplanar faces on link(v) are called a *flat ear*. The smallest example of an UP, and also the most common during the experiments, is a triangular prism having the same triangulation as the Schönhardt polyhedron, i.e. each quadrilateral face is triangulated such that every vertex has a degree 4 (see Figure 4.18(c)). Notice that if only one diagonal of one quadrilateral face is flipped (think of a flip22 in a 2D triangulation) then the prism can easily be tetrahedralized with three tetrahedra. Thus, to recover from an untetrahedralizable star(v), I propose flipping the diagonal of one of its flat ear and continue the deletion process as usual afterwards. This is an iterative solution: one flip might be sufficient in some cases, but if another untetrahedralizable star(v) is later obtained then another diagonal must be flipped. Unflipping simply allow us to get one triangulation of link(v) that permits the tetrahedralization of star(v) (by Lemma 4.5 there is always one). Flipping the diagonal of a flat ear obviously involves the modification of the tetrahedra, inside and outside star(v), incident to the diagonal edge. The only way to flip the diagonal edge is with a flip44 when the tetrahedra—incident to the flat ear—inside and outside star(v) are in *config44*. Inside star(v), there will always be only two tetrahedra, but outside there can be more than two. Modifying the tetrahedra inside star(v) is allowed since they have not been processed yet, but the ones outside star(v) are by definition Delaunay. The following lemma shows that it is always possible to modify them.

Lemma 4.8. *Let star(v) be an intermediate polyhedron impossible to tetrahedralize. The vertices of the tetrahedra incident to a flat ear of star(v) and outside star(v) all lie on the same sphere.*

Proof. Refer to Figure 4.19. Let a, b, c and d be four cocircular vertices forming a flat ear of star(v), and triangulated with the triangles abc and bcd. Observe that at least two tetrahedra are incident to the ear and outside star(v) (some other tetrahedra could also be incident to the edge bc). Let $abcp$ be one such tetrahedron, and let ζ be its circumsphere. Because a, b, c and d are cocircular, the vertex d lies exactly on ζ.

If the tetrahedron incident to face bcd has p as its apex, then the proof is finished. Assume it is not the case, and let q be another vertex forming a tetrahedron $bcdq$. Because all the tetrahedra outside star(v) are Delaunay, q cannot be inside ζ, and it cannot be outside ζ either. Indeed, if q is located anywhere outside ζ (e.g. where q' is in Figure 4.19) the circumsphere of $bcdq$ will contain p, because, as the distance between the centre of ζ and q grows, the circumsphere of $bcdq$ grows and its centre always lies on a line perpendicular to the plane $abcd$. Therefore, q lies on ζ.

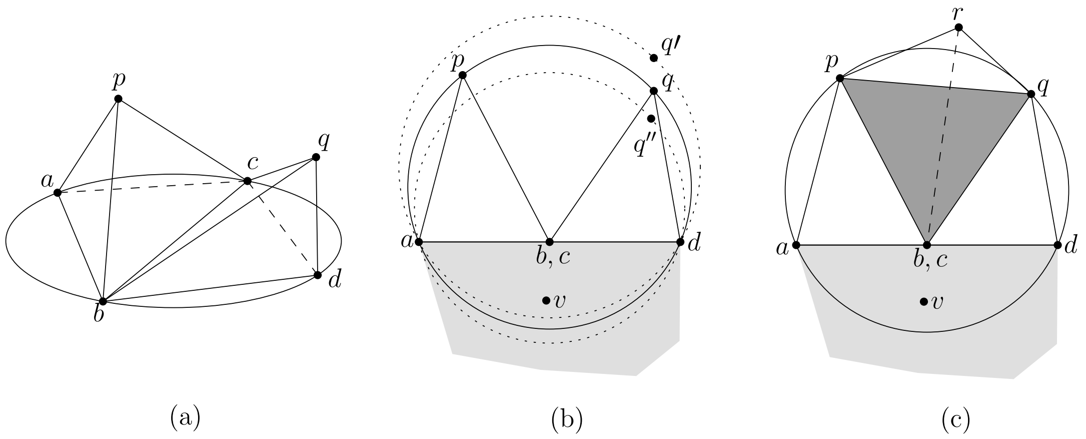


Figure 4.19: **(a)** Perspective view of the flat ear *abcd*, with two incident tetrahedra. **(b)** **(c)** Same case viewed if the observer were looking directly at the edge *bc*. The shaded area is the interior of star(v).

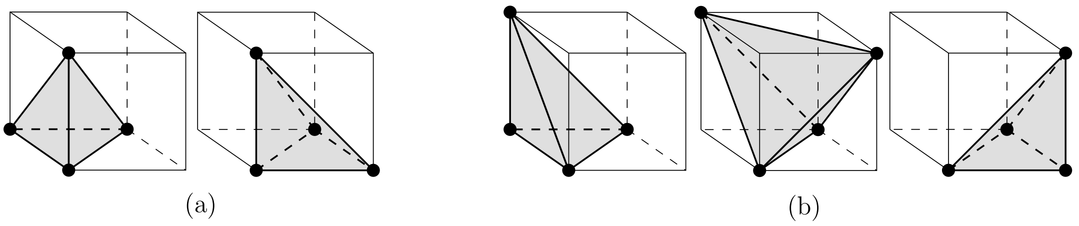


Figure 4.20: Two possible configurations when unflipping. The flat ear σ is the bottom face of the cube. **(a)** Two tetrahedra are incident to σ. **(b)** Three tetrahedra are incident to σ.

The only problem left is the tetrahedron or tetrahedra incident to the edge bc (lying between the tetrahedra $abcp$ and $bcdq$). Notice that $pbcq$ will be the only tetrahedron unless a face is spanned by another vertex r and the vertices b and c. The only way for any vertex r to be part of a tetrahedron incident to bc is if r lies directly on ζ. Indeed, r cannot be inside ζ, and if it lies outside ζ then the face bcr will not exist: the Delaunay tetrahedralization of the five vertices p, b, c, q and r must contain the tetrahedron $pbcq$ as r lies outside its circumsphere. □

The configuration of tetrahedra incident to a flat ear and outside star(v) will not always be the same, but it suffices to modify it such that only two tetrahedra τ_1^o and τ_2^o (τ_i^o represents the tetrahedra outside star(v)) that are in config44 with the two tetrahedra τ_1 and τ_2 inside star(v). Then a flip44 performed on τ_1, τ_2, τ_1^o and τ_2^o will change the diagonal of the flat ear. The modification of the tetrahedra outside star(v) is permitted by Lemma 4.8, and because all the vertices involved lie on a sphere (they form a convex polyhedron), the flips will be possible.

In practice, the most common case of an UP arises when most of the points in a set are distributed on a 3D grid, that is when the spacing between points in the $x-y-z$ directions is constant. In such a case, unflipping can be seen as modifying the tetrahedralization of a cube to make sure it is consistent with the one of adjacent cubes. Figure 4.20 shows two possible configurations for the tetrahedra incident to a flat ear σ and outside star(v). In the first configuration, only two tetrahedra τ_1^o and τ_2^o are adjacent to σ, thus already forming a config44. In the second configuration, three tetrahedra are incident to σ, and a flip23 on two of the three tetrahedra will modify the configuration such that only two tetrahedra are incident to σ. A flip44 is then possible.

4.4.6 Discussion

The problem of deleting a single vertex in a DT has highlighted the gap that sometimes exists between theory and practice; this gap is mostly caused by degeneracies that are more difficult to handle in higher dimensions. The algorithm of Devillers (2002a), although proved valid in any dimensions and easily implemented in two dimensions, does not generalise straightforwardly to three dimensions since a perturbation scheme must be used to ensure its robustness.

The alternative presented, DeleteInSphere, deletes a single vertex v in a DT in $\mathcal{O}(f\,k)$ time, where f is the number of ears of star(v) and k the number of edges incident to v. Although any vertex in a DT of n points in $\mathbb{R}^3$ can have at most $\mathcal{O}(n)$ incident tetrahedra, as explained in Section 3.2.4, when the points are distributed uniformly, the expected complexity of the DT is linear, and thus each vertex is expected to have $\mathcal{O}(1)$ incident tetrahedra. Also, when the set of points is distributed according to a Poisson distribution, a typical vertex v has around 15.5 incident edges and around 40.6 incident triangular faces (which gives the number of 2-ears of star(v)) (Okabe et al., 2000). Despite the fact that DeleteInSphere is sub-optimal, it is efficient in practice since k and f are generally small for most input.

In the absence of degenerate cases, the implementation of DeleteInSphere is relatively simple, but as I have shown, different methods must be used to ensure its robustness. To make it fully robust for any configurations of data, several things must be done. First, the coplanarity of vertices must be handled with the degenerate flip44. Second, cocircular/cospherical vertices, which may lead to an untetrahedralizable polyhedron, must be handled with either symbolic perturbations or unflipping. The latter method is independent of any ordering of the vertices or of the knowledge of previous operations performed on the DT, and has to be used for some applications, e.g. when in a kinetic system is used, or when an ordering of points is not wished or impossible.

4.5 Moving Points in a Delaunay Tetrahedralization

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, because it 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). De Fabritiis and Coveney (2003) use for instance 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 (2004) found that continuous updates permit them

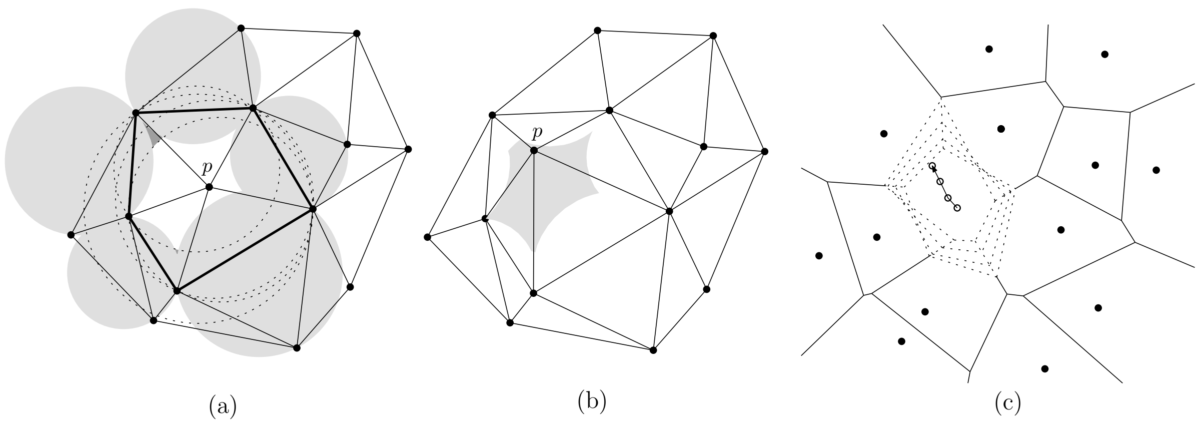


Figure 4.21: **(a)** The combinatorial structure of the DT will not change as long as the vertex p is moved within the white polygon. **(b)** p has moved but S is still in general position. The combinatorial structure of DT(S) has not changed. Only the location of p has changed, and so have the edges incident to it. **(c)** Example of the consequences of moving p on the VD.

to update the VD/DT in approximately half to three quarters the time it takes to recompute the whole 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 4.2. *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 Figure 4.21).*

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 arise, the topological structures of VD(S) and DT(S) will be modified; the critical moments are called *topological events*. Observe in Figure 4.21(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 4.2 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.

4.5.1 Two-dimensional Work

The Observation 4.2 was used by Roos (1991) who analysed the complexity of the movement of points in a two-dimensional VD, and proposed an algorithm to update the VD. As is the case in this thesis, although he discusses the movement of points in a VD, the algorithm he developed is based on the dual. When a topological event arise, the update to the DT is made with a bistellar 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) 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.

Similar algorithms have also been proposed, see for instance Bajaj and Bouma (1990) and Imai et al. (1989). Moreover, Gold (1990) and Gold et al. (1995) (with more details in Mostafavi (2001) and Mostafavi and Gold (2004)) 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.

4.5.2 Three-dimensional Work

The work of Roos (1991) has been generalised to three- and higher-dimensional space by Albers et al. (Albers, 1991; Albers and Roos, 1992; Albers et al., 1998). 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 (2003) 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 Albers et al.'s, but they show how the InSphere test must be modified to consider the radius of each ball. Neither of these algorithms discuss degenerate cases, or how collisions are handled.

The major impediment to the implementation of Albers et al.'s algorithm in three dimensions is that, as Gavrilova and Rokne (2003) observe, calculating the zeros of the function InSphere can not 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. (2004) 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 (1991), but 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 available in related disciplines where there is a real need. Ferrez (2001) and Schaller and Meyer-Hermann (2004) 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 4.2, 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 could 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.

4.5.3 A Flip-based Algorithm

I discuss in the following a new algorithm 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 (Gold, 1990; Gold et al., 1995; Mostafavi, 2001; Mostafavi and Gold, 2004). Unlike Albers et al.'s method where all the pairs of simplices must be tested, the algorithm I present permits us to move 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 bistellar flips and is conceptually very simple. Most of the basic concepts and operations required for inserting a point and deleting a vertex (flipping, ears of a star, flipping of an ear, etc.) are used directly for the movement of a point, and not many extra operators are necessary. This ensures that the implementation of the algorithm is fast and relatively easy. The implementation of the movement of a single point actually proved to be much simpler than the deletion problem, as problems with untetrahedralizable polyhedra do not arise.

4.5.3.1 Types of tetrahedra

Three types of tetrahedra, with respect to the moving point p, must be defined (see Figure 4.22(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 4.4.2. 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

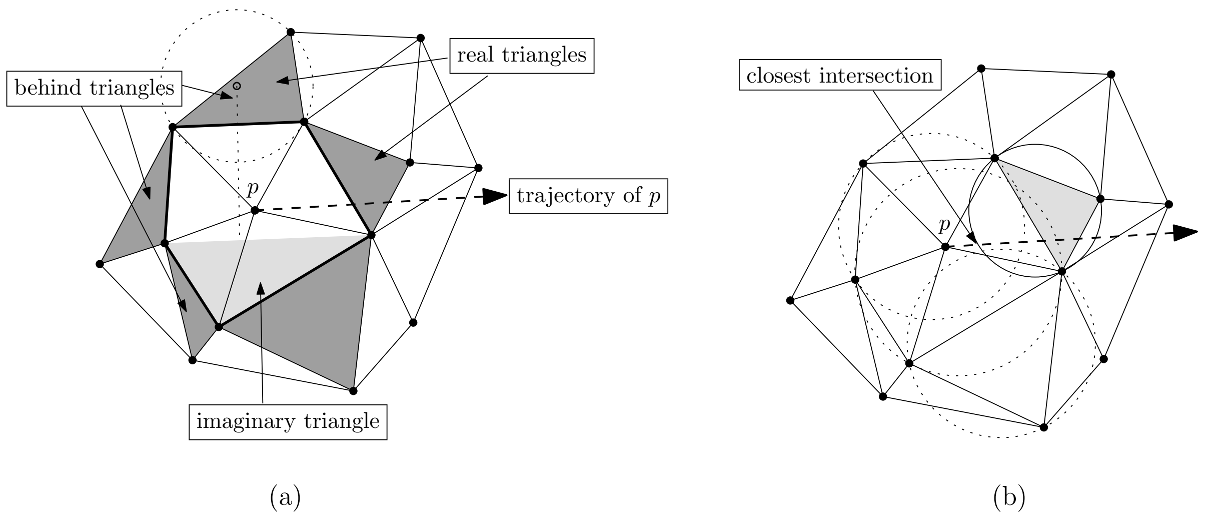


Figure 4.22: **(a)** The different types of triangles needed to move the vertex p along the trajectory. Real triangles are the dark shaded ones, and one example of an imaginary triangle is light shaded. Notice that a behind triangle is always also a real triangle. **(b)** p must be moved to the closest intersection of a circumcircle along the trajectory. The triangle having the closest intersection is the shaded triangle (a real triangle).

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 Figure 4.22(a).

4.5.3.2 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 Figure 4.22(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 cases possible (this is illustrated in Figure 4.23):

(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 (Algorithm 4.6), is for moving a single point in a set S (while all the other ones are fixed). It is assumed in this section that S is in general position; the degenerate cases are discussed in the next section. 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 might 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 traversing all the tetrahedra forming star(p) with a BFS-like algorithm on the dual of the tetrahedra. This

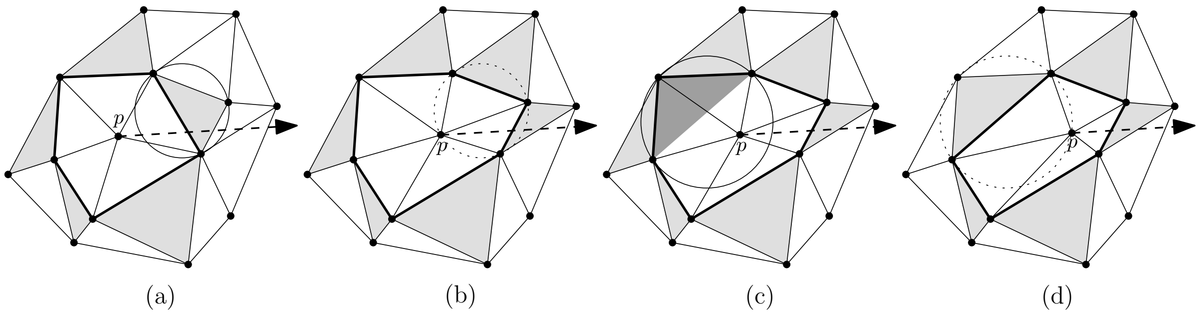


Figure 4.23: Two cases are possible when p is moved along a trajectory. At all times, the real triangles are the light shaded ones. **(a)** The closest intersection is with a real triangle. **(b)** p is moved to the circumcircle, a flip22 is performed and the real triangles are updated. **(c)** The next 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.

Algorithm 4.6: MoveOnePoint($\mathcal{T}$, p, x)

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

initialise $B$ // let $B$ be a simple dynamic list
while $p$ is not at location $x$ do
    $\tau_1 \leftarrow$ get tetrahedron (real or imaginary) having closest intersection with trajectory
    move $p$ to intersection
    if $\tau_1$ is a real tetrahedron then
        $\tau \leftarrow$ get tetrahedron inside star($p$) adjacent to $\tau_1$
        if $\tau \cup \tau_1$ is convex then
            // this is equivalent to case#1 in Figure 4.11
            flip23($\tau, \tau_1$)
        else
            // case#2 in Figure 4.11
            flip32($\tau, \tau_1, \tau_2$)
        end
    else
        // $\tau_1$ is an imaginary tetrahedron
        if $\tau_1$ is a 2-ear then
            remove from $B$ the 2 tetrahedra outside star($p$) incident to $\tau_1$
            flip23($\tau_1$)
            add $\tau_1$ to $B$ // the ear becomes a behind tetrahedron
        else
            // $\tau_1$ is a 3-ear
            remove from $B$ the 3 tetrahedra outside star($p$) incident to $\tau_1$
            flip32($\tau_1$)
            add $\tau_1$ to $B$ // the ear becomes a behind tetrahedron
        end
    end
end
```

engineering decision also depends on the data structure used to store the DT (see Section 6.3 for different possibilities).

In order to get the tetrahedron (real or imaginary) whose circumsphere has the closest intersection, a simple test that computes the intersection between a line segment and a sphere is used. As explained in Bourke (1992), 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. (1998) 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 Figure 4.23, 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.

4.5.3.3 Degeneracies

The degenerate cases for a moving point are easier to handle than the ones for the insertion or deletion algorithms, mostly because no points are added or deleted (and thus we have complete tetrahedralization at all times), and also because problems with untetrahedralizable polyhedra can not happen.

In the context of this thesis, the principal difficulty when implementing MoveOnePoint is caused by the floating-point arithmetic used by modern computers. The issues are explained in Section 6.2, but it basically means that some calculations are not exact, and the small errors can have disastrous consequences. These calculations are not related to the two predicates Orient and InSphere (these are robust in the prototype developed for this research), but to other functions, particularly the one computing the intersections between the trajectory and the circumspheres. Because of these errors and of degenerate cases, it was found that implementing a *flippability* test was helpful, i.e. before moving a point to the closest intersection, we test if a flip will be possible. This test is easily performed, since the tetrahedron whose intersection is involved is known, and the usual test, as described in Section 4.3.2 for InsertOnePoint, is used.

What follows is a list of the degenerate cases that can arise during the movement of a single point at a time. If robust arithmetic was used for all the calculations (which would also slow down the implementation tremendously), most of these degenerate cases would not cause problem.

Cospherical points. The flippability test is required when many points are cospherical because not every tetrahedron spanned by the cospherical points can be flipped. When p is moved to the circumsphere of five or more points, we must verify which tetrahedron/tetrahedra can be flipped.

Floating-point arithmetic can also create problems when many points are cospherical, or nearly cospherical. The intersection tests between the trajectory and the circumspheres could return that a given tetrahedron has the closest intersection, but in reality it is another one, and thus the flippability must be used to ensure that the combinatorial structure of the VD/DT is not corrupted. In brief, even if a given tetrahedron has the closest intersection, if a flip can not be performed the tetrahedron is simply ignored and the second closest tetrahedron is processed. Observe that this test does not have to be performed each time the closest intersection is sought, but only when two or more intersections are very close to each other (a tolerance can be used).

Moving backwards. To solve the potential problems that can arise with floating-point arithmetic, it was found that allowing the moving point to move 'backwards' helps. In other words, sometimes moving backwards the point p and performing a flip unlocks a situation and the algorithm can continue. This permits us to correct 'errors' that could have happened before. Notice that the movements in the opposite direction are only for very short distances, e.g. when the closest intersection of a real tetrahedron is slightly before the trajectory, and that the flippability test must be performed before moving backwards.

Coplanar points. The presence of four or more coplanar points is not problematic in itself because it is guaranteed that a flip is always possible (Albers et al., 1998). But because of calculation errors, it is worth verify if a config44 is present before performing a flip23, to avoid deadlocks.

Sphere tangential to the trajectory. In the situation where the closest intersection with a sphere is when the trajectory is tangential with the sphere (only one intersection), the sphere can be ignored and the second closest intersection can be processed. Tangential spheres can be ignored because it does not matter if the flip is performed or not, so it is better to avoid unnecessary operations.

Here again floating-point arithmetic can cause problems when a trajectory is very close to a sphere: a non-intersecting case could be calculated as intersection, and vice versa.

Collisions. Depending on the application, different solutions must be considered. If the points are real objects, e.g. boats in two dimensions, then collisions are obviously problematic. But in the case of a simulation in geoscience for instance, points simply represent buckets of water or air, and collisions have no meaning. MoveOnePoint must nevertheless consider this case. A simple solution was implemented: when p is about to collide with another point $q \in S$, simply delete p in $\mathrm{DT}(S)$, and reinsert it right after q along the trajectory. The collision detection is performed by testing the distance between p and the points forming $\mathrm{link}(p)$ against a tolerance, before moving p to the closest intersection. Observe that in that case, DeleteInSphere must be made robust with the unflipping method, as keeping $\mathrm{DT}(S)$ consistent with a perturbation scheme is (almost) impossible.

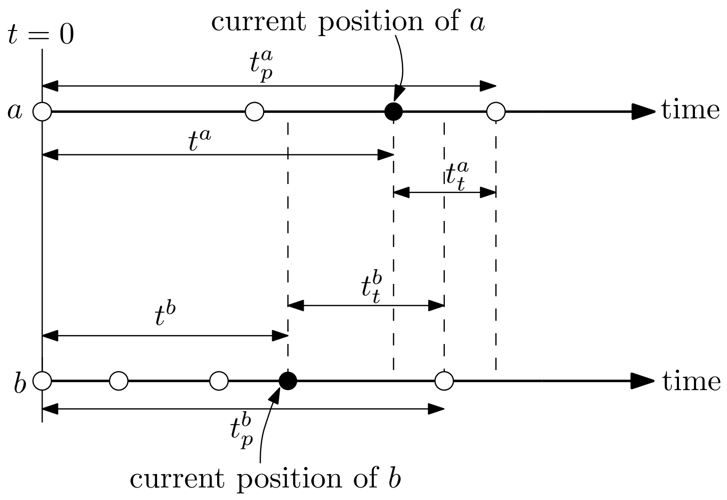


Figure 4.24: The three types of time needed for moving several points at the same time. The next movement to be made is for point b, since t_p^b is before t_p^a. (Figure after Mostafavi and Gold (2004))

4.5.3.4 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 (2004), 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 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:

$$t_p = t + t_t\,. \tag{4.5}$$

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 Figure 4.24 for the movement of two points a and b. Observe that a went through two topological events before reaching its current position, 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 forming link(i), plus the points forming the link of these, must be updated.

Chapter 5

GIS Operations for Trivariate Fields

This chapter describes several GIS and spatial analysis operations, for trivariate fields, which are based on the VD and/or the DT. Besides the obvious advantages that the indexing of a set of unconnected points brings (e.g. the closest-point queries are efficiently performed), many spatial analysis operations use the properties of the VD/DT to help the user have a better understanding of a dataset. While the VD (or the DT) is not mandatory for all the operations presented, it greatly simplifies and optimises many of them; in fact, a few of the operations would not even be possible without first constructing the VD/DT. The list of operations presented in this chapter is not exhaustive, as other operations based on the VD are possible, especially when other metrics are used. For instance, Okabe et al. (1994) present 35 GIS-related operations in two dimensions based on the generalised VD, operations that are theoretically possible in three dimensions.

I begin this chapter by discussing which GIS operations are useful when modelling trivariate fields, and by highlighting the major differences with bivariate fields. I then discuss a central problem of this thesis: the interpolation methods that can be used to reconstruct a field from the set of samples that were collected to study it. Some other operations are then presented, among them are some functions that help in visualising a dataset, and also an extension of the map algebra framework, where all the fields and operations are Voronoi-based.

5.1 A Classification of Three-dimensional GIS Operations

This section briefly discusses the GIS operations that are required in a GIS modelling three-dimensional geoscientific datasets. The classification is based on the one by Albrecht (1996), see Section 2.4.1 on page 22. Not all of Albrecht's GIS operations are relevant in the context of trivariate fields, and more are definitively needed, such as visualisation (Raper, 1992).

The updated classification would be as follows:

1. **Search**: interpolation, (re)classification, thematic and spatial search;
2. **Location Analysis**: buffer zone, overlay, and creation of 3D VD;
3. **Visualisation**: isosurface, slicing;
4. **Distribution/Neighbourhood**: cost, proximity, and nearest neighbour;
5. **Spatial Analysis**: statistical and pattern analysis, shape;
6. **Measurements**: distance, area, volume and centroid.

First, notice that the terrain analysis operations are not relevant in three dimensions, and have been removed from the list. A new category has been added: visualisation. As Houlding (1994) argues, "visualization is essentially a qualitative form of spatial analysis, although it can also be used to a limited extent in a quantitative sense, provided we have the means of tracking [...] or interrogating any point in terms of characteristic or variable values". I give more details about visualisation operations in Section 5.4.

All the search operations are clearly relevant in three dimensions, and they generalise easily. The interpolation methods are discussed in the next section, reclassification is part of map algebra (see Section 5.3), and spatial searches are performed efficiently when the VD is used (see Sections 4.2.4 and 4.2.5 about respectively the traversal and the point location problem).

The location analysis operations are also all needed in three dimensions. Although not used in this thesis, buffer zones are useful in geoscience, for example when different datasets are analysed together, and one wants to identify all the samples in one dataset that are within a certain distance of another object. Sprague et al. (2006) give a few examples with the system they developed to model geological data. Although overlay is mostly a concept related to object-based models stored in different datasets, objects can be extracted from fields and in a GIS the possibility of having different datasets is desirable. The topological relationships between objects (e.g. difference, union and intersection) are usually based on the framework of Egenhofer (1995).

The distribution/neighbourhood operations can also obviously be used in three dimensions, as can the measurement operations.

Finally, although not used or implemented in the context of this thesis, statistical methods can clearly help in the analysis of a datasets, and the same is true for shape analysis methods (e.g. the shape of an isosurface, see Section 5.4.1 below, could indicate some properties).

5.2 Spatial Interpolation[1]

Given a set of samples to which an attribute a is attached, spatial interpolation is the procedure used to estimate the value of the attribute at an unsampled location x. To achieve that, it creates a function f, called the interpolant, that tries to fit the samples as well as possible.

Interpolation methods are an essential operation in this thesis because they permit us to reconstruct a field from the set of samples that were collected to study it. They are also an important part in commercial GISs where they have been used for years to model, among other things, elevation data. They are crucial for the visualisation process (e.g. generation of contour lines), for the conversion of data from one format to another (e.g. from scattered points to raster), to have a better understanding of a dataset or simply to identify 'bad' samples. The result of interpolation—usually a surface that represents the real terrain—must be as accurate as possible because it often forms the basis for spatial analysis, for example runoff modelling or visibility analysis. Although interpolation, as implemented in commercial GISs, helps in creating three-dimensional surfaces, it is intrinsically a two-dimensional operation for only the $x - y$ coordinates of each sample are used and the elevation is considered an attribute.

[1]This section is based on two papers: Ledoux and Gold (2004a) and Ledoux and Gold (2005a).

To model the kind of datasets dealt with in this thesis, three-dimensional interpolation methods that consider the properties of the data must be used. While most of the interpolation methods used in GIS intuitively extend to three dimensions, it is not obvious that they preserve their properties or are appropriate for geoscientific datasets. I discuss in this section the generalisation to three dimensions of some of the interpolation methods found in GIS or geoscientific modelling packages. I restrict my discussion to methods that can be applied in a context of dynamic or interactive modelling (see Section 2.5 on page 25). For each method, I present and evaluate their properties, and also discuss implementation details to ensure that they are computationally efficient.

5.2.1 What Is a Good Interpolation Method?

Watson (1992), in his authoritative book, lists the essential properties of an 'ideal' interpolation method for bivariate geoscientific datasets. These properties can realistically be present for trivariate datasets, and are as follows:

1. **exact**: the interpolant must 'honour' the data points, or 'pass through' them.
2. **continuous**: a single and unique value must be obtained at each location. This is called a C^0 interpolant in mathematics.
3. **smooth**: it is desirable for some applications to have a function for which the first or second derivative is possible everywhere; such functions are respectively referred to as C^1 and C^2 interpolants.
4. **local**: the interpolation function uses only some neighbouring samples to estimate the value at a given location. This ensures that a sample with a gross error will not propagate its error to the whole interpolant.
5. **adaptability**: the function should give realistic results for anisotropic data distributions and/or for datasets where the data density varies greatly from one location to another.

In the context of dynamic/interactive modelling, I add to this list:

6. the method must be **computationally efficient**.
7. the method must require as little input as possible from the user, i.e. it should be **automatic** and not rely on user-defined parameters that require *a priori* knowledge of the dataset.

I describe in the following four interpolation methods, commonly found in GIS and geoscientific software, that respect some of the criteria of an ideal method for an interactive system. The major omission from my list is Kriging (Matheron, 1971; Oliver and Webster, 1990), which, although possessing many of the ideal properties and being popular in many application domains because it produces an interpolant that minimises the error variance at each location, cannot be realistically be used in a dynamic environment. Indeed, the Kriging estimation is based on a function characterising the dependence between the attributes of any two samples that are at a given distance from each other. This function is obtained by studying the variance of the attributes according to the distance and the direction, and fitting a simple function. This is done manually by the user, and is a somewhat time-consuming process that is unthinkable to do every time the dataset is modified.

The four interpolation methods discussed are all weighted-average methods, which are methods that use only some sample points, to which weights (importance) are assigned. The interpolation function f of such methods have the following form:

$$f(x) = \frac{\sum_{i=1}^{k} w_i(x)\, a_i}{\sum_{i=1}^{k} w_i(x)} \tag{5.1}$$

where $w_i(x)$ is the weight of each neighbour p_i (with respect to the interpolation location x) used in the interpolation process, and a_i the attribute of p_i.

5.2.2 Nearest Neighbour

Nearest neighbour is a very simple interpolation method: the value of an attribute at location x is simply assumed to be equal to the attribute of the nearest data point. Given a set S of data points in a three-dimensional space, if interpolation is performed with this method at many locations close to each other, the result is VD(S), where all the points inside a Voronoi cell have the same value. The VD actually creates a piecewise model, where the interpolation function inside each Voronoi cell is a constant function.

Although the method possesses many of the desirable properties (it is exact, local and can handle anisotropic data distributions), the reconstruction of continuous fields can not realistically be done using it since it fails lamentably in properties 2 and 3. The interpolation function is indeed discontinuous at the border of cells. It is nevertheless the perfect method for reconstructing discrete fields, and is also often used in remote sensing to avoid averaging or blurring the resulting image.

The implementation of the method sounds easy: simply find the closest data point and assign its value to the interpolation location. The difficulty lies in finding an efficient way to get the closest data point. The simplest way consists of measuring the distance for each of the n points in the dataset, but this is too slow for large datasets. To speed up this brute-force algorithm, auxiliary data structures that will spatially index the points must be used. They usually subdivide hierarchically the space into cells (usually squares or rectangles) and build a tree that uses $\mathcal{O}(n)$ space; examples of such trees are the KD-tree, the R-tree and the octree (see van Oosterom (1999) for a survey of two-dimensional structures; the three-dimensional methods are simple extensions). Thus, to find the nearest neighbour, it suffices to navigate in the tree and simply test the points in the adjacent cells. Locating the cell containing a test point can for example be done efficiently with a KD-tree (in $\mathcal{O}(\log n)$ time). An obvious solution in the context of this thesis consists of building the VD for the set of points and identifying inside which cell the interpolation point lies; that can be done with a trivial modification of the algorithm Walk as shown on page 55.

5.2.3 Distance-based Methods

Distance-based methods are probably the most known methods and they are widely used in many fields. As shown in Figure 5.1(a), in two dimensions they often use a 'searching circle', whose radius is user-defined, to select the data points p_i involved in the interpolation at location x. The weight assigned to each is typically inversely proportional to the square of the distance from x to p_i. Other weights can also be used, see Watson (1992) for a discussion. The size

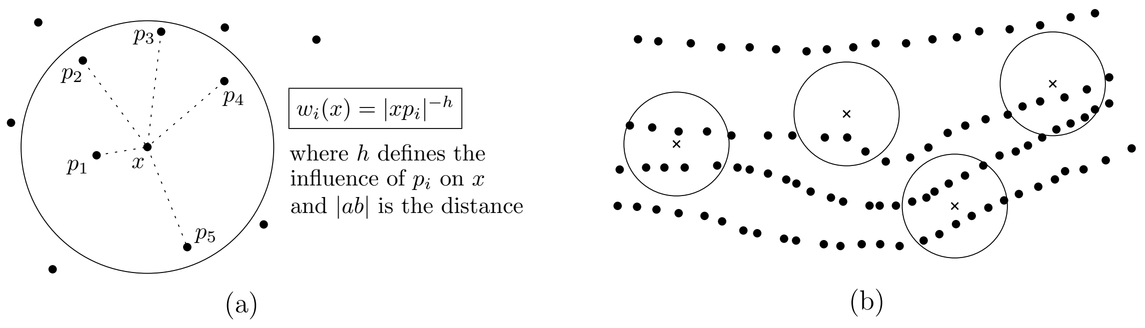


Figure 5.1: (a) Inverse distance to a power interpolation. **(b)** Problems with anisotropic datasets.

of the radius of the searching circle influences greatly the result of the interpolation: a very big radius means that the resulting surface will be smooth or 'flattened'; on the other hand, a radius that is too small might have dramatic consequences if for example no data points are inside the circle. A good knowledge of the dataset is thus required to select this parameter.

This method has many flaws when the data distribution is highly anisotropic or varies greatly in one dataset because a fixed-radius circle will not necessarily be appropriate everywhere in the dataset. Figure 5.1(b) shows one example where one circle, when used with a dataset extracted from contour lines, clearly gives erroneous results at some locations. The major problem with the method comes from the fact that the criterion, for both selecting data points and assigning them a weight, is one-dimensional and therefore does not take into account the spatial distribution of the data points close to the interpolation location. A criterion based on areas and volumes for respectively bivariate and trivariate data yields better results, as the next two sections demonstrate.

The generalisation of this method to three dimensions is straightforward: a searching sphere with a given radius is used. The same problems with the one-dimensionality of the method will be even worse because the search must be performed in one more dimension. The method has too many problems to be considered for geoscientific datasets: the interpolant is not guaranteed to be continuous, especially when the dataset has an anisotropic distribution, and the criterion has to be selected carefully by the user.

Note that the implementation problems are also similar to the ones encountered with the previous method, and an auxiliary data structure must be used to avoid testing all the points in a dataset.

5.2.4 Linear Interpolation in Simplices

This method is popular in two dimensions for terrain modelling applications and is based on a triangulation of the data points. As is the case for the VD, a triangulation is a piecewise subdivision of the space covered by the data points, and in the context of interpolation a linear function is assigned to each piece. Interpolating at location x means first finding inside which triangle x lies, and then the height is estimated by linear interpolation on the plane defined by the three vertices forming the triangle. This can be efficiently implemented by using barycentric coordinates, which are local coordinates defined within a triangle (see Figure 5.2 for details). To obtain satisfactory results, this method is usually used in two dimensions with Delaunay

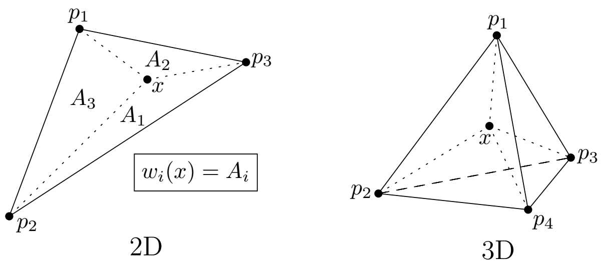


Figure 5.2: Barycentric coordinates in two and three dimensions. A_i represents the area of the triangle formed by x and one edge.

triangles because they have a 'good shape'. This ensures that the three vertices used in the interpolation process will most likely be close to and around the interpolation location.

The generalisation of this method to three dimensions is as follows: linear interpolation is performed within each tetrahedron of a tetrahedralization. Note that the barycentric coordinates can be generalised to linearly interpolate inside a tetrahedron.

As explained in Section 3.2.2, finding tetrahedra having a good shape is not as easy as in two dimensions, and the presence of slivers yield bad results for the interpolation process. This is why different empirical methods have been developed to improve the shape of certain tetrahedra in a DT. For instance, Cheng et al. (2000) show that a tetrahedralization where all the tetrahedra are well-shaped can be obtained as follows: start with the DT, assign some importance to some data points forming slivers, and modify locally some tetrahedra (with the help of flips) to obtain a *regular triangulation* of the set of points (Edelsbrunner and Shah, 1996). Contrary to some other methods in engineering where the shape of the elements are improved by inserting new points (see Shewchuk (1997b)), Cheng et al.'s method eliminates slivers without adding any new point and without moving any.

The construction of the DT and the point location problem have both been covered previously. The linear interpolation itself is assumed to be performed in constant time since only four data points are involved. When used on a tetrahedralization having well-shaped tetrahedra, the linear interpolation method possesses all the ingredients of an ideal method but one: the first (or second) derivative of the resulting function is not possible at the vertices, edges and faces of the triangulation. Akima (1978) solved this problem in two dimensions by using higher order functions in each piece of the triangulation, and the same could be done in three dimensions.

5.2.5 Natural Neighbour Interpolation

It has been shown that *natural neighbour interpolation* (Sibson, 1980, 1981) avoids most of the problems of conventional methods and performs well for irregularly distributed data (Gold, 1989; Sambridge et al., 1995; Watson and Phillip, 1987). This is a weighted-average technique based on the VD for both selecting the set of neighbours of the interpolation point x and for determining the weight of each. The neighbours used in an estimation are selected using the adjacency relationships of the VD, which results in the selection of neighbours that both surround and are close to x. The weight of each neighbour is based on the volume (for convenience, in this section, 'volume' is used to define area in 2D, volume in 3D and hyper-volume in higher dimensions) that the Voronoi cell of x 'steals' from the Voronoi cell of the neighbours in the absence of x.

Although the concepts behind natural neighbour interpolation are simple and easy to understand, its implementation is far from being straightforward, especially in higher dimensions. The main reasons are that the method requires the computation of two VDs—one with and one without the interpolation point—and also the computation of volumes of Voronoi cells. This involves algorithms for both constructing a VD and deleting a point from it. By comparison, conventional interpolation methods are relatively easy to implement; this is probably why they can be found in most GIS and geoscientific modelling packages. Surprisingly, although many authors present the properties and advantages of the method, few discuss details concerning its implementation.

I present in the following a simple natural neighbour interpolation algorithm valid in any dimensions. The algorithm works directly on the DT and uses the concept of flipping for inserting and deleting the interpolation point. The Voronoi cells are extracted from the DT and their volumes are calculated by decomposing them. I believe the algorithm to be considerably simpler to implement than other known methods, as only an incremental insertion algorithm based on flips, with some minor modifications, are needed.

5.2.5.1 Related Work

Very few researchers discuss implementation details of the natural neighbour interpolation probably because only the two-dimensional case is usually considered and the implementation of the method in that space is relatively easy. Efficient algorithms for manipulating a two-dimensional VD/DT exist and are rather easy to implement, and computing the area of convex polygons is trivial. As an alternative to inserting and deleting the interpolation point x, Watson (1992) presents an algorithm that mimics the insertion of the interpolation point x. The idea is to extract the Voronoi vertices of the VDs with and without x, order them around x and find the stolen areas by decomposing them into triangles.

In three and higher dimensions, things get more complicated because, as seen in Chapter 4, the algorithms to manipulate a VD are difficult to implement in the presence of degenerate cases. Sambridge et al. (1995) describe three-dimensional methods to compute a VD, insert a new point x in it and compute volumes of Voronoi polyhedra, but they do not explain how x is deleted at the end. Owen (1992) also proposes a sub-optimal solution which consists of inserting x in the DT with the Bowyer-Watson's algorithm and computing the stolen volumes by calculating the volume of the Voronoi cells with and without x by intersecting three-dimensional planes (which is computationally expensive). Instead of using a deletion algorithm, the tetrahedra that were removed by the insertion of x are saved and sewn back to the original triangulation. Also, the idea of mimicking the insertion (Watson, 1992) has been generalised to three dimensions by Boissonnat and Cazals (2002) and to arbitrary dimensions by Watson (2001). To calculate the stolen volumes in one operation, both algorithms use somewhat complicated methods to "order" the Voronoi vertices of the two VDs and then decompose the volumes into simplices. The time complexity of these two algorithms is the same as the one to insert one point in a VD/DT. Note that this method also requires the use of an auxiliary data structure to store the tetrahedra that would be removed by the insertion of x.

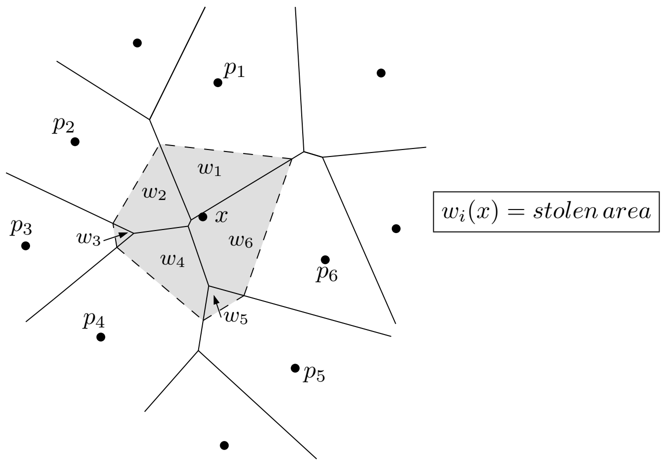


Figure 5.3: Natural neighbour coordinates of x in the plane.

5.2.5.2 Natural Neighbours

The idea of natural neighbour is closely related to the concept of Voronoi diagram. Let VD(S) be the VD of a set S of points in $\mathbb{R}^d$, and let p be a point in S. The natural neighbours of p are the points in $q \in S$ whose Voronoi cell is contiguous to $\mathcal{V}_p$. For example, in Figure 3.8(a) on page 38, p has seven natural neighbours (the points generating the light grey cells).

Natural Neighbour Coordinates. The concept of natural neighbours can also be applied to a point x that is not present in S. In that case, the natural neighbours of x are the points in S whose Voronoi cell would be modified if the point x were inserted in VD(S). The insertion of x creates a new Voronoi cell $\mathcal{V}_x$ that 'steals' volume from the Voronoi cells of its 'would be' natural neighbours, as shown in Figure 5.3. This idea forms the basis of natural neighbour coordinates (Sibson, 1980, 1981), which define quantitatively the amount $\mathcal{V}_x$ steals from each of its natural neighbours. Let $\mathcal{D}$ be a VD(S), and $\mathcal{D}^x = \mathcal{D} \cup \{x\}$. The Voronoi cell of a point p in $\mathcal{D}$ is defined by $\mathcal{V}_p$, and $\mathcal{V}_p^x$ is its cell in $\mathcal{D}^x$. The natural neighbour coordinate of x with respect to a point p_i is

$$w_i(x) = \frac{Vol(\mathcal{V}_{p_i} \cap \mathcal{V}_x^x)}{Vol(\mathcal{V}_x^x)} \tag{5.2}$$

where $Vol(\mathcal{V}_{p_i})$ represents the volume of $\mathcal{V}_{p_i}$. For any x, the value of $w_i(x)$ will always be between 0 and 1: 0 when p_i is not a natural neighbour of x, and 1 when x is exactly at the same location as p_i. A further important consideration is that the sum of the volumes stolen from each of the k natural neighbours is equal to $Vol(V_x^x)$, in other words:

$$\sum_{i=1}^{k} w_i(x) = 1. \tag{5.3}$$

Therefore, the higher the value of $w_i(x)$ is, the stronger is the 'influence' of p_i on x. The natural neighbour coordinates are influenced by both the distance from x to p_i and the spatial distribution of the p_i around x.

Natural Neighbour Interpolation. Based on the natural neighbour coordinates, Robin Sibson developed a weighted average interpolation technique that he named *natural neighbour interpolation* (Sibson, 1980, 1981); note that the method is also sometimes called Sibson's method. The points used to estimate the value of an attribute at location x are the natural neighbours of x, and the weight of each neighbour is equal to the natural neighbour coordinate of x with respect to this neighbour. If we consider that each data point in S has an attribute a_i (a scalar value), the natural neighbour interpolation function, which is also called the interpolant, is

$$f(x) = \sum_{i=1}^{k} w_i(x)\, a_i \tag{5.4}$$

where $f(x)$ is the interpolated function value at the location x.

The natural neighbour interpolant possesses all the wished properties enumerated earlier, except that the first derivative is undefined at the data points. To obtain a continuous interpolant even at the data points, Sibson uses the weights defined in Equation 5.2 in a quadratic equation where the gradient at x is considered. To my knowledge, this method has not been used with success with real data and therefore I do not use it. Other ways to remove the discontinuities at the data points have been proposed: Watson (1992) describes different methods to estimate the gradient at x and how to incorporate it in Equation 5.4; and Gold (1989) proposes to modify the weight of each p_i with a simple hermitian polynomial so that, as x approaches p_i, the derivative of $f(x)$ approaches 0. Modifying Equation 5.4 to obtain a continuous function can yield very good results in some cases, but with some datasets the resulting surface can contain unwanted effects. Different datasets require different methods and parameters, and, for this reason, modifications should be applied with great care. Notice that these methods first require to compute the natural neighbour interpolant as described in Equation 5.4, and the algorithm described below can be easily modified to take into account the first derivative.

5.2.5.3 Flips and Construction of the DT in $\mathbb{R}^d$

Because the algorithm presented in this section is valid in any dimensions, the flipping and insertion operations in d-dimensional space need to be defined. They are both simple extensions of their three-dimensional counterparts.

Consider the set S of $d+2$ points in $\mathbb{R}^d$. Similarly to the case in $\mathbb{R}^3$ (see Figure 4.3), a flip is the operation that retriangulates conv(S) by removing m d-simplices and replacing them by $d+2-m$ different d-simplices. Lawson (1986) shows that $d+2$ points in $\mathbb{R}^d$ can always be triangulated in at most two different ways; thus, the number of flips needed to modify a triangulation does not increase with the dimension of the space. Also, note that only one of the two possible triangulations of S respects the Delaunay criterion (empty circumball).

The insertion of a single point p in a DT in $\mathbb{R}^d$ using bistellar flips has also been generalised (Edelsbrunner and Shah, 1996). The algorithm is basically the same as in three dimensions: sometimes some flips will not be possible, but there will always exist another one such that p can be inserted. The algorithm used is therefore based on incremental insertions.

5.2.5.4 A Flip-based Natural Neighbour Interpolation Algorithm

The natural neighbour interpolation algorithm I propose, InterpolNN as detailed in Algorithm 5.1, performs all the operations directly on the DT (with flips) and the Voronoi cells are extracted when needed. It is based on a very simple idea: insert the interpolation point x in the DT, calculate the volume of the Voronoi cell of each natural neighbour of x, then remove x and recalculate the volumes to obtain the stolen volumes. To speed up the algorithm, two modifications are applied. The first one concerns the deletion of x from the DT. Based on Lemmas 4.4 and 4.2, we can affirm that if x was added to the triangulation with a sequence l of flips, simply performing the inverse flips of l in reverse order will delete x. During the insertion of x, the sequence of flips is therefore stored, to be 'rolled back' later. The second modification concerns how the overlap between Voronoi cells with and without the presence of x is calculated. I show that only some facets of a Voronoi cell are needed to obtain the overlapping volume.

Algorithm 5.1: InterpolNN($\mathcal{T}$, x)

```
   Input: A Delaunay triangulation $\mathcal{T}$ in $\mathbb{R}^d$, and the interpolation location $x$
   Output: An estimate of the value of the attribute at location $x$
 1 insert $x$ in $\mathcal{T}$ with $d$-dimensional version of InsertOnePoint
   // as flips are performed, store them in a simple list $L$
 2 foreach $p_i$ of $x$ do
 3     compute vol($\mathcal{V}^x_{p_i}$) and store it
 4 end
 5 compute vol($\mathcal{V}^x_x$) and store it
 6 delete $x$ from $\mathcal{T}^x$ // rollback the sequence used for insertion
 7 foreach $p_i$ of $x$ in $\mathcal{T}^x$ do
 8     compute vol($\mathcal{V}_{p_i}$)
 9     compute $w_i(x)$ // Equation 5.2
10 end
11 Return value obtained from Equation 5.4
```

Rollback of the Flips. Because a flip affects only a predefined number of simplices, it is possible to identify uniquely a flip with a $(d-2)$-simplex. To remember the sequence of flips, a simple list containing $(d-2)$-simplices is stored. That means that in two dimensions, the list contains the p_i in the order that they became natural neighbour during the insertion. In three dimensions, only a flip23 adds a new p_i to x; a flip32 only changes the configuration of some tetrahedra. In the case of a flip23, the storage of the edge xp_i that is created by the flip is sufficient. A flip32 deletes a tetrahedron and modifies the configuration of the two others such that, after the flip, they are both incident to x and share a common face abx; the edge ab of this face is stored. Therefore, in two dimensions, to delete x we take one p_i, find the two triangles incident to the edge xp_i and perform a flip22. When x has only three natural neighbours, a flip31 deletes x completely from the triangulation. In three dimensions, if the current edge is xp_i, a flip32 is used on the three tetrahedra incident to xp_i; and if ab is the current edge, then a flip23 is performed on the two tetrahedra sharing the face abx.

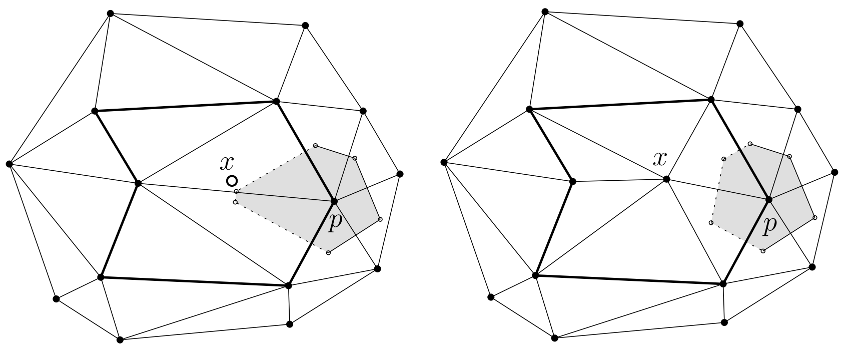


Figure 5.4: The insertion of x in a DT in $\mathbb{R}^d$ will modify only certain facets of $\mathcal{V}_p$ (p being a natural neighbour of x). The modified facets (edges in two dimensions) of $\mathcal{V}_p$ are dotted.

Observe that this method is not affected by degenerate cases. If the insertion algorithm used is robust against all special cases, then rolling back the flips is guaranteed to work since each $(d-2)$-simplex identifies uniquely a flip.

Volume of a Voronoi cell. The volume of a d-dimensional Voronoi cell is computed by decomposing it into d-simplices—not necessarily Delaunay simplices—and summing their volumes. The volume of a d-simplex σ is easily computed:

$$vol(\sigma) = \frac{1}{d!} \left| det \begin{pmatrix} v^0 & \dots & v^d \\ 1 & \dots & 1 \end{pmatrix} \right| \tag{5.5}$$

where v^i is a d-dimensional vector representing the coordinates of a vertex and det is the determinant of the matrix. Triangulating a Voronoi cell is easily performed since it is a convex polytope.

In order to implement InterpolNN, we do not need to know the volume of the Voronoi cells of the p_i in $\mathcal{T}$ and $\mathcal{T}^x$, but only the *difference* between the two volumes. By looking at Figure 5.4, it is easy to see that some parts of a Voronoi cell will not be affected by the insertion of x in $\mathcal{T}$, and computing them twice to subtract afterwards is computationally expensive and useless.

When x is inserted in a DT in $\mathbb{R}^d$, only a certain set of conflicting d-simplices are deleted and replaced by new ones; let the union of these d-simplices be the star-shaped but not necessarily convex polytope P. Only the Voronoi facets dual to the Delaunay edges inside or on the boundary of P will be modified when x is either inserted or deleted. This is because these edges are the only ones having different d-simplices incident to them in $\mathcal{T}$ and $\mathcal{T}^x$. Therefore, to optimise this step of the algorithm (lines 3 and 8), only these Voronoi facets are processed. The 'volume' of a single Voronoi facet is computed by decomposing it into simplices formed by x and the Voronoi vertices forming the facet. In two dimensions that means one triangle for each Voronoi facet (an edge), and in three dimensions the Voronoi facets (a convex face) can be for instance first fan-shaped triangulated, and then tetrahedra are formed by each resulting triangle on the face and x.

Finally, observe that for InterpolNN to be robust, the set of points must be contained in a $(d+1)$-simplex big enough to contain all the points (as in Section 4.2.1), otherwise the Voronoi cells of the points on the boundary of S will be unbounded and have infinite volumes.

5.2.5.5 Theoretical Performances

As already explained in Section 4.3.2 for the 3D case (but it is the same idea in higher dimensions), by using a flipping algorithm to insert x in a d-dimensional DT $\mathcal{T}$, each flip performed removes one and only one conflicting d-simplex from $\mathcal{T}$. Hence, if s d-simplices are conflicting with x, then exactly s flips are needed to obtain $\mathcal{T}^x$.

Each two-dimensional flip adds a new natural neighbour to x. The number of flips needed to insert x is therefore proportional to the degree of x (the number of incident edges) after its insertion. Without any assumptions on the distribution of the data, the average degree of a vertex in a 2D DT is 6; which means an average of four flips are needed to insert x (one flip13 plus three flip22). This is not the case in three dimensions (a flip32 does not add a new natural neighbour to x) and it is therefore more complicated to give a value to s. It can nevertheless be affirmed that the value of s will be somewhere between the number of edges and the number of tetrahedra incident to x in $\mathcal{T}^x$; these two values are respectively around 15.5 and 27.1 when points are distributed according to a Poisson distribution (Okabe et al., 2000). As a result, if x conflicts with s simplices in $\mathcal{T}$ then $\mathcal{O}(s)$ time is needed to insert it.

Deleting x from $\mathcal{T}^x$ also requires s flips; but this step is done even faster than the insertion because operations to test if a simplex is Delaunay are not needed, nor are tests to determine what flip to perform. Also, the volume of each Voronoi cell is computed only partly, and this operation is assumed to be done in constant time. In InterpolNN, if k is the degree of x in a d-dimensional DT, then the volume of k Voronoi cells must be partly computed twice: with and without x in $\mathcal{T}$.

As a conclusion, InterpolNN has a time complexity of $\mathcal{O}(s)$, which is the same as an algorithm to insert a single point in a Delaunay triangulation. However, the algorithm is obviously slower by a certain factor since x must be deleted and parts of the volumes of the Voronoi cells of its natural neighbours must be computed.

5.2.6 Discussion

This section has demonstrated that the weighted-average methods commonly used in GISs do generalise to three dimensions, but sometimes new problems appear, for example with the shape of tetrahedra or with the computational efficiency. Also, the flaws present in some two-dimensional methods will often be amplified in three and higher dimensions. Natural neighbour interpolation seems to be the method that is the most appropriate for modelling geoscientific dataset because it is robust, entirely objective, and gives good results when the data distribution is anisotropic.

5.3 A Voronoi-based Map Algebra[2]

Because the unnatural discretisation of space needed to use the map algebra framework (Tomlin, 1983) has been severely criticised (see Section 2.4.2 on page 23), many have proposed improvements. For example, Takeyama (1996) proposes Geo-Algebra, a mathematical formalisation and generalisation of map algebra that integrates the concepts of *templates* and cellular

[2]This section is based on Ledoux and Gold (2006a).

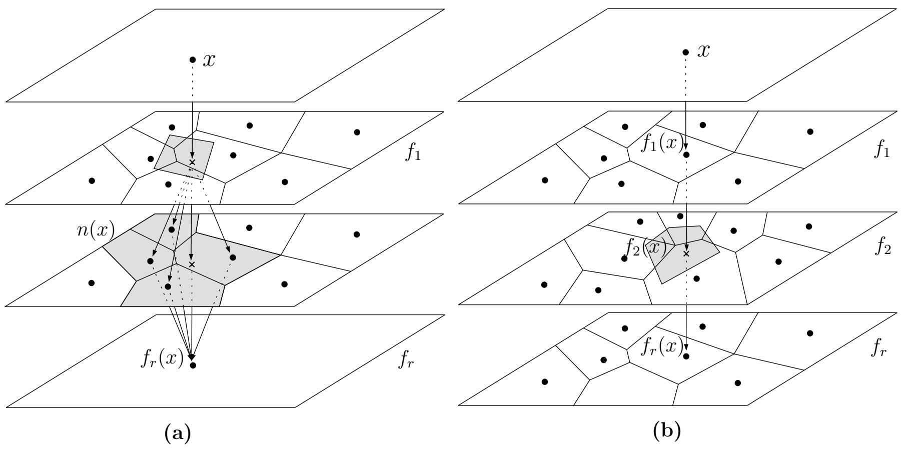


Figure 5.5: Two Voronoi-based map algebra operations. The top layer represents the spatial extent of the fields, and x is a location for which the value in the resulting field f_r (bottom layer) is sought. **(a)** A unary focal operation performed on the field f_1. The third layer represents the neighbourhood function $n(x)$. **(b)** A binary local operation performed on the fields f_1 and f_2.

automata under the same framework. The templates, developed for *image algebra* (Ritter et al., 1990), extends the concept of neighbourhood of a location, and the addition of cellular automata permits us to model geographic processes. Pullar (2001) also uses the idea of templates and shows how they can help to solve several practical GIS-related problems. But the main criticism is related to the fact that map algebra was developed for fields stored as raster files. Recognising that, Kemp (1993) shows that alternative representations (e.g. TINs and contour lines) are viable solutions. She proposes to have operations—similar to map algebra's—for modelling fields, which are not all stored under the same representation. She therefore defines a set of rules to convert the different types of fields to other ones when binary operations are applied. For example, if two fields, one stored as a TIN and the other as contour lines, are analysed then the contours must first be converted to TIN before any manipulation is done. Haklay (2004), also to avoid the drawbacks of raster structures, proposes a system where only the data points (samples) and the spatial interpolation function used to reconstruct the field are stored. Each field is thus defined mathematically, which allows the manipulation of different fields in a formulaic form.

This section presents a variant of map algebra where every field and every operation is based on the VD. This eliminates the need to first convert to grids all the datasets involved in an operation (and moreover to grids that have the same orientation and resolution), as the VD can be used directly to reconstruct the fields.

5.3.1 Unary Operations

When a field is represented by the VD, unary operations are simple and robust. Indeed, as shown in the previous section, discrete fields can be elegantly represented directly with the VD where a constant function is assigned to each cell, and continuous fields can be efficiently reconstructed with natural neighbour interpolation. Also, the neighbouring function needed for focal operations is simply the natural neighbours of every location x, as defined in the previous section. Figure 5.5(a) shows a focal operation performed on a field f_1. Since at location x there are no samples, a data point is temporarily inserted in the VD to extract the natural neighbours of x (the generators of the shaded cells). The result, $f_r(x)$, is for example the average of the values of the samples; notice that the value at location x is involved in the process and can be obtained easily with natural neighbour interpolation.

5.3.2 Binary Operations

Although Kemp (1993) claims that "in order to manipulate two fields simultaneously (as in addition or multiplication), the locations for which there are simple finite numbers representing the value of the field must correspond", I argue that there is no need for two VDs to correspond in order to perform a binary operation because the value at any locations can be obtained readily with interpolation functions. Moreover, since the VD is rotationally invariant (like a vector map), we are relieved from the burden of resampling datasets to be able to perform operations on them.

When performing a binary operation, if the two VDs do not correspond—and in practice they rarely will do!—the trickiest part is to decide where the 'output' data points will be located. Let two fields f_1 and f_2 be involved in one operation, then several options are possible. First, the output data points can be located at the sampled locations of f_1, or f_2, or even both. An example where the output data points have the same locations as the samples in f_1 is shown in Figure 5.5(b). Since there are no samples at location x in f_2, the value is estimated with natural neighbour interpolation. The result, $f_r(x)$, could for example be the average of the two values $f_1(x)$ and $f_2(x)$. It is also possible to randomly generate a 'normal' distribution of data points in space (e.g. a Poisson distribution) and to output these. But one should keep in mind that in many applications the samples can be meaningful, and it is therefore recommended to always keep the original samples and if needed to densify them by randomly adding some data points. The VD also permits us to vary the distribution of data points across space, for example having more data points where the terrain is rugged, and less for flat areas.

5.3.3 Zonal Operations

Since the local and focal operations are rather straightforward when the fields are stored with the VD, the only operation left to discuss is the zonal operation. As with the other map algebra operations, a zonal operation must also output a field because its result might be used subsequently as the input in another operation. With a Voronoi-based map algebra, the output has to be a VD, and the major difficulty in this case it that we must find a VD that conforms to (or approximates) the set of zones. Since zones come from many sources, different cases will arise. The first two-dimensional example is a remote sensing image that was classified into several groups (e.g. land use). Such a dataset can easily be converted to a VD: simply construct the VD of the centre of every pixel. Although this results in a huge VD, it can easily be simplified by deleting all data points whose natural neighbours have the same value. Notice in Figure 5.6 that the deletion of a single point is a local operation, and that the adjacent cells will simply merge and fill up the space taken by the original cell. The second example is with *in situ* data, for instance in oceanography a dataset indicating the presence (or not) of fish in a water body. The VD of such a dataset can obviously be used directly. The third example is a set of arbitrary zones, such as a vector map of Europe. In two dimensions, it is possible to approximate the zones with a VD (Suzuki and Iri, 1986), but the algorithm is

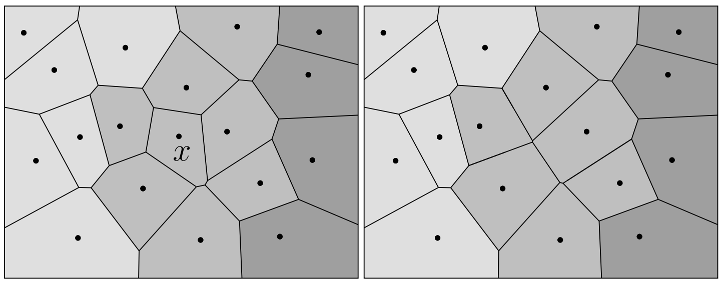


Figure 5.6: Simplification of a discrete field represented with the VD. The natural neighbours of the data point x all have the same attribute as x (here the value is defined by the colour), and deleting it does not change the field.

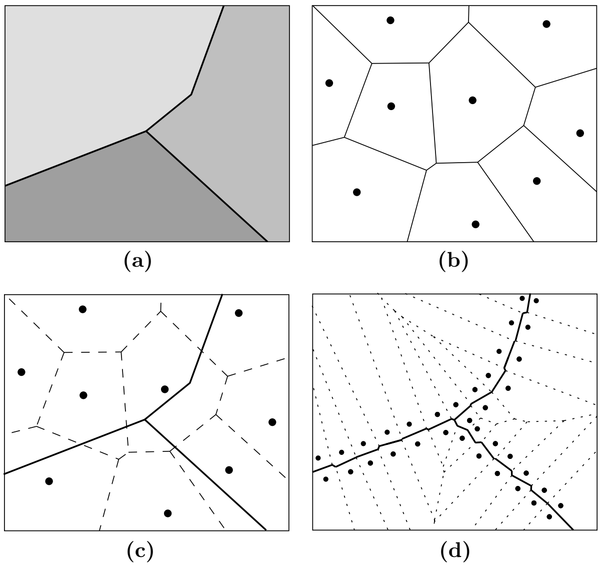


Figure 5.7: Zonal operations with a Voronoi-based map algebra. **(a)** A vector map of three zones. **(b)** A continuous field represented with the VD. **(c)** When overlaid, notice many Voronoi cells overlap the zones. **(d)** Approximation of the borders of the zones with the VD.

complex and the results are not always satisfactory. A simpler option is to define a set of “fringe” points on each side of a line segment, and label each point with the value associated to the zone. Gold et al. (1996) show that the boundaries can be reconstructed/approximated automatically with Voronoi edges. An example is shown in Figure 5.7: a set of three zones appears in Figure 5.7(a), and in Figure 5.7(d) the Voronoi edges for which the values on the left and right are different are used to approximate the boundaries of the zones. Since each location x in the output field of a zonal operation summarises the values of a field in a given zone, we must make sure that the locations used for the operation are sufficient and distributed all over the zone. Let us go back to the example of the temperature across Europe to find the average in each country. Figure 5.7(a) shows a vector map with three countries, and the temperature field f_1 is represented by a VD in Figure 5.7(b). Notice that when the two datasets are overlaid (Figure 5.7(c)), many Voronoi cells cover more than one zone. Thus, simply using the original samples (with a point-in-polygon operation) will clearly yield inaccurate results. The output field f_r, which would contain the average temperature for each country, must be a VD, and

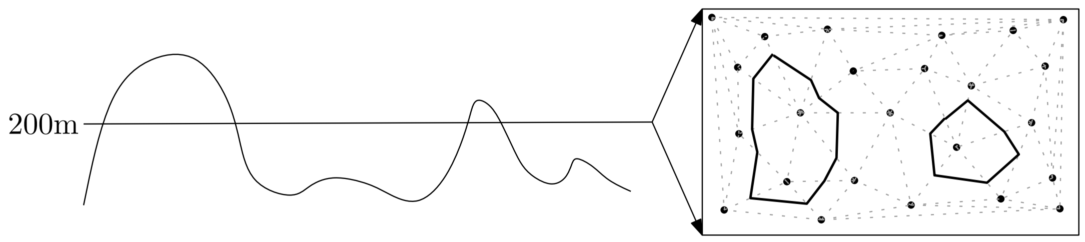


Figure 5.8: Cross-section of a terrain (left), and the 200m isoline extracted from a TIN (right).

it can be created with the fringe method (Figure 5.7(d)). Because the value assigned to each data point correspond to the temperature for the whole zone, I suggest estimating, with the natural neighbour interpolation, the value at many randomly distributed locations all over each zone.

5.4 Visualisation Tools

It is notoriously difficult to visualise trivariate fields, even if a three-dimensional computer environment offering translation, rotation and zoom is available. The major problem is that unlike bivariate fields, where the attribute a can be treated as another dimension, we can not 'lift' every location x_i by its value a_i to create a surface in one more dimension and visualise/analyse it—we can not see in four dimensions!

We therefore have to resort to other techniques to be able to visualise the spatial variation of trivariate fields and to extract meaningful information from them. Within the field of *volume visualisation* (Kaufman, 1996), there are mainly three approaches: (1) direct volume rendering; (2) extraction of isosurfaces; (3) slicing.

The first approach permits us to visualise the whole dataset at once by projecting the information onto a single plane (the screen). The mapping is done by assigning at every location in the field a colour and an *opacity*, and then creating a single image where the colour of a given pixel is obtained by accumulating the colour attributes of the locations that project to this pixel. This approach yields good results, but is also complex, the rendered image takes long to compute, and algorithms are usually based on voxel data. Hence, since the VD/DT is not relevant to this technique, only the approaches of isosurfaces and slicing are discussed in this section.

5.4.1 Extraction of Isosurfaces

Given a trivariate field $f(x, y, z) = a$, an isosurface is the set of points in space where $f(x, y, z) = a_0$, where a_0 is a constant. Isosurfaces, also called *level sets*, are the three-dimensional analogous concept to isolines (also called contour lines), which have been traditionally used to represent the elevation in topographic maps. Of course, in practice, isolines and isosurfaces are only approximated from the computer representation of a field.

In two dimensions, isolines are usually extracted directly from a TIN or a regular grid. As shown in Figure 5.8, the idea is to compute the intersection between the level value (e.g. 200m)

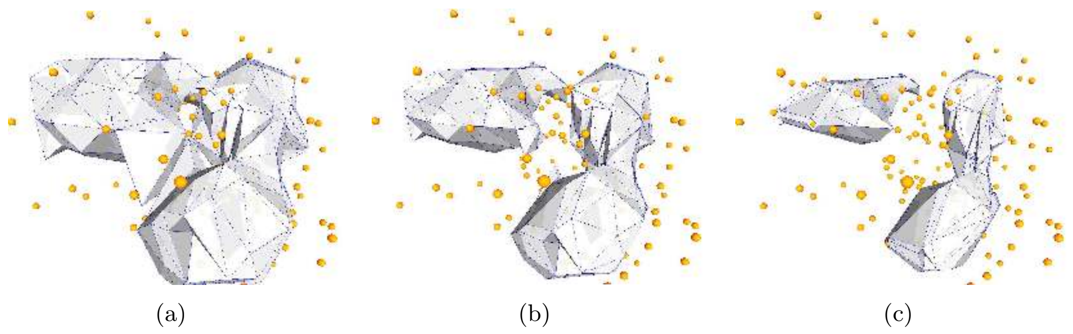


Figure 5.9: An example of an oceanographic dataset where each point has the temperature of the water, and three isosurface extracted (for a value of respectively 2.0, 2.5 and 3.5) from this dataset.

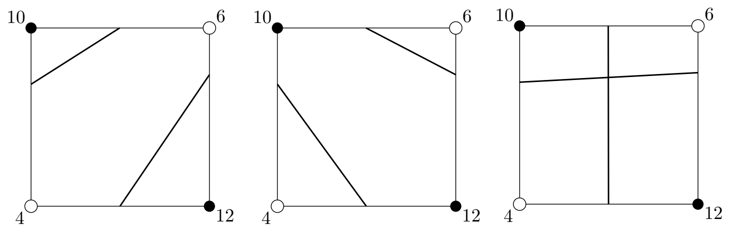


Figure 5.10: Ambiguous extraction of an isoline where the attribute is 8.

and the terrain, represented for instance with a TIN. Each triangle is scanned and segment lines are extracted to form an approximation of an isoline.

Marching Cubes. The same idea can be used in three dimensions to extract surfaces from a field (see three examples in Figure 5.9).

The principal and most known algorithm for extracting an isosurface is the *Marching Cubes* (Lorensen and Cline, 1987). The isosurface is computed by finding the intersections between the isosurface and each cell of the representation of a field, which is assumed to be voxel. Linear interpolation is used along the edges of each cube of a voxel representation to extract 'polygonal patches' of the isosurface. There exist 256 different cases for the intersection with a cube (considering that the value of each of the eight vertices of a cube is 'above' of 'under' the threshold), although if we consider the symmetry in a cube that comes down to only 15 cases. The major problem with Marching Cubes is that the isosurface may contain 'holes' or 'cracks' when a cube is formed by certain configurations of above and under vertices. The ambiguities are shown in Figure 5.10 for the two-dimensional case when two vertices are above the threshold, and two under, and they form a 'saddle'. The three-dimensional case is similar, with many more cases possible.

Marching Tetrahedra. Although Wilhems and van Gelder (1990) describe various methods to fix the ambiguities, the simplest solution is to subdivide the cubes into tetrahedra. The so-called *Marching Tetrahedra* algorithm is very simple: each tetrahedron is tested for the intersection with the isosurface, and triangular faces are extracted from the tetrahedra by linear interpolation on the edges. The resulting isosurface is guaranteed to be topologically consistent (i.e. will not contain holes), except at the border of the dataset. But again, if a

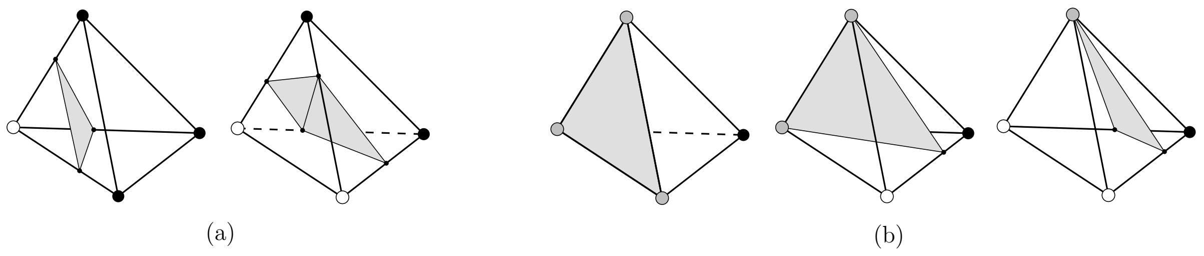


Figure 5.11: **(a)** Two cases for the intersection of an isosurface and a tetrahedron: either 3 or 4 intersections. **(b)** Three cases when some vertices (the grey ones) have exactly the same value as the isosurface.

big tetrahedron is used where the vertices are assigned to a value lower than the minimum value of the field, then all the isosurfaces extracted are guaranteed to be 'watertight'. The nice thing about the algorithm is that only three cases for the intersection of the isosurface and a tetrahedron can arise:

1. the four vertices have a higher (or lower) value. No intersection.
2. one vertex has a higher (or lower) value, hence the three others have a lower (or higher) value. Three intersections are thus defined, and a triangular face is extracted. See Figure 5.11(a) on the left.
3. two vertices have a higher (or lower) value and the others have a lower (or higher) value. Four intersections are thus defined. To ensure that triangular faces are extracted (better output for graphics cards), the polygon can be split into two triangles, with an arbitrary diagonal. See Figure 5.11(a) on the right.

The only degenerate cases possible are when one or more vertices have exactly the same value as the isosurface. These cases are handled very easily, and the intersection is simply assumed to be at the vertices themselves (see Figure 5.11(b)). Notice that the case when three vertices have exactly the same value, then the complete face of the tetrahedron must be extracted to ensure topological consistency.

Speeding up the algorithm. The major drawback of Marching Tetrahedra is that every tetrahedron used to represent the field must be visited to extract one or more isosurfaces, even if the isosurface actually intersects only a few. Different methods have therefore been proposed to speed up the algorithm. A typical one is by Cignoni et al. (1996) who propose a theoretically fast algorithm where one isosurface is extracted in $\mathcal{O}(\log s + k)$, where s is the number of tetrahedra representing the field and k the size of the output. To achieve that, the maximum and minimum values of each tetrahedron is store in an *interval tree* that permits the efficient retrieval of all intervals containing a given value. The major disadvantage of the technique is that a large run-time data structure of $\Theta(s)$ must be built, in addition to the data structure used for the tetrahedralization.

It is possible to reduce the size of the additional data structure by observing that an isosurface is a surface without holes formed of triangular faces, and that starting from one tetrahedron intersecting the surface, it is possible to use the adjacency relationships in the tetrahedralization to extract the whole surface. This also permits us to build a surface where the topological relationships between the triangular patches are known, unlike the methods previously mentioned.

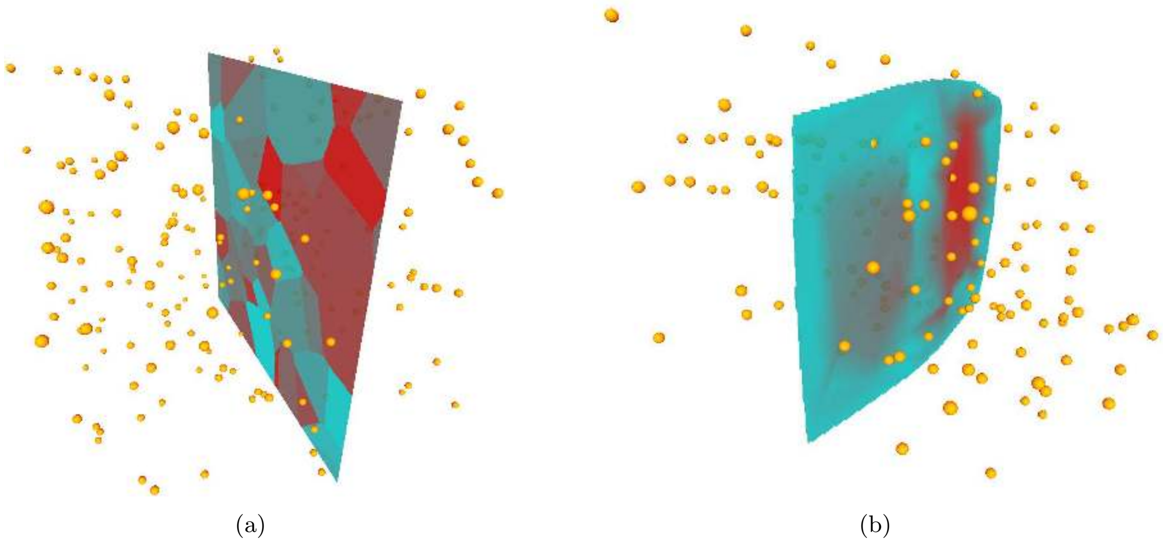


Figure 5.12: (a) A dataset, representing a discrete field, sliced according to a plane. Each colour represents one Voronoi cell. **(b)** A dataset, representing a continuous field, sliced according to a plane.

Therefore, to reduce the number of tetrahedra visited to extract an isosurface, it suffices to store only one tetrahedron from which the extraction starts, and 'traverse' the isosurface. Notice in Figure 5.8 that a field can contain more than one isoline (isosurface) for a given value, and that they are not necessarily connected. Carr et al. (2003) propose a data structure that uses only $\mathcal{O}(n)$ space (n being the number of points in the tetrahedralization) and permits to compute and store the starting tetrahedra for any isosurface.

It should be noticed that none of the 'optimised' methods were implemented in the prototype GIS developed for this research, as the extra data structures do not permit the deletion or the movement of points. If a single deletion or the movement of a single points require recomputing the whole data structure, then it is certainly not worth it. Recently, some theoretical work have been made in that sense (Edelsbrunner et al., 2004), but a report on the implementation (Mascarenhas and Snoeyink, 2005) show that even the implementation in two dimensions is intricate. The choice of implementing a simple algorithm is also corroborated by the practical performances obtained with the prototype GIS, as described in Section 6.4.

5.4.2 Slicing a Dataset

Although the slicing method is rather simple, it can still help users to have a greater understanding of a dataset, and therefore reduce the analysis time. Slicing involves 'cutting' a dataset according to a certain plane, and then visualise the spatial variation of a given attribute on that plane, for instance with the help of colours.

For a discrete field represented with the VD, it would for example mean slicing the Voronoi cells and outputting on the screen the intersections with a colour that is based on the value of the attribute assigned to each cell. Figure 5.12(a) illustrate one example; notice that even if the cells might look like Voronoi cells in two dimensions, they are not.

Figure 5.12(b) shows an example where a continuous field (represented by its samples), is sliced according to a certain plane, but instead of just using the Voronoi cells, interpolation is performed at regular intervals on the plane with natural neighbour interpolation (but any other method could also be used). In the case here, red means that the value of the attribute is higher.

It is moreover possible to display many slices at the same time, and I describe in Section 7.4 how the slices can be valuable for the analysis of geological microstructures.

5.5 Conversion between Formats/Spatial Models

As explained in Chapter 2, trivariate fields can be stored in different formats (or spatial models): voxels, octrees, tetrahedralization, Voronoi diagram, etc. The conversion between formats is an important part of a GIS because datasets will most likely come from different sources, and it is also useful to export datasets in different formats so that they can be used with other programs. The grid (voxels) is by far the most popular format to store trivariate fields in a GIS, and it is also the format in which specialised simulation tools in geoscience will output their results.

5.5.1 Scattered Points to Grid

The conversion from scattered points to grid is trivial: simply interpolate at regular locations in three dimensions (which represent the centre of each voxel) and output the results in the appropriate format (grids can be stored in many ways). Any of the interpolation methods previously described can be used.

5.5.2 Grid to Scattered Points

The inverse operation, the conversion of a grid to scattered points, is more complex. Given a three-dimensional grid, it is possible to simply create one data point at the centre of each voxel in the grid, and then construct the VD/DT of the point set. Observe that this way the Voronoi cells would be cubes equivalent to the original voxels. But that creates a huge dataset, and a lot of the information stored in the data points is usually redundant. This conversion problem can also be seen as a special case of the simplification/generalisation of a field. Indeed, we might want to reduce the number of samples in the field, so that the storage cost of the field is less, and so that subsequent operations on the field are sped up. The field simplification problem is defined as follows. Given a set S of points in $\mathbb{R}^3$ representing a field f (where each point p in S as an attribute a attached to itself), the aim is to find a subset R of S which will approximate f as accurately as possible, using as few points as possible. The subset R will contain the 'important' points of S, i.e. a point p is important when a at location p can not be accurately estimated by using the neighbours of p.

5.5.2.1 Discrete Fields

Discrete fields represented with grids can be simplified with the method described in Section 5.3 and shown in Figure 5.6: delete the Voronoi cells which have the same value as all their natural neighbours.

5.5.2.2 Continuous Fields

Garland and Heckbert (1995) present a review of different methods that have been proposed in the literature for the simplification of TINs, but the same ideas are readily applicable for trivariate fields represented by a point set. Fowler and Little (1979) discuss an algorithm where the important points are specific features in the TIN, e.g. summits, ridges or valleys. This method is difficult to generalise to three dimensions as similar notions are not well-defined. Heller (1990) proposes a solution where, instead of simplify a grid, the final TIN is constructed by starting with a coarse TIN, and a point belonging to the grid is added if its error in the TIN is greater than a threshold value.

Drop heuristic. The simplification algorithm I describe in this section is a generalisation to three dimensions of the *drop heuristic* method (Lee, 1989, 1991). The input of the new algorithm presented is a trivariate field f represented by a set S of points, and it proceeds as follows. First, DT(S) (or VD(S)) is constructed—that will permit us to evaluate the 'error' of each point. Second, an iterative process, where the point in S having the minimum error is deleted from S at each step, begins. The process ends when the minimum error is greater than a given threshold (ϵ_{max}); the points left in S after the deletions form the subset we are looking for. The pseudo-code of the algorithm, called SimplifyField, is presented in Algorithm 5.2. The error associated with each point v, denoted error(v), is calculated by interpolating at location v after the vertex v has been temporarily removed from DT(S), and comparing the value obtained with the real attribute a of v, thus error$(v) = |a - estimation|$. As shown in Figure 5.13 for a one-dimensional case, when the error is more than ϵ_{max} then the point must be kept, if it is less then the point can be discarded.

Lee's original method uses linear interpolation in triangles, i.e. after v has been temporarily deleted from DT(S), the triangulation is updated and the estimation is obtained with the triangle containing location v. However, observe that any other interpolation method could also be used, and that natural neighbour interpolation is particularly suited because it involves the insertion/deletion of vertices. By using the ideas described in Section 5.2.5 to implement InterpolNN, the process could be implemented efficiently. Before temporarily deleting a given vertex v, simply store vol($\mathcal{V}^v_{p_i}$), the volume of the Voronoi cells of the natural neighbour p_i of v, and recalculate vol($\mathcal{V}_{p_i}$) after the deletion of v. This will permit us to obtain the natural neighbour coordinates, and an accurate estimation of the attribute.

The algorithm SimplifyField in its simplest form could be slow, but as van Kreveld (1997) explains, different techniques can be used to speed it up. For instance, after removing a point from DT(S), it is not required to recalculate the error for all the remaining points in S, as only the natural neighbours p_i of the deleted point are affected. It therefore suffices to only recalculate error(p_i). A priority queue storing the error for all the points in S, sorted from the smallest to the biggest error, is kept and updated after each deletion. To optimise further the algorithm, van Kreveld (1997) proposes to keep a pointer from each point v in S to its error(v)

Algorithm 5.2: SimplifyField(S, ϵ_{max})

```
Input: A set $S$ of points in $\mathbb{R}^3$, and the error threshold $\epsilon_{max}$
Output: A subset $R$ of $S$
// let $Q$ be a priority queue
1  compute DT($S$)
2  foreach $v \in S$ do
3      DeleteInSphere (DT($S$), $v$)
4      compute error($v$) and insert in $Q$
5      InsertOnePoint (DT($S \backslash \{v\}$), $v$)
6  end
7  $\epsilon_{min} \leftarrow$ get smallest error($v$) from $Q$
8  while $\epsilon_{min} < \epsilon_{max}$ do
9      DeleteInSphere (DT($S$), $v$)
10     foreach $p_i$ of $v$ do
           // for every natural neighbour of $v$
11         remove error($p_i$) from $Q$
12         recompute error($p_i$) and insert in $Q$
13     end
14     $\epsilon_{min} \leftarrow$ get smallest error($v$) from $Q$
15 end
16 Return points in $S$
```

in the priority queue, and vice versa. A method I suggest involves the use of the rollback of the flips, as used for InterpolNN, so that after deleting temporarily a point, it can be reinserted quickly without computing any InSphere tests.

Time complexity. In the worst case, SimplifyField will take $\mathcal{O}(n^2)$, where n is the number of points in S, but that hides a big constant. Indeed, $\mathcal{O}(n^2)$ is needed to construct the initial DT(S), and there exist some very extreme configurations for which the insertion/deletion of a single point could trigger $\mathcal{O}(n)$ flips. The update of the priority queue could also be slow for the same configurations, since a given vertex could have $\mathcal{O}(n)$ natural neighbours. But if we assume that the points in S are uniformly distributed, then every point has $\mathcal{O}(1)$ natural neighbours (Dwyer, 1991), and the construction of DT(S) is expected to take $\mathcal{O}(n \log n)$ (Edelsbrunner and Shah, 1996). The initialisation of the priority queue takes $\mathcal{O}(n)$, and each insertion/deletion in it takes $\mathcal{O}(\log n)$. The insertions and deletions of points in S are moreover assumed to be performed in constant time, since each vertex has a constant number of natural neighbours. The overall algorithm therefore takes $\mathcal{O}(n \log n)$, under the assumptions described.

5.6 Extraction of Boundary Surfaces

A boundary surface is an object that can be extracted from a field, and it represents the space where the value of an attribute changes abruptly. An example is the delimitation between a zone of high pressure and one of low pressure in a field representing the air pressure.

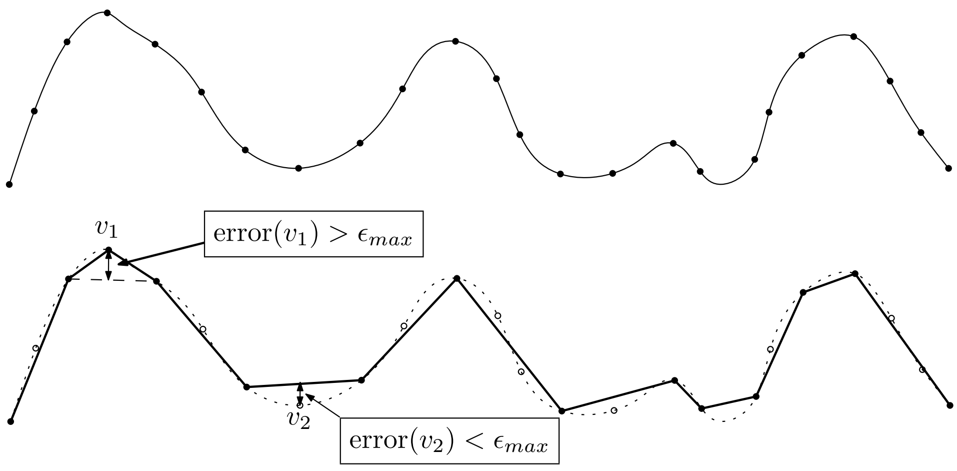

v_1
error(v_1) > ϵ_{max}
v_2
error(v_2) < ϵ_{max}

Figure 5.13: A one-dimensional example of the field simplification operation. Top: the field and its set of samples. Bottom: the subset of samples used to represent the field.

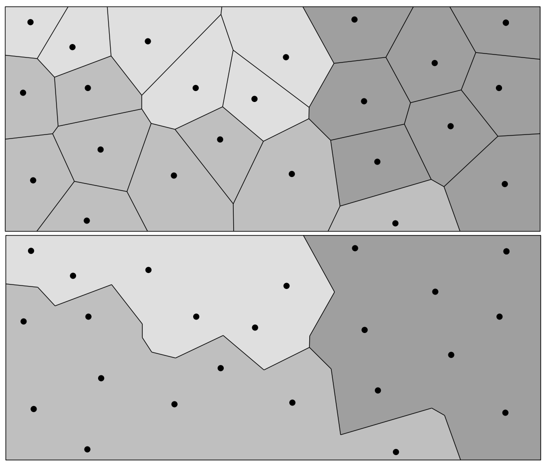
When a discrete field is represented with the VD, a first approximation of the boundaries in the field can be extracted easily. Assume we have a discrete field f representing the rock types and that we have three types, as in Figure 5.14(a). By simply discarding the Voronoi edges shared by cells having the same attribute, we obtain the boundary approximation of the rock types. This technique has also been used for the rapid digitisation of polygonal maps in forestry (Gold et al., 1996). As shown in Figure 5.14(b), in three dimensions, the surfaces extracted are formed of Voronoi faces shared by cells having different attributes.

Observe that boundaries can also be extracted from continuous fields, if the field is reclassified first. An example is a field representing the temperature in Celsius, and we create four categories: -10–0℃; 0–10℃; 10–20℃; and 20–30℃ (observe that the reclassification is an unary local operation in the map algebra jargon). Then it is possible to extract very easily a boundary representing where the air temperature is for instance 0℃.

5.7 Reconstruction of Surfaces

The reconstruction of a surface from a set S of samples that was collected on the surface is an important problem in many disciplines, for instance in engineering when one wants to have a digital model of an instrument or a machine already existing. The samples are usually collected by laser scans, resulting in a dataset having a dense distribution. Figure 5.15 illustrates an example. Observe that the attributes attached to the samples (if any) are not used in the reconstruction, but only the locations of the samples; in other words, do not confuse these surfaces with isosurfaces.

To reconstruct a surface, several methods based on different paradigms have been proposed, see Dey (2004) for a survey. The most successful methods, in the sense that they can theoretically guarantee under certain conditions that the surface will be artifact-free, are based on the VD/DT. I will not go into the details here because it is out of scope for this research, but I will simply state that once DT(S) is built, it is possible to extract triangular faces in DT(S) such that the original surface is approximated (a piecewise linear function is created with the faces).


(a)

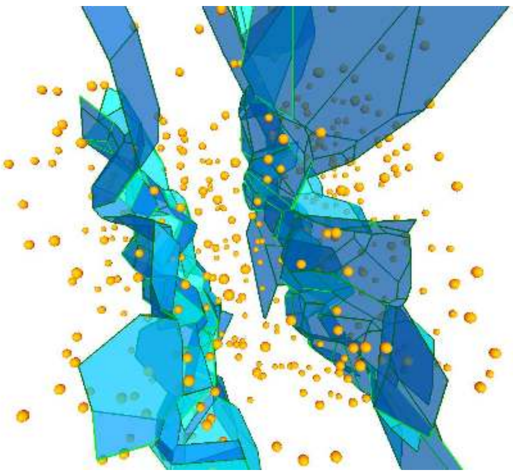

(b)

Figure 5.14: Examples of the extraction of boundaries in a discrete field in **(a)** two dimensions and **(b)** three dimensions.

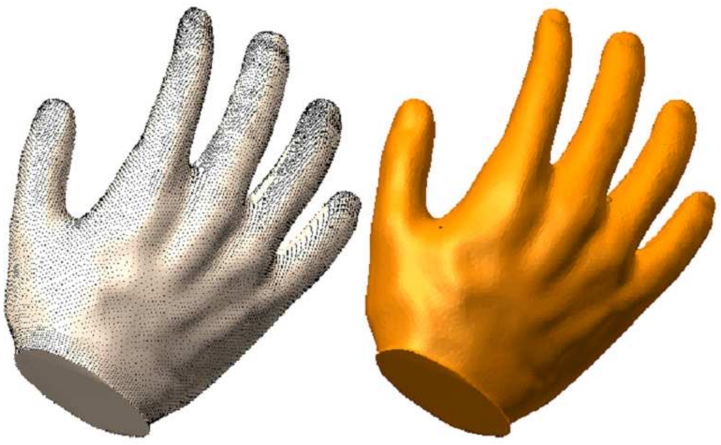


Figure 5.15: Laser scan of a hand, and the reconstructed surface. (Figure taken from Amenta et al. (2001))

Popular implementations of such algorithms are the *Power Crust* of Amenta et al. (2001), and the *Cocone* of Dey and Goswami (2003).

This is clearly not related to fields, but, it is still relevant for a system managing geoscientific datasets. For example, as explained in Chapter 7, some devices used in oceanography can collect field-type data, but they are also equipped with side-scan sonar that can be used to survey the seafloor and the exterior surface of complex man-made objects, to help in submarine construction (Paton et al., 1997). The status of pipelines and cables laid on the seafloor could be investigated for instance.

5.8 Detection of Clusters

Assume that we have a set of points in $\mathbb{R}^3$, and that we want to study the 'shape' of this set, i.e. identify shapes that are implied by the point distribution. Points could for instance be distributed on a surface or a line, or contain clusters. Many techniques could be used, and I describe in this section the *alpha shapes*, or α-shapes.

To explain the concept of an alpha shape, Fischer (2005) compares it to a huge mass of ice cream in $\mathbb{R}^3$ containing 'hard' chocolate pieces at the location of the points in the set S. Assume we use a hypothetical sphere-formed ice cream spoon to carve out all the ice cream we can reach without taking any chocolate pieces (it is also possible to carve out from the inside, i.e. the spoon carves out ice cream at all locations where it fits). The resulting object will be formed of points, arcs and caps, and is not guaranteed to be connected. The α-shape of S is obtained by substituting the arcs and the caps by respectively straight edges and triangles. The 'α' in α-shape is a parameter that defines the radius of the spoon—a small radius will therefore allow us to eat a lot of ice cream.

Formally, the concept of α-shapes is a generalisation of the convex hull of a set of points. α-shapes are actually subcomplexes of the Delaunay tetrahedralization of S, i.e. the triangular faces and edges of a given α-shape can be derived from DT(S). For a given value of α, where $0 < \alpha < \infty$, the α-shape is formed by all the simplices in DT(S) having an empty circumsphere (they do not contain any points in S). Observe that when $\alpha = \infty$, the resulting α-shape is conv(S). Figure 5.16 shows a few examples of α-shapes of a point set in $\mathbb{R}^2$. Edelsbrunner and Mücke (1994) discuss at length α-shapes in three dimensions, and describe an algorithm to construct all the α-shapes of a set of points. It starts by computing DT(S), and then for each k-simplex (where $1 \leq k \leq 3$) it calculates the radius of the smallest circumsphere to obtain the α-interval for which the k-simplex is present. Note that this algorithm was not implemented for this thesis, as the predicates needed are complex and it is difficult to ensure robustness.

As explained in Edelsbrunner and Mücke (1994), when an α-shape algorithm comes with a visualisation software where the user can interactively modify the α parameter and see the shapes created, then it becomes a powerful tool in many application disciplines. They mention, among others, reconstruction of surfaces, automatic mesh generation, and the study of the distribution of galaxies.

Alpha shapes can also obviously be used to detect clusters in geoscientific datasets. Observe in Figure 5.16(c) that when α is small, the cluster of points at the top right of S now forms a polygon that is not connected to the rest of the dataset. Similarly, Lucieer and Kraak (2004)

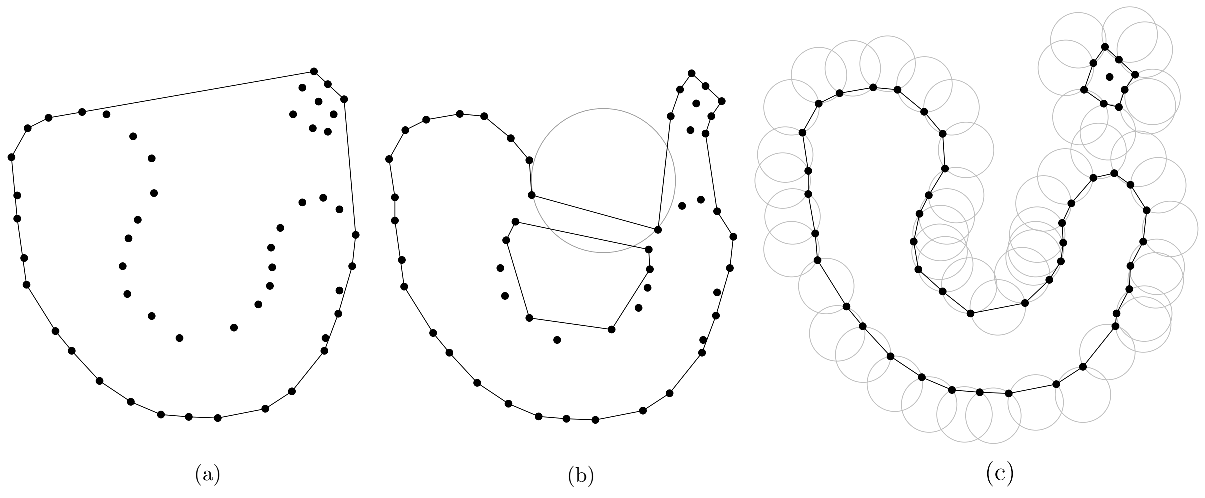


Figure 5.16: Examples of the alpha shapes of a set S of points in the plane when α **(a)** equals ∞ (which results in conv(S)); and when in **(b)** and **(c)** it is the radius of the grey circle(s).

use α-shapes for the classification of remotely sensed imagery: their classification becomes fuzzy because the user can interactively adjust α so that different classes are formed.

Chapter 6

Implementation Issues

This chapter discusses the technical issues that arose during the implementation of the prototype GIS developed for this research. Firstly, the important issue of the robustness of the algorithms developed, in light of the arithmetic used by computers, is examined. Secondly, possible data structures to store the VD and/or the DT are reviewed, and a new one that was developed during this research is presented. Finally, the practical performance of many algorithms developed are analysed with different datasets, and compared with what is arguably the *de facto* standard in computational geometry implementations, the Computational Geometry Algorithms Library (`CGAL`) (Boissonnat et al., 2002).

6.1 Tools Used for the Research

The Borland Delphi environment, the object-oriented version of the language `Pascal`, was used to build the prototype GIS. The algorithms described in Chapters 4 and 5, unless explicitly mentioned, have been implemented and no third-party libraries have been used—this excludes the `OpenGL` library that is used for visualisation. The visualisation engine[1] permits the user to zoom in/out, rotate the scene, and also translate it in any direction.

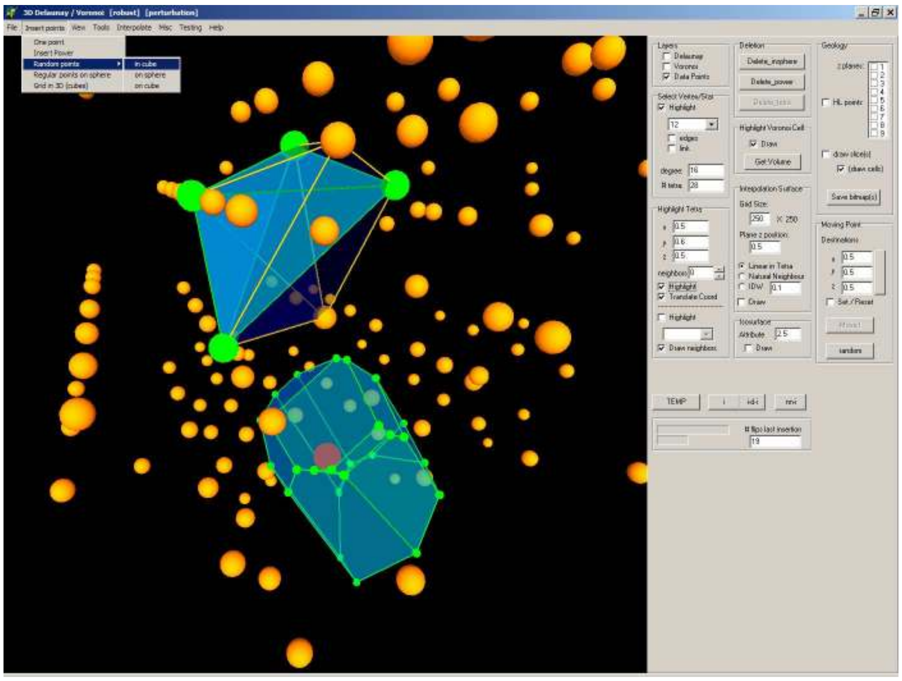
The program developed permits us to construct the VD/DT of a set of points, to manipulate it, to query it and to visualise the results; Figure 6.1 shows its interface.

An important consideration is that the program is a prototype, and not a full-scale GIS. Its main aim is to help in the development of algorithms, and help in testing them. Several common GIS functions have therefore not been included, as the focus was on the algorithms for three-dimensional fields described previously. Also, although the visualisation is an important part of this thesis, the visualisation tools implemented have not been optimised for big datasets, and with more than 500 points, the user can see a major slow down. If the visualisation is removed, all the operations are performed rather quickly, as Section 6.4 reports.

6.2 Robustness Issues

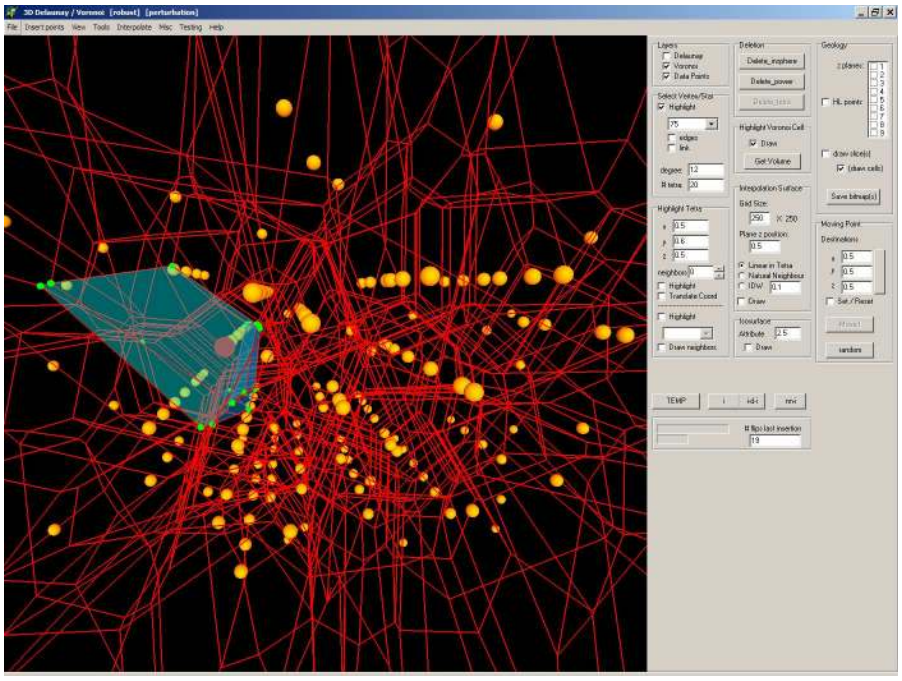
Consider the admittedly simple algorithm AddRealNumbers, as shown in Algorithm 6.1, that adds 0.1 one hundred times. It should obviously returns 'true', but if it is implemented on a computer with floating-point arithmetic, the odds that it will return 'true' are rather

[1]It was developed with `OpenGL` by Rafał Goralski and Marcin Dzieszko.



Layers: Delaunay ☐, Voronoi ☐, Data Points ☑
Select Vertex/Stat: Highlight ☑, 12, edges ☐, link ☐
degree: 16
# tetra: 28
Highlight Tetra: x 0.5, y 0.6, z 0.5, neighbors 0, Highlight ☑, Translate Coord ☑
Highlight ☐
Draw neighbors ☑
Highlight Voronoi Cell: Draw ☑
Interpolation Surface: Grid Size: 250 X 250, Plane z position: 0.5, Linear in Tetra ◉, Natural Neighbour ○, IDW ○ 0.1, Draw ☐
Isosurface: Attribute 2.5, Draw ☐
Geology: z planes: 1 2 3 4 5 6 7 8 9, HL points ☐, draw slice(s) ☐, (draw cells) ☑
Moving Point: Destination: x 0.5, y 0.5, z 0.5, Set / Reset ☐
# flips last insertion: 19

(a)



Layers: Delaunay ☐, Voronoi ☑, Data Points ☑
Select Vertex/Stat: Highlight ☑, 75, edges ☐, link ☐
degree: 12
# tetra: 20
Highlight Tetra: x 0.5, y 0.6, z 0.5, neighbors 0, Highlight ☐, Translate Coord ☐
Highlight ☐
Draw neighbors ☐
Highlight Voronoi Cell: Draw ☑
Interpolation Surface: Grid Size: 250 X 250, Plane z position: 0.5, Linear in Tetra ◉, Natural Neighbour ○, IDW ○ 0.1, Draw ☐
Isosurface: Attribute 2.5, Draw ☐
Geology: z planes: 1 2 3 4 5 6 7 8 9, HL points ☐, draw slice(s) ☐, (draw cells) ☑
Moving Point: Destination: x 0.5, y 0.5, z 0.5, Set / Reset ☐
# flips last insertion: 19

(b)

Figure 6.1: Interface of the prototype GIS built for this research.

Algorithm 6.1: AddRealNumbers

```
Input: —
Output: the result true or false
1 $m \leftarrow 0.0$
2 for $i \leftarrow 0$ to $99$ do
3 |   $m \leftarrow m + 0.1$
4 end
5 if $m = 10.0$ then
6 |   Return true
7 else
8 |   Return false
9 end
```

low (if not nil). This is because floating-point arithmetic offers only an approximation to real numbers, which are always rounded to the closest possible value in the computer. When one chooses to implement an algorithm with floating-point arithmetic, it is important he understands the consequences of his choice. Floating-point arithmetic to represent real numbers is ubiquitous because it has many advantages: it is available almost on every platform, and, more importantly, it has been highly optimised so that arithmetic operations are performed very fast.

I began implementing the prototype GIS for this thesis under the naive assumption that floating-point arithmetic was 'good enough' and that I would be able to understand the special cases, and fix them by adding some more code. My method was the most widely used to fix numerical non-robustness: the *tolerance*. In other words, if two values are very close to each other, then they are equal. Tolerances can probably yield satisfactory results for simple problems (e.g. the intersection of two line segments), but for more complex ones like the construction of the DT, where the combinatorial structure could be invalidated by small movements of the vertices, it is risky. How should one define the 'optimal' tolerance? I observed that tweaking the tolerance to fix a given problem was usually easy, the problem being that it was also creating another problem somewhere else. Even with the tolerance well-defined (or so I thought), during the development of algorithms I became frustrated because my program would sometimes crash, or even output something that was not a valid tetrahedralization. "Was it the arithmetic? My algorithm? Or a mistake I made while coding it?" I wondered all the time. The development of algorithms was hindered by the fact that the source of the problem was never known. The only solution was to use exact arithmetic: my first question was therefore answered—that left two unanswered but that was better than nothing!

The major obstacle to using exact arithmetic (Yap and Dubé, 1995) is the speed of computation: it is very slow. Unlike floating-point arithmetic, an arithmetic operation can not be assumed to be performed in constant time. The complexity of an operation depends on the numbers n of bits used to store a number, and the multiplication of two numbers can have for instance a complexity of $\mathcal{O}(n^2)$. Karasick et al. (1991) reported that the naive implementation of a divide-and-conquer algorithm to construct the two-dimensional DT was slowed down dramatically by a factor of 10 000, although they showed that by carefully selecting when to use exact arithmetic, they could reduce the factor to around 5 for points uniformly distributed.

As mentioned in Section 4.2.2, the algorithms for manipulating a DT make their only important decisions based on the result of two geometric predicates. For this reason, if one wants to build robust algorithms, only these two predicates need to be implemented with exact arithmetic. Observe that we are not interested in the exact values returned by ORIENT and INSPHERE, but rather by the sign of the result (although we must be able to detect when the value of the determinant is exactly 0). Floating-point arithmetic will most likely compute correctly the sign of the determinant when the points involved in a predicates are 'clearly' in general position, but when four points are nearly coplanar, there is a chance that ORIENT returns that they are coplanar. Similarly, five cospherical points can be mistakenly considered not cospherical by INSPHERE because the result of the determinant is not exactly 0, but perhaps something like $1,555234 \times 10^{-18}$. The problem of robustness of an algorithm is therefore tightly linked to the problems of degeneracies, as it is special cases that will create problems when computing a predicate.

The solution I have retained for the implementation of my prototype is due to Shewchuk (1997a,b). He developed and implemented—his code is publicly available on the Internet[2]—fast geometric predicates that are 'adaptive', which means that their speed is inversely proportional to the degree of uncertainty of the result. His method works like a filter that will activate exact arithmetic only when it is needed, but since the exact result of the predicate is not sought, as soon as the sign of the determinant can be correctly computed, the process stops. He shows that using his adaptive predicates for building a DT in two dimensions slows down by about 8% the total running time for 1 000 000 randomly generated points, and by about 30% when those points form a tilted grid. In three dimensions, his predicates slows down by about 35% the total running time for 10 000 randomly generated points, and by a factor of 11 when those are generated on the surface of a sphere. As Shewchuk (1997b) states: "these predicates cost little more than ordinary nonrobust predicates, but never sacrifice correctness for speed".

Because the predicates ORIENT and INSPHERE have been implemented with Shewchuk's predicates, the construction of the DT and the deletion of a vertex in the prototype are totally robust. On the other hand, the movement of a point in a VD/DT requires more operations that were not implemented with exact arithmetic. In most cases that does not create a problem, but it is still possible that the program crashes because of imprecision with numbers.

Also, Shewchuk (1997a) implemented only the predicates necessary for constructing the two- and three-dimensional DT, which means that the only arithmetic operations available are $+$, - and $\times$ (the sign of a determinant does not require the $\div$). For this reason, the deletion with the power distance (see Section 4.4.5.1) is not robust because the power predicate requires a division. The construction of the VD is not totally robust either because the computation of Voronoi vertices (the centre of a sphere defined by four points) requires a division. In the prototype, for some very degenerate configurations (e.g. points distributed uniformly on a sphere), the division behaves badly and corrupts the VD.

6.3 Data Structures[3]

As it was mentioned in Chapters 3 and 4, most of the algorithms and implementations available to construct the three-dimensional VD (and thus indirectly the DT) store only the DT and

[2] www.cs.cmu.edu/~quake/robust.html

[3] This section is the result of joint work with Christopher Gold and Marcin Dzieszko, and is based on: Gold et al. (2005), Ledoux and Gold (2005b), and Ledoux and Gold (2006c).

perform their topological operations on tetrahedra. This is usually justified by the fact that given one structure the other is implicitly known, and also because managing and storing simplices over arbitrary polyhedra simplifies the topological operations and permits the design of simpler data structures.

The prototype developed for the work described in this thesis was first implemented using a very simple data structure storing every tetrahedron and its four adjacent tetrahedra. Although this results in an efficient implementation, both in terms of memory and speed (for the construction and the manipulation operations), it has major drawbacks when analysing a dataset because, as shown in the previous chapter, many operations are based on the VD. As described in Section 4.1, the computation of the Voronoi elements (vertices, edges, faces and polyhedra) is a computationally expensive operation that can slow down the prototype and hinder the analysis of a dataset.

It can be argued that storing both the DT and the VD simultaneously can be beneficial for a broad range of applications, and that doing so offers a more powerful and flexible solution than the other known data structures storing three-dimensional cell complexes (see below for a review). Keeping both subdivisions can optimise certain spatial analysis operations in GIS based on the VD, and permit new applications involving the volume and the boundary of an object. It also offers a flexible solution, because attributes for any elements of both subdivisions can be stored, which is of considerable importance for the modelling of real-world features. On the other hand, keeping two subdivisions at the same time obviously has drawbacks: it increases the requirements for storage, and the construction/modification algorithms will be slower. I nevertheless believe that, in many real-world applications, the major constraint is not the speed of construction of the topological models of a large number of points or the space needed in memory, but rather the ability to interactively construct, edit and query the desired model.

What is needed is a data structure like the *quad-edge* (Guibas and Stolfi, 1985)—it has been used for many years to represent the primal and dual subdivisions of two-dimensional manifolds—but for three-dimensional manifolds.

This section presents a new data structure to store and manipulate simultaneously the primal and dual subdivisions of a three-dimensional manifold. The structure, called the *augmented quad-edge* (AQE), is based on the quad-edge structure. Each 3-cell of a complex is constructed using the usual quad-edge structure, and two 3-cells are linked together by the edge that is dual to their shared face. The basic properties, the primitive operators, and the storage costs of the AQE are also presented. The primary feature of the data structure is ease of use in managing and navigating these cell complexes. While high in storage, it is computationally efficient, requiring no searching operations in order to move from cell to cell.

6.3.1 Related Work

Because the computer science community tends to favour efficient implementation over the analysis of datasets, most papers in the literature discuss data structures for storing only tetrahedra. The simplest data structure, as shown in Figure 6.2 and as implemented for the prototype, considers the tetrahedron as being its atom and stores each tetrahedron with four pointers to its vertices and four pointers to its adjacent tetrahedra. This is the direct generalisation of the triangle-based data structure that has been used for many years to store the two-dimensional DT. As explained in Section 4.2.2, the vertices and faces of each tetrahedron τ must be 'oriented' and ordered in a consistent way if we want to manipulate or perform some

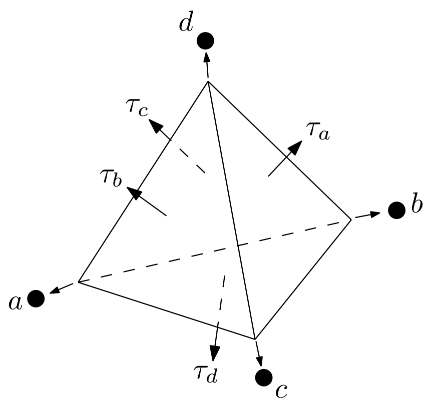


Figure 6.2: The tetrahedron-based data structure.

operations on a DT. The four vertices in τ (a, b, c and d) are ordered such that ORIENT (a, b, c, d) returns a positive value, and each adjacent tetrahedron τ_i is indexed in such a way that the vertex i points to it. In other words, τ_i, that is adjacent to τ, shared three vertices with τ, but does not contain the vertex i. This is a very space-efficient structure because it can be stored in a simple array of pointers to vertices and tetrahedra. It also yields a fast implementation, and it is therefore the structure of choice in many projects. Shewchuk (1997b) notably reports the implementation of his program to compute the constrained Delaunay triangulation in the plane to be nearly twice as fast when a triangle-based structure is used, instead of the quad-edge. `CGAL` notably uses it in three dimensions but has added to each vertex a pointer to one of its incident tetrahedra, to speed up the extraction of the Voronoi cells (Boissonnat et al., 2002). It is also possible to store the tetrahedra implicitly by considering a data structure where the atom is a triangular face having three pointers to its vertices and six pointers to its adjacent faces, as described by Shewchuk (1997b).

Many data structures to store arbitrary three-dimensional cell complexes are extensions of work done for the two-dimensional case. Examples are the *half-face* of Lopes and Tavares (1997) (extension of the *half-edge* (Mäntylä, 1988)) and the *Generalized Maps* (G-Maps) (Bertrand et al., 1993; Lienhardt, 1994). The latter structure, which is actually valid to represent a broader class of objects called *cellular quasi-manifolds*, can be seen as a generalisation to d-manifolds of boundary models as each k-cell (where $0 < k \leq d$) is recursively decomposed into cells of lower dimensionality, and the topological relationships between adjacent k-cells are kept. The resulting structure is thus very space-consuming, as each lower-dimensionality cell must be stored in cells of higher dimensionality: a vertex shared by five edges is for instance stored in each of the five edges. It is nevertheless being used in a commercial system for the modelling of geoscientific data[4]. Although G-Maps stores only one subdivision, it offers efficient functions to extract all the elements of the dual subdivision. One of the few structures permitting the simultaneous preservation of primal and dual subdivisions of 3-manifolds is the *facet-edge* structure of Dobkin and Laszlo (1989). As its name implies, the atom is a pair formed of a face and an edge, and the next facet-edge pairs (one in each direction) around the face and the edge (incident faces to the edge) are stored. Unlike G-Maps, a face shared by two cells needs only be stored once. The facet-edge structure comes with a set of operations to modify cells and to navigate within both subdivisions. Its generality makes navigation within a single cell impossible, and hence the authors suggest storing extra information for each edge

[4]The gOcad system (`www.gocad.org`).

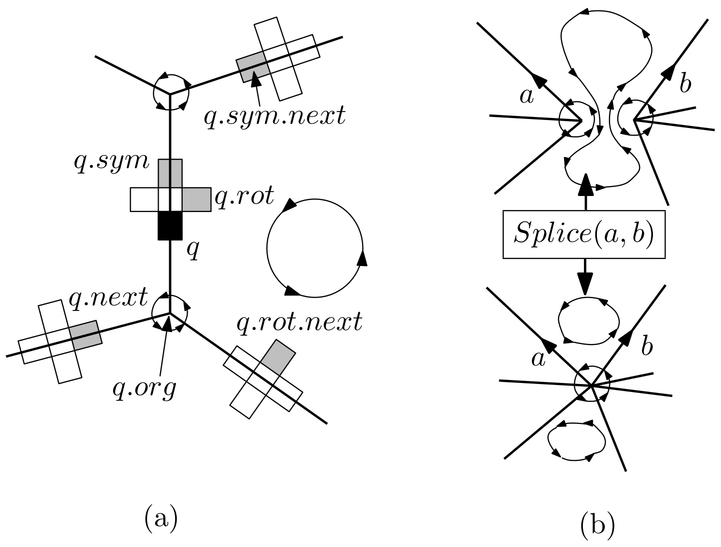


Figure 6.3: (a) The quad-edge data structure and some basic operators (only one subdivision is represented). The starting quad q is the black quad, and the resulting quads are grey. **(b)** The *Splice* operator.

and using the quad-edge operators. Also, unlike the quad-edge that is being used in many implementations of the two-dimensional VD/DT, the facet-edge has been found difficult to implement in practice and, to my knowledge, has not been used in 'real projects'. It should finally be noticed that the facet-edge structure has also been generalised to any dimensions by Brisson (1989).

6.3.2 Quad-edge Structure

The quad-edge data structure (Guibas and Stolfi, 1985) was developed to represent simultaneously the primal and dual subdivisions of a 2-manifold, and also to navigate from edge to edge in these subdivisions. Each quad-edge represents one geometrical edge of a subdivision, and this edge is decomposed into four directed edges: two in the primal, and their two respective dual edges (as seen in Section 3.1.5, the dual of an edge in 2D is an edge). Each directed edge is represented by a *quad*, which, in a given subdivision, will either point to one vertex or to a face; in the dual, a quad points to the dual of what it is pointing to in the primal (remember that in 2D, faces and vertices are dual). Its advantages are firstly that there is no distinction between the primal and the dual representation (it is symmetric with respect to edges and faces/polygons), and secondly that all operations are performed as pointer operations only, thus giving an algebraic representation to its operations. Figure 6.3(a) shows the basic structure, with four branches (quads) for each edge of the graph being stored (the dual edges are not drawn), one for each of the incident vertices or faces. There are three pointers on each quad: one to the vertex or face object (org), one to the next anticlockwise quad-edge around that object ($next$), and one to link the four quads together in a loop (rot). The three operations org, rot and $next$ are sufficient for navigating any subdivisions, but higher level operators can also be defined: sym is for example equal to two rot applied consecutively, thus $q.sym = q.rot.rot$.

In brief, all vertices or faces have complete topological loops around them. Notice that the *rot* operation actually permits us to navigate between the primal and the dual subdivisions. Guibas and Stolfi (1985) also use the *flip* operator (do not confuse with the flip as described in Section 4.2.3) to navigate to the other side of the 2-manifold (a manifold is orientable), but this operation is not used for this research.

Only two commands are used to modify a graph: *MakeEdge* to create a new edge on a manifold, and *Splice* to connect/disconnect quad-edges together (illustrated in Figure 6.3(b)). In the simplest case, *Splice* connects two separate *next* loops, joining the two nodes together, and at the same time splitting the *next* loop around the common face. *Splice* is its own inverse.

Guibas and Stolfi (1985) give many implementation details about the structure and its operations, and one can therefore implement it relatively easily.

6.3.3 Augmented Quad-edge Structure

The augmented quad-edge (AQE) uses the 'normal' quad-edge, which is valid for any 2-manifolds, to represent each 3-cell of a 3D complex, in either space. For instance, each tetrahedron and each Voronoi cell are independently represented with the quad-edge, which is akin to a boundary representation (*b-rep*). With this simple structure, it is possible to navigate within a single cell with the quad-edge operators, but in order to do the same for a 3D cell complex two things are missing: a way to 'link' adjacent cells in a given space, and also a mechanism to navigate to the dual space. First, notice that in this case two of the four *org* pointers of a quad-edge point to vertices forming the 2-manifold, but the other two (which in 2D point to the dual, or a face) are not used in 3D. Remember also that in 3D the dual of a face is an edge. The idea is therefore to use this dual edge to 'link' two cells sharing a face: the unused face pointers simply point to their dual edge. This permits us to 'link' cells together in either space, and also to navigate from a space to its dual. Indeed, we may move from any quad-edge with a face pointer to a quad-edge in the dual cell complex, and from there we may return to a different 2-manifold in the original cell complex if needed.

A quad-edge is divided into four quads q, and there exist two types of quads: a q_f points to a face, and a q_v points to a vertex. One *rot* operation applied to a q_f returns a q_v, and vice versa. A q_f identifies uniquely, like the facet-edge (Dobkin and Laszlo, 1989), a pair (face, edge). Therefore q_f has also a linked quad $q_f^\star$ in the dual that is defined by (face*, edge*); remember from Section 3.1.5 that if C is a k-cell in $\mathbb{R}^d$, its dual cell is denoted by $C^\star$ and is a $(d-k)$-cell.

One issue remains to be resolved: as each face is penetrated by several dual edges, a consistent rule must be defined to select the appropriate one. Indeed, with the AQE, the dual edge to a face has to be stored for all the dual 3-cells sharing that edge. A triangular face has for example three dual edges since each of its three vertices becomes a 3-cell in the dual. A q_v has its *org* pointer set to a vertex, and a q_f has its *org* pointer set to $q_f^\star$. The pointer to $q_f^\star$ from q_f is called *through*, as shown in Figure 6.4(a). The quad $q_f^\star$ is defined as belonging to the dual cell which encloses the node pointed to by $q_f.rot = q_v$. This is sufficient to define the *through* pointer structure. In Figure 6.4(a), the triangular face abc has three dual edges and may be represented by the quad q_f (the black quad) on the edge ab. The dual edge of abc is uv, and the grey face represents the face that is dual to the edge ab. The operation $q_f.through$ gives $q_f^\star$ which is situated on the edge uv and on the face dual to ab; this face belongs to the cell

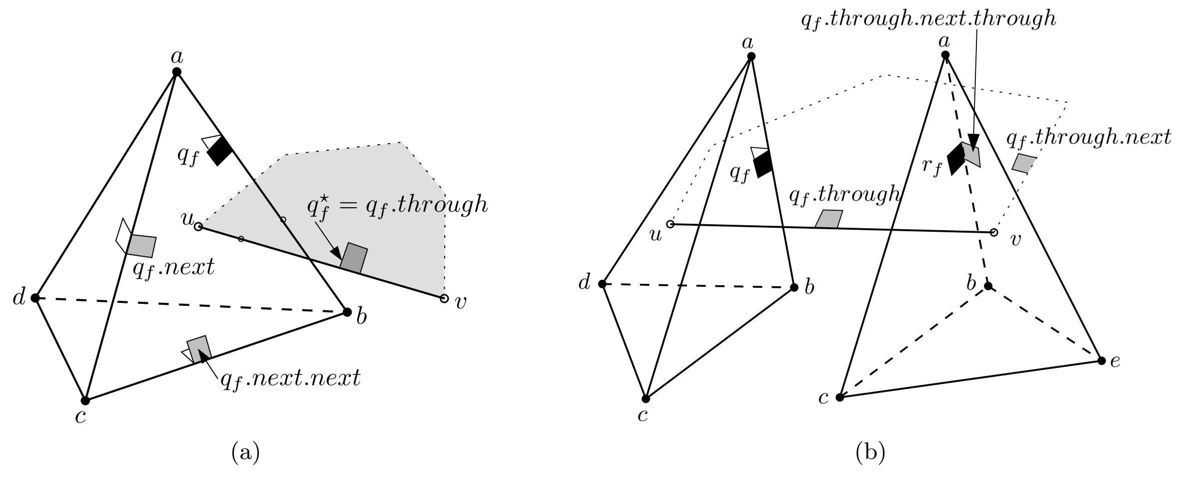


Figure 6.4: (a) The *through* pointer. The black quad q_f is the starting one, and the grey ones are resulting quads after an operation. **(b)** The *adjacent* operator. The quads q_f and r_f represent the same pair (face, edge), but belong to different 3-cells.

dual to vertex b since $q_f.rot.org$ gives b. Note also that $q_f.next.through$ gives a quad on a face which belongs to the cell dual to the vertex a; and $q_f.next.next.through$ to c.

6.3.4 Navigation and Construction Operators

Navigation Operators. As mentioned earlier, because the quad-edge structure is used to represent each cell of a three-dimensional cell complex, the 'usual' navigation operators (*next* and *rot*) can be used to navigate within a 3-cell. Only one new operator is needed to navigate within a three-dimensional cell complex and to be able to visit every polyhedron, face, node or quad-edge: the *through* operator, as described in the previous section.

Let q_f be a quad and $q_f^\star$ its dual quad, then the following properties are valid:

$$q_f^\star = q_f.through \tag{6.1}$$

$$q_f = q_f.through.through = q_f^\star.through \tag{6.2}$$

Different higher-level navigation operators can also be defined. One that is particularly useful is an operator for 'jumping' from a cell to another adjacent cell, in either space. This navigation operator is called *adjacent*. Let q_f be a quad on the boundary of a common face between two cells, as shown in Figure 6.4(b). We want to 'jump' to the quad r_f representing the same pair (face, edge) as q_f but on the adjacent cell. We do this by using the dual structure to navigate. Starting from q_f, $q_f.through$ gives $q_f^\star$, and $q_f^\star.next$ gives a quad whose *through* pointer gives a quad in the original space but in a neighbouring cell. The *rot* operator is used twice to set the result to r_f. Thus,

$$q_f.adjacent = q_f.through.next.through.rot^2 \tag{6.3}$$

Other relations are also interesting. First, the *adjacent* operator is symmetrical:

$$q_f = q_f.adjacent.adjacent \tag{6.4}$$

Second, if q_f and r_f are adjacent quads as in Figure 6.4(b) (such that $r_f = q_f.adjacent$), notice that their respective dual quads $q_f^\star$ and $r_f^\star$ are also adjacent, i.e. they represent the same pair (face, edge) but are part of different 3-cells.

Construction Operators. New construction operators are required to build the DT and the VD. There are three categories of operators:

Creation of new independent 3-cells, e.g. a Delaunay tetrahedron or a Voronoi cell. Because the quad-edge structure is used, its construction operators *MakeEdge* and *Splice* are the only operators needed. This operator is called *CreateCell*.

Modification of existing cells. When a topological operation is performed on the primal subdivision, the dual must also be updated. The operators *Sharpen*, *Truncate* and *ReplaceEdge*, described in Section 6.3.6, are examples of operators that can be used to update the VD when the DT is modified.

Assemblage or linkage of two cells. The operators *Sew* and *Unsew* are used to respectively set and reset the *through* pointers of all q_f belonging to a face shared by two cells.

6.3.5 Storage Costs and Comparisons

This section analyses the storage costs of the AQE when it is used to represent both the DT and the VD.

With the AQE, each cell (tetrahedron or Voronoi cell) is stored separately. Each tetrahedron contains: six edges, each represented by four quads containing three pointers (*org*, *next* and *rot*). This makes a total of 72 pointers. Notice that every q_f has one and only one $q_f^\star$, and also that the same is true for every q_v (there is always one q_v when a *rot* is applied to a given q_f). As a result, the total number of pointers for the dual is also 72, which makes a total of 144 pointers for each tetrahedron.

This is obviously more than the data structures used to store only the DT. For example, the tetrahedron-based structure uses only eight pointers per tetrahedron. To my knowledge, the only data structure preserving both a primal and dual 3D subdivision is the facet-edge (Dobkin and Laszlo, 1989). With it, each pair (face, edge) contains (considering both the primal and the dual): four pointers to vertices (two Delaunay and two Voronoi), and also four pointers to the *next* facet-edge (the navigation requires the use of the dual to access the adjacent facet-edges around the face and the edge). There are three pairs per triangular face, and a tetrahedron contains four faces. However, with this structure, the common faces between two tetrahedra are not stored twice—and we can state that each face is shared by two tetrahedra, except the ones on the convex hull of the data set. As a result, the total number of pointers per tetrahedron is 48, but, as explained previously, extra information needs to be stored if one wants to use the quad-edge operators to navigate within a single cell.

To summarise, the AQE is about three times more space-consuming than the facet-edge, in the worst case. A factor of two is consistent with my initial objective to maintain the primal and dual structures simultaneously, and to use the same set of operations in each space. This can be achieved by compressing the quad-edge as Guibas and Stolfi proposed: the *rot* pointers between quads are not explicitly stored and the last two bits of each pointer are used to identify each quad of a quad-edge.

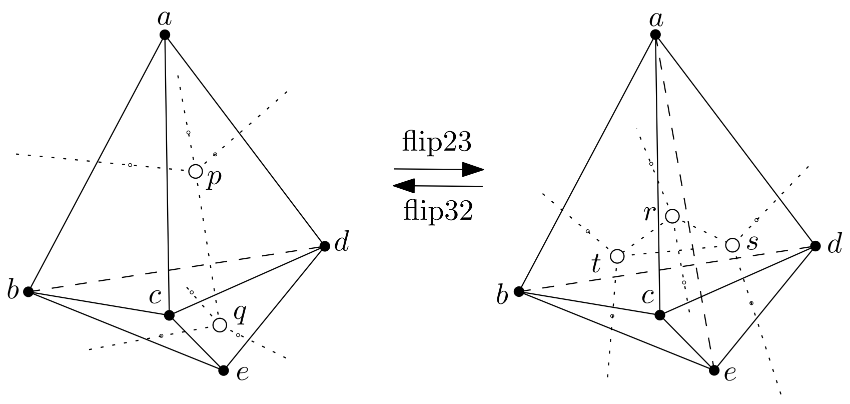


Figure 6.5: *Truncate* and *Sharpen* operators. The DT is in black lines, and the VD is dotted lines and white vertices.

6.3.6 Implementation of the Flips with the AQE

To construct and manipulate simultaneously the DT and the VD, we need to modify the flip operations so that the VD is updated as well. Because the flips are simple topological operations involving a fixed set of tetrahedra, the operations to update the VD are relatively simple and can be described using algebraic statements exclusively. The only data elements are the (Voronoi or Delaunay) quad-edges and the (Voronoi or Delaunay) nodes. For the rest of this section, it is assumed that the primal subdivision is the DT and the dual the VD.

Each flip in the primal is implemented with the three construction operators defined earlier. The tetrahedra involved in the flip need first to be unsewn (with *Unsew*) from the ones not involved, deleted, and then new tetrahedra need to be created (with *CreateCell*), sewn (with *Sew*) together and also to the ones not involved in the flip.

Although more complicated, the update of the dual requires only three new operators, as shown in the following. Recall that the Voronoi cell of a point is formed by all the centres of the circumspheres of the tetrahedra incident to it. A flip will only modify (delete and create) tetrahedra that are located inside conv(T), where T is formed by the five points involved in the flip. Therefore, only five Voronoi cells are modified by a flip23 or a flip32, and a flip14 modifies four Voronoi cells and creates a new one. The faces on the boundary of conv(T) are not modified by a flip, and therefore to update the dual after a flip only the *through* pointers of the edges dual to these faces have to be updated. For example, a flip23 has only two different kinds of vertices (referring to Figure 6.5): a and e both have one incident tetrahedron before the flip23 and three after; and b, c and d all have two before and after (but they are different). For the vertex a (as for e), the operator *Truncate* is needed to update the dual and involves deleting the Voronoi vertex p dual to its incident tetrahedron, creating three new Voronoi vertices (forming a new Voronoi face rst) and finally linking these vertices to the three Voronoi edges dual to the three Delaunay faces incident to a. The reverse operation, needed for a flip32, is called *Sharpen*.

The operator *ReplaceEdge* is needed to update the Voronoi cell of the vertex b (or c or d), and involves setting the dual edges of the four incident Delaunay faces of b to the two new Voronoi points r and t created by a flip23 or flip32. Notice that the Voronoi cell of b is modified, but contains the same number of vertices, edges and faces before and after a flip.

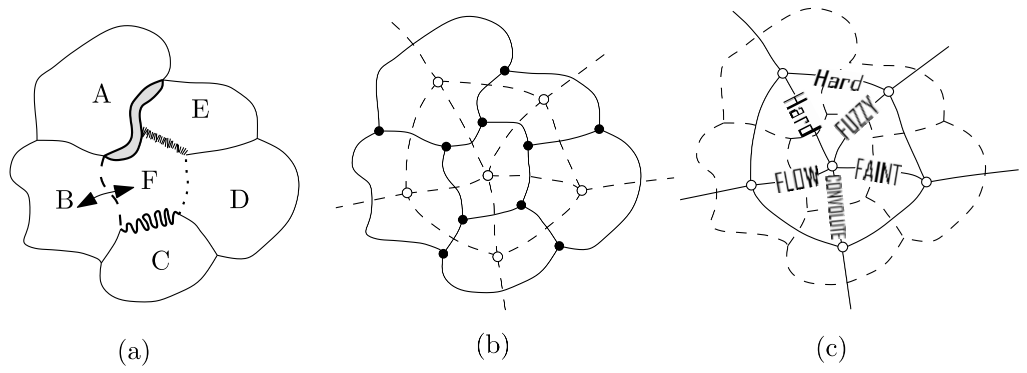


Figure 6.6: (a) Six map objects and their boundaries. **(b)** The same map stored as a graph and its dual (dotted lines). **(c)** The dual graph is used to describe the relationships between adjacent polygons. (Figure after Gold (1991))

As for the flip14, it can be implemented with the operators already described. Referring to Figure 4.3(b) on page 50, a new Voronoi cell (dual to e) must be created and sewn to the ones of a, b, c and d. The updates to the Voronoi cells are simply done with *Truncate* since the points a, b, c and d all have one incident tetrahedron before the flip14, and three after.

The operations described here have been implemented successfully, and allow both the DT and the VD to be updated after a flip, thus it is possible to insert a new point in a VD/DT, but also theoretically delete or move one.

6.3.7 Applications

The AQE is a general-purpose data structure that can be useful not only for representing fields with the VD/DT, but actually for a variety of applications where three-dimensional solid objects are involved. The following presents two examples of applications for which the dual of a subdivision is meaningful, and therefore is worth storing. Storing two subdivisions can optimise some of the operations necessary for an application and offer a more flexible option for the storage of attributes.

A two-dimensional example where the dual is meaningful was given by Gold (1991). He presented a polygonal map, as the one in Figure 6.6(a), where each polygon (a map object) has certain attributes, and where the boundaries between two map objects also have some sort of attributes, e.g. the boundary form or the flow. His claim was that the boundaries do not characterise *per se* any of the objects, but rather the adjacency relationships that exist between any two objects. He therefore suggested to store the relationships between two given map objects in the edge that is dual to their shared boundary (Figure 6.6(c)).

6.3.7.1 Crust and Skeleton

Although most of the efforts of the meshing community have been targeted at storing regular and irregular subdivisions, some real-world applications are only possible when both the DT and the VD are known *explicitly*, i.e. properties of the DT and the VD are both used to solve a problem. Obvious two-dimensional examples are the definitions of the crust and skeleton of

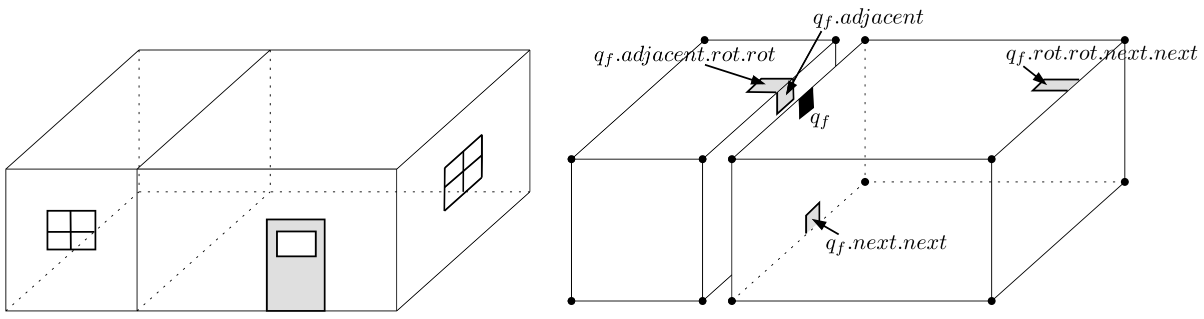


Figure 6.7: An apartment building can be modelled with the AQE. The dual subdivision is not shown.

a sufficiently well-sampled polygon. It has been shown that these two structures are subsets of respectively the DT and the VD (Amenta et al., 1998), and the extraction of either the crust or the skeleton requires the knowledge of both the DT and the VD. Gold and Snoeyink (2001) show that the crust/skeleton concept helps in many cartographic problems such as the reconstruction of terrain models from contour lines, the estimation of a river network from boundaries, and the extraction of "topology" from scanned maps. One interesting application is polygon generalisation: a polygon may be generalised by first constructing its skeleton, then simplifying this skeleton, and finally reconstructing the polygon from the simplified skeleton. This idea has been extended to three dimensions for the reconstruction and simplification of surfaces and features from laser scans (Tam and Heidrich, 2003). The method is similar to the two-dimensional one and involves the extraction of the DT of the set of points, and the 3D skeleton of the reconstructed objects (with the VD). The surface is simplified by modifying the skeleton, which as a consequence modifies the crust.

6.3.7.2 Modelling Apartment Buildings

Perhaps the main advantage of using the AQE is that attributes can be stored for any elements of both subdivisions (vertices, edges, faces and volumes). Indeed, the vertex attributes may be stored along with the $x-y-z$ coordinates in the vertex; edge attributes in a quad pointing to a vertex; face attributes in a quad pointing to a face; and volume attributes in the vertex that is dual to a volume.

One potential three-dimensional application exploiting duality to store attributes is the modelling of an apartment building. Each room is represented by one 3-cell (see Figure 6.7 for a simple example), and in a similar way to the example in Figure 6.6, the dual edge to each face (e.g. the four walls, the floor or the ceiling) is used to store the relationships between two adjacent rooms, such as the visibility (instead of explicitly representing a window for instance), or the noise transmission. Moreover, it is possible to store attributes for the properties of different parts of one room: the finish of a wall or the floor type (attributes attached to 2-cells), the type of moulding (attributes attached to the 1-cells), or the properties for the whole room (e.g. its volume or renting price) can be stored in the dual vertex to the 3-cell. A major advantage of using the AQE for modelling apartment buildings is that a wall (or a floor) is stored twice, once for each of the two rooms sharing the wall. This is consistent with the way one would conceptualise many rooms in a building: the wall separating two rooms will not necessarily have the same properties on each side. For example, the finish on one side of the wall might be different from the one on the other side, or one side might have a gold ceiling moulding and the other no moulding at all.

The AQE can also be used to search efficiently a three-dimensional cell complex. Referring to Figure 6.7, if q_f (the black quad) is a quad representing the wall between the two rooms, it is possible to retrieve all the elements forming the room on the right by using breadth-first search on the 3-cell representing the room (using quad-edge operators). Each element of the wall that q_f represents can also be retrieved with the quad-edge operators *rot* and *next*; and the same is true for the ceiling or the floor that can be accessed directly from the AQE algebra, without searching. The attributes characterising the relation between the two rooms are obtained with $q_f.through$, and then the attributes of the whole room can be obtained directly because they are stored in the vertex dual to the room. Also, from q_f it is possible to navigate locally and visit the adjacent room with $q_f.adjacent$, and then for instance move to its ceiling by applying *rot* twice. Given a k-cell C, it is actually possible to identify all the l-cells (where $k < l \leq 3$) incident to C using the navigation operators of the AQE described earlier. Notice that in the case of retrieving all the elements incident to a vertex one can simply move to the dual subdivision and perform a breadth-first search on the 3-cell because the elements retrieved are dual to elements incident in the primal. A further important consideration is that a complete three-dimensional cell complex can be easily and efficiently traversed without storing extra information (see Section 4.2.4).

When the attributes are included in a search, it is for example possible to look for an edge that represents gold ceiling moulding, incident to a white ceiling and a blue wall, and with a red corner boss; or for all the rooms having more than 100 m^2 but not having a carpet.

6.3.8 Discussion

It should be noticed that the AQE is only valid for cell complexes, and that subdivisions where 'dangling' elements are present cannot be represented with the structure in its current form. In other words, non-manifold objects, such as the one shown in Figure 3.4(c) on page 35, can not be represented.

The AQE preserves the lower dimensionality (two-dimensional) navigation and construction operations, as well as keeping the underlying quad-edge properties in three dimensions. As a consequence, the data structure is directly navigable and forms an 'edge algebra', allowing us to combine our pointer operations by following well defined rules, and to show the equivalence of different ways of getting to the same destination. Also, every operation has an inverse.

While the data structure requires somewhat more storage than some more simple structures, it has the advantage of conceptual simplicity and involves only a simple extension of the 2D topological relationships. The fact that the dual subdivision is also stored permits the storage of attributes for any elements of a three-dimensional cell complex, and makes the structure particularly valuable for a wide range of applications.

6.4 Practical Performance

I report in this section experiments that were made on a Pentium 4 (2.8 GHz), with 1 GB of RAM and running the Windows XP Professional operating system. I tested the running time of a few algorithms described in Chapters 4 and 5: construction of the DT, deletion of vertices in the DT, interpolation with natural neighbours, and extraction of isosurfaces.

The practical performances of the algorithm MoveOnePoint are difficult to compare with other implementations since I could not have access to any, and at this moment only one point is allowed to move at a time in my prototype. Since the movement of a point uses mostly the operations developed for inserting and deleting a point, and since this section demonstrates that these are efficient, I can conjecture that the operation is efficient in practice. The major problem with the implementation is that it still sometimes crashes because of floating-point arithmetic.

Because all the experiments had to be made under Windows, the choice of implementations to compare mine with was quite limited, as most computer scientists use and develop software under Linux/Unix. To my knowledge, the only dynamic implementation of the 3D DT is `CGAL`. It actually offers much more than just the possibility to compute the DT and delete vertices in it: many fundamental algorithms and data structures in computational geometry—in two, three and higher dimensions—have been implemented and are being used for several projects in universities and in industry. The source code of `CGAL` was written in `C++`, and can be compiled under different platforms, including Windows. The major obstacles to using under Windows programs developed under Linux is the fact that the exact arithmetic schemes do not port easily—the implementations are rather low-level and most schemes are not implemented under Windows. Fortunately, the scheme used by `CGAL`, the `Core` library (Karamcheti et al., 1999), does work under Windows.

It should be noticed that `CGAL` is not the fastest implementation for the computation of a DT in $\mathbb{R}^3$, as there are reports of implementations that outperform it, see for instance Amenta et al. (2003) and Liu and Snoeyink (2005a). Apparently, the fastest implementation is `Pyramid`[5] by Jonathan Shewchuk. It is the fastest 'overall' if the data structures used fit in main memory, and if we exclude implementations that are not robust in all cases and where assumptions about the input are made. An example of the latter is `tess3` (Liu and Snoeyink, 2005b), which was developed for constructing the 3D DT of points representing atom coordinates in proteins. For these cases, the points are usually in general position, well distributed, and have a limited precision, all of which allowed Liu and Snoeyink to use floating-point arithmetic and take some engineering decisions that speed up the construction of the DT.

The aim of this section is not to show that my implementation of the algorithms is the fastest—because it is not—but to demonstrate that it is comparable to other solutions available and can realistically be used for real-world applications. The goal of this research was to build a dynamic/kinetic spatial model based on the 3D VD/DT, and conceptual simplicity was favoured over theoretically optimal algorithms. Because of that, engineering decisions had to be taken, decisions that affected most of the operations. Examples of decisions are the simple point location solution implemented, and the use of bistellar flips. Theoretically optimal point location schemes to construct a DT, such as the one in Edelsbrunner and Shah (1996), require us to keep an additional graph that can not (or only with great difficulties) be kept up-to-date as vertices are removed or moved in the DT. Also, in my case the point location is used not only to construct the DT, but also to analyse it subsequently (e.g. for interpolating). Bistellar flips were chosen as the atomic operation to modify a DT because the movement of points is based on them (many of the functions implemented for the deletion were the same as for the

[5]It is still in beta version, but has been used by some researchers in the computational geometry community. The first description of `Pyramid` dates back to Shewchuk (1997b).

movement of a point), and because keeping at all times a complete tetrahedralization is less error-prone. I am aware that flipping to insert and delete points requires more operations than creating a hole (Liu and Snoeyink, 2005b), but I also believe that it offers a good trade-off simplicity versus running time. Therefore, for all the reasons mentioned, it is not suggested that my prototype can compete in speed with implementations of the DT that were optimised and fine-tuned with only one operation in mind.

For all the experiments described in this section, all the drawing procedures were removed from the prototype and the running time shown is the time only for the operation itself, i.e. the time to read the input file and output the results are not taken into account. Each plot of running time was made from around 50 time samples collected during the process. The data structure used is the tetrahedron-based structure, and the datasets were chosen such that all the data and the data structures to store the DT fit in main memory. The prototype was not optimised in any ways for very big datasets, such as the ones in Blandford et al. (2005) and Isenburg et al. (2006). Unless explicitly mentioned, the adaptive predicates of Shewchuk (1997a) are used for both Orient and InSphere for all the operations (which includes for instance Walk), but standard floating-point arithmetic is used for all the other operations. The `CGAL` code (release 3.1, December 2004) was compiled under Windows with `MinGW` (a port to Windows for the `gcc` compiler), and was made 'comparable' to my prototype. In other words, the predicates Orient and InSphere also use robust arithmetic but all other operations are made with floating-point arithmetic[6]. Also, all the 'checks' in `CGAL` have been disabled; these are functions verifying that the combinatorial structure of the DT stays coherent after an operation (disabling them sped up the insertion and the deletion by around 20%).

6.4.1 Datasets Used

The prototype was tested with four different datasets, which represent a variety of spatial distributions that one is likely to find, and where degeneracies are present:

50k: a set of 50 000 points randomly distributed in a unit cube;

sphere: a set of 25 000 points randomly distributed on the surface of a sphere;

cubes: a set of 15x15x15 points regularly distributed in the $x - y - z$ directions, forming a cube with 3 375 points;

ocean: a real oceanographic dataset of the Bering Sea[7]. Figure 6.8 shows the dataset, where the points are distributed along water columns. It contains many of such water columns, and the points are regularly distributed along each column (at regular intervals of depth), although this is not always the case as some data points were removed because of instrument errors. Also, notice that the water columns do not all go to the same depth.

Table 6.1 contains the details of the four datasets. The tetrahedra outside the convex hull of the dataset exist because of the big tetrahedron (see Section 4.2.1), and k is the degree of a vertex in the DT. Observe that in the case of the ocean dataset, the anisotropic distribution means that one vertex has a degree of 123, but that the number of tetrahedra is still linear.

All the coordinates $x - y - z$ of the data points are stored with double precision ('double' in Delphi and `C++`).

[6] The `CGAL` kernel `Exact_predicates_inexact_constructions_kernel` was used.

[7] Taken from the Oceanographic In-Situ Data Access: http://www.epic.noaa.gov/epic/ewb/

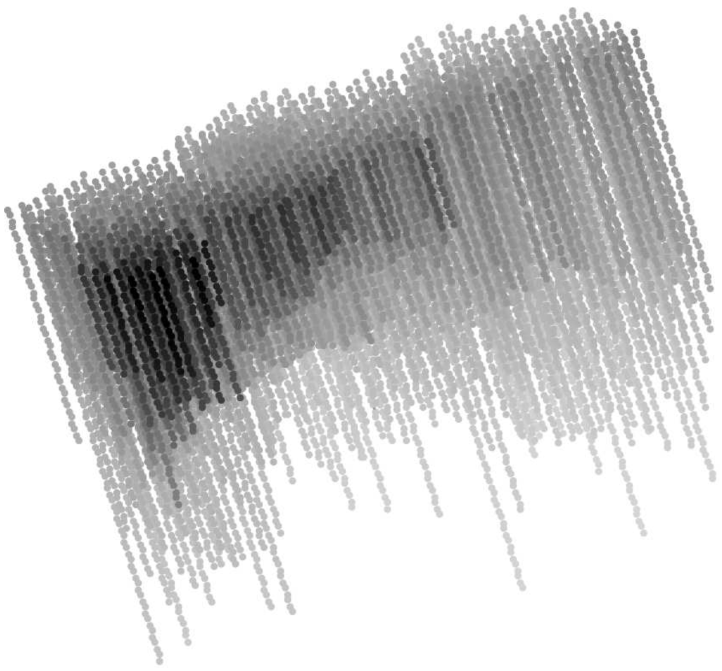


Figure 6.8: Perspective view of the ocean dataset, formed by many water columns.

	# pts	# tetra (total)	# tetra outside conv(S)	k_{min}	k_{max}	k_{avg}
50k	50 000	335 731	507	5	37	15.4
sphere	25 000	126 392	50 749	4	81	11.1
cubes	3 375	18 946	2 482	4	14	12.9
ocean	14 550	86 338	2 156	6	123	13.8

Table 6.1: Details concerning the datasets used for the experiments.

	mine (robust)	mine (float)	CGAL hierarchy
50k	10.6	7.6	9.6
sphere	20.6	—	49.5
cubes	6.6	1.1	37.9
cubes–r	1.7	0.4	23.3
ocean	16.9	—	58.1
ocean–r	5.4	—	55.8

Table 6.2: Running times (in seconds) for the construction of the DT of the datasets.

Also, the optimality of the incremental insertion algorithms is valid only under the randomness assumption, i.e. the points in the set must be inserted in a random order. This is why many implementations actually first shuffle the input, see for instance Amenta et al. (2001) and Barber et al. (1996). As Amenta et al. (2003) and Liu and Snoeyink (2005b) note, when constructing a DT (and also when deleting many points in a DT), a trade-off between inserting points close to each other (to minimise the point location time) and inserting points evenly across the space so that clusters are avoided (which creates long and skinny tetrahedra) must be sought. Clusters can increase the average degree of vertices, and create configurations where many points are coplanar and/or cospherical.

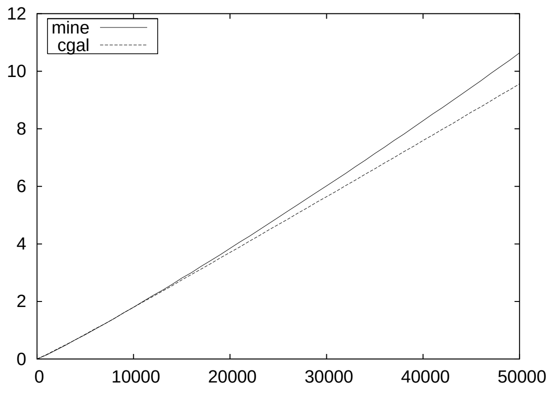
The ocean and cubes datasets have a strong *spatial coherence*: in the original ocean file, the water columns are listed one after the other, and in each water column the points are listed from the surface to the bottom of the sea; and the cube file was constructed using three imbricated loops (so the dataset is constructed ‘line by line’). For this reason, the order of the points in the two datasets were randomly shuffled to obtain two new datasets: ocean–r and cubes–r.

6.4.2 Construction of the DT

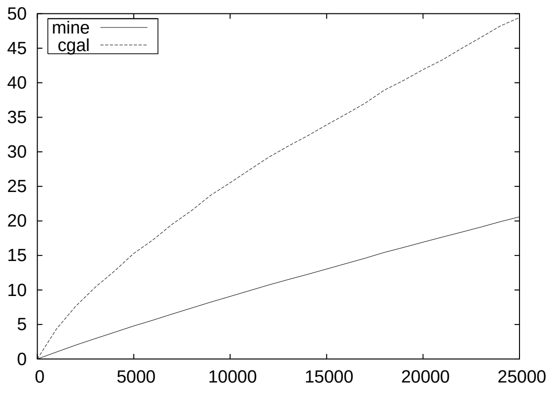
My prototype uses an incremental insertion algorithm, where each point is inserted with InsertOnePoint (Algorithm 4.3 in page 59), to construct the DT, and the point location strategy is Walk, as described in page 55.

CGAL also uses incremental insertion to construct a DT, but instead of flipping, the Bowyer-Watson algorithm is used (see Section 4.3.1). For the point location strategy, CGAL implements the Delaunay hierarchy of Devillers (2002b): for a set S of points, the first DT(S) is constructed, and then other levels are created by sampling some points in S and creating the other DTs. The tetrahedra sharing the vertices between two levels are linked together. The point location involves walking (with Walk) to the tetrahedron at the top level containing the target point, and then going down one level and continuing this way until the tetrahedron in DT(S) containing the target point is found.

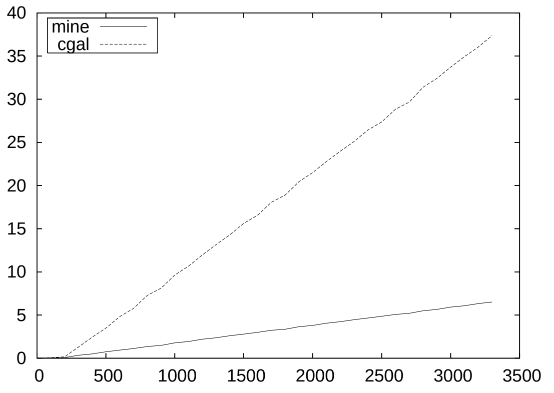
The running times of my prototype versus CGAL for the six datasets are reported in Figure 6.9 and summarised in Table 6.2. The results obtained are rather surprising as my prototype is faster for five of the datasets, and by a factor of almost 14 for the cubes–r dataset; the only dataset where my prototype is slower is for 50k, but only by 10%. An important observation is that the behaviour of my prototype is linear as the number of points grows, and that for all the six datasets. Notice also that shuffling the cubes and the ocean datasets improved the construction of the DT of my prototype by a factor of respectively 4 and 3; while with CGAL the factors were respectively 1.6 and almost no change.



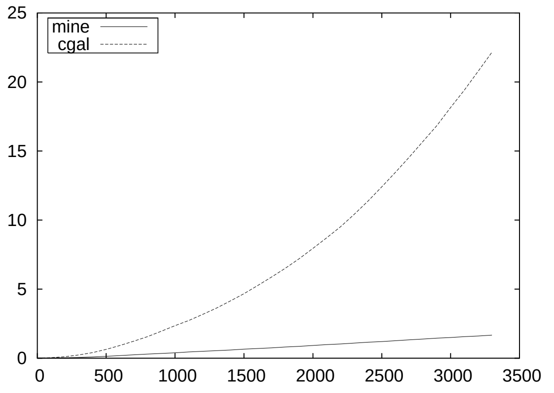
(a) 50k



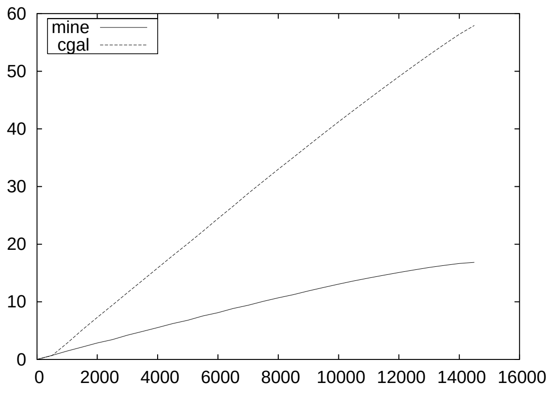
(b) sphere



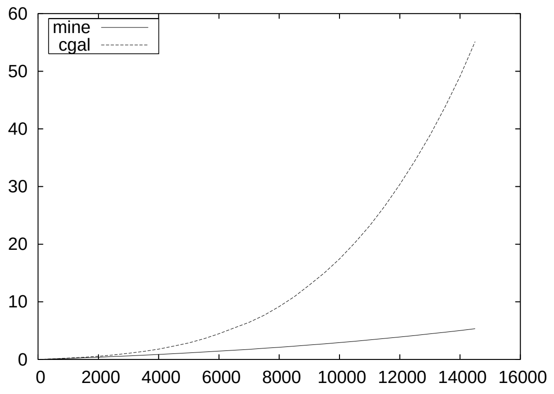
(c) cubes



(d) cubes–r



(e) ocean



(f) ocean–r

Figure 6.9: Running times for the construction of the DT for the six datasets. For all the plots, the horizontal axis is the number of points, and the vertical axis the time in seconds.

The reasons of such important differences between my prototype and CGAL are not totally clear to me because in many reports CGAL performs rather well, and can compare easily with the other implementations available. For instance, Boissonnat et al. (2002) report constructing the DT of 100 000 points randomly distributed in a cube in 12.1s, on a computer relatively slower than the one used for my experiments (a Pentium 3, 500 MHz with 512 MB of RAM). By comparison, for 100 000 points randomly distributed, CGAL takes 19.7s on the computer I used. The major differences were that they used CGAL under Linux, and also used integers to store the coordinates of the 100 000 points. The use of integers can speed up an implementation (computation with integers is faster), but unfortunately can not be used in all situations. When dealing with real-world data, the coordinates of the points are usually converted to integers by multiplying them by a constant; this can lead to problems in a dynamic context if the next points to be added are unknown, as there might be a big difference in the order of magnitude of the precision of a point, which would invalid the constant used. Integers are also limited to 4 bytes, which could be problematic for some datasets.

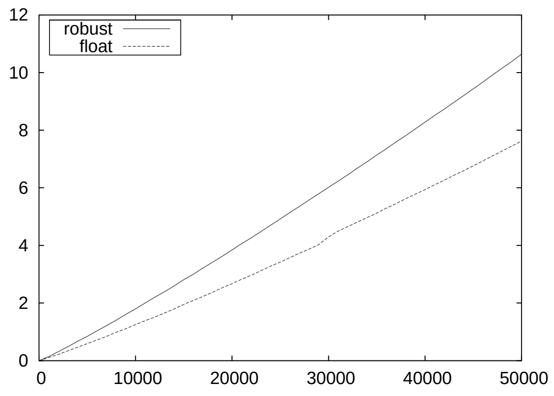
Three reasons could explain the somewhat poor results of CGAL I obtained during my experiments. Firstly, because CGAL performs very poorly for datasets containing degeneracies (even when they are shuffled), the robust arithmetic used by CGAL is probably the bottleneck. The arithmetic used in my prototype is adaptive, while CGAL uses a completely different scheme (Karamcheti et al., 1999), which might not be as fast, especially under Windows. Secondly, it seems that the compiler used, and the platform, have a big influence on the speed of the code. Indeed, Liu and Snoeyink (2005a) report having differences in CGAL's running time of a factor of as much as 2.5 when it was compiled on Linux with different kernels, and that similar differences were found between versions of CGAL. Thirdly, although it is less likely, the perturbation scheme used by CGAL could be the culprit. Since CGAL 3.0, the perturbation is indeed based on the lexicographic index of the points (and no more on the order of insertion of the points), and therefore potentially more work is required when many points are cospherical, because the unique DT must be found. By contrast, the perturbation scheme used in my prototype does not slow down the insertion process, as it assumes that the last point inserted is always outside the already existing spheres (see Section 4.4.5.3).

The speed of my prototype, with robust and with standard floating-point arithmetic, was also compared to verify the claims made in Shewchuk (1997a). The running times for each dataset are given in Table 6.2 (a dash '—' means that the program crashed), and plots for the 50k and the cubes datasets are in Figure 6.10. As expected, the DT of the datasets containing many degenerate cases could not be constructed, although, surprisingly, the cubes and cubes–r datasets could be constructed (this by no means implies that all regularly spaced datasets could have been constructed, since during other experiments many regularly-spaced datasets crashed). The construction of the DT for the 50k dataset is around 40% slower when robust arithmetic is used, and the cubes and cubes–r datasets are slower by a magnitude of respectively 6 and 4.6. These numbers corroborate the results in Shewchuk (1997a).

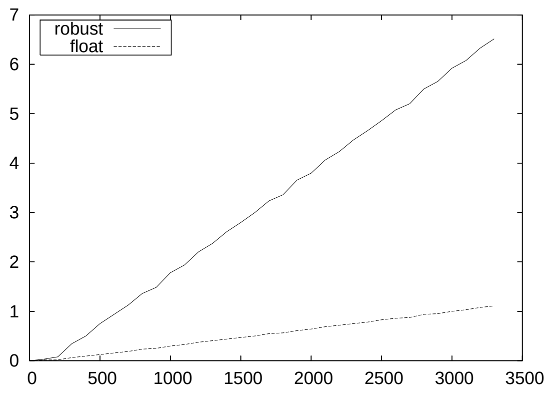
6.4.3 Deletion of a Vertex in the DT

The deletion algorithms of my prototype and of CGAL were tested by deleting sequentially the points in the six datasets (in the reverse order with which they were inserted).

For my prototype, the algorithm DeleteInSphere (Algorithm 4.5) was used, and the degenerate cases were handled with either the perturbation scheme described in Section 4.4.5.3 or



(a) 50k



(b) cubes

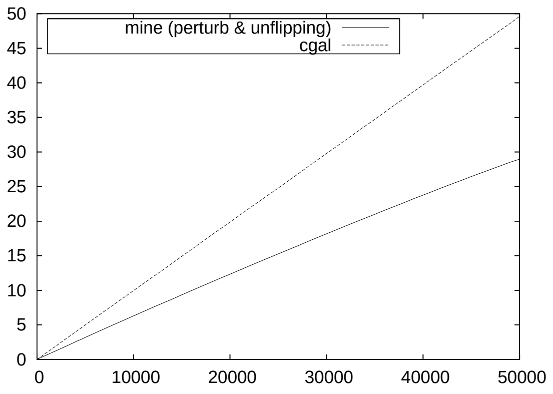
Figure 6.10: Comparison of the running times for the construction of the DT, with my algorithm, between the robust arithmetic of Shewchuk (1997a) and floating-point arithmetic. For both plots, the horizontal axis is the number of points, and the vertical axis the time in seconds.

	mine (perturb)	mine (unflipping)	# UP	mine (float)	CGAL hierarchy
50k	29.0	29.0	0	22.5	49.6
cubes	61.0	59.9	0	—	156.3
cubes_r	3.8	3.8	7	—	28.7
sphere	99.3	89.7	0	—	117.1
ocean	433.5	403.4	0	—	193.8
ocean_r	9.6	9.4	101	—	56.1

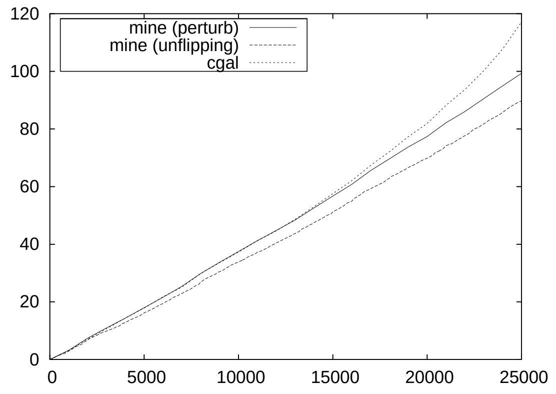
Table 6.3: Running times (in seconds) for the sequential deletion of all the points in the datasets. '# UP' is the number of untetrahedralizable polyhedra that the unflipping algorithm encountered.

the unflipping method of Section 4.4.5.4. Because a simple tetrahedron-based data structure is being used, the retrieval of a tetrahedron incident to the vertex v to be deleted is obtained with a WALK in the tetrahedralization. The deletion algorithm implemented in CGAL, described in Devillers and Teillaud (2003), is conceptually the same as DELETEINSPHERE, but a hole is created and a perturbation scheme is used to deal with degeneracies. A point location strategy is not required since CGAL keeps, for each vertex in a DT, a pointer to one of its incident tetrahedra (Boissonnat et al., 2002).

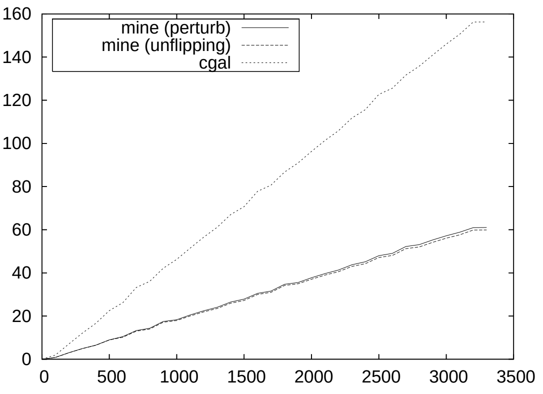
The running times of my prototype versus CGAL for the six datasets are reported in Figure 6.11 and summarised in Table 6.3. First, observe that my prototype is faster for all the datasets, except the ocean. When the points are in general position (50k dataset), my prototype is faster by a factor of 1.7; when degeneracies are present, the factor can go up to around 7 (for the cubes–r). These differences are probably due to the same reasons as for the construction of the DT. Also, the behaviour of my prototype is linear (or almost linear) for the six datasets, while CGAL is either linear or is faster towards the end of the sequential deletions. The latter can probably be explained by the fact that less work is required when a vertex has a smaller degree. Moreover, notice that the unflipping method is marginally faster than (or equal to) the symbolic perturbations method for the six datasets; for the 50k, cubes–r and ocean–r the running times were virtually the same, so only one line was drawn.



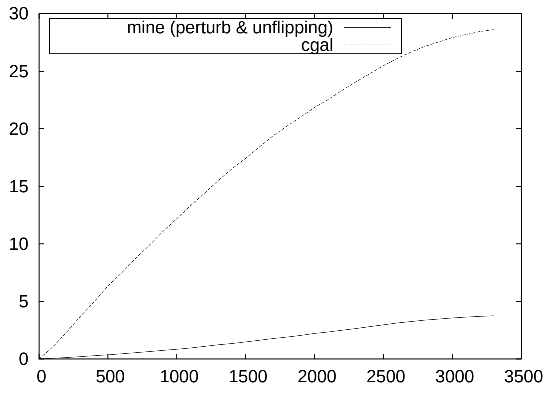
(a) 50k



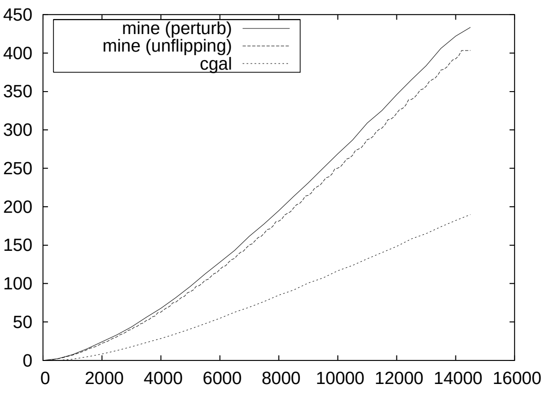
(b) sphere



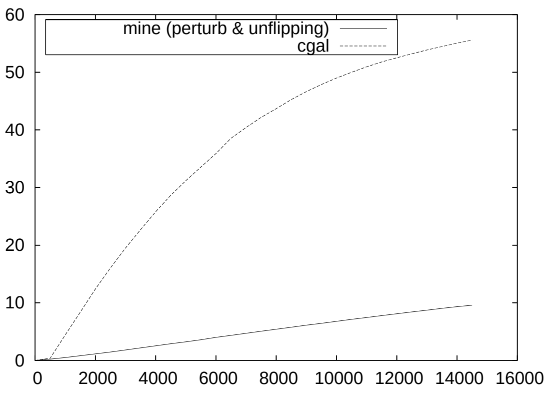
(c) cubes



(d) cubes–r



(e) ocean



(f) ocean–r

Figure 6.11: Running times for the sequential deletion of all the points in the DT for the six datasets. For all the plots, the horizontal axis is the number of points, and the vertical axis the time in seconds.

The fact that unflipping is faster than using perturbations was expected from the theory: when a perturbation scheme is used, a unique DT is sought, and that prevents some ears to be flipped, even though they could be. Theoretically, the unflipping method can take much longer for deleting a single vertex than when a perturbation scheme is used (because it is an iterative process), but in practice the cases where the star of the vertex is an untetrahedralizable polyhedron (UP) are rare, and only when the dataset contains many degeneracies. As shown in Table 6.3, only the cubes–r and the ocean–r encountered UPs during the sequential deletion of all the points, and yet the behaviour is still linear and the running times faster than `CGAL`.

Only for the sphere and ocean datasets does `CGAL` get similar or faster running times. For the former, it is only around 25% slower, but that was expected by the theory: when points are on a sphere, many ears can not be flipped so a lot of time can be spent finding a flippable ear (`CGAL` obviously avoids this as a hole is created). The deletion of all the points in the ocean dataset is about twice faster with `CGAL`, but again that was expected from the theory. Indeed, as explained in Section 4.4.5.2, when the tetrahedra forming an ear are coplanar according to two different planes (see the 'umbrella' configuration in Figure 4.17(c) in page 71), the deletion is slowed down because the star of the vertex to be deleted must remain a 'simple polyhedron' at all times, and many flips are 'held' before they can be performed. As the experiment shows, this slows down the deletion (because most of the deletions in the ocean dataset will have this 'umbrella' configuration), but when the dataset is randomly shuffled, this is not that problematic anymore for it is faster than `CGAL`.

The experiment was also tested with floating-point arithmetic, and, as shown in Table 6.3, the program crashed for all the datasets but 50k, for which the deletion was sped up by around 25%.

6.4.4 Natural Neighbour Interpolation

An experiment was also made to test the speed of the natural neighbour interpolation algorithm, InterpolNN, as presented in Section 5.2.5.4. As mentioned in that section, not many algorithms to compute the natural neighbour coordinates exist in three dimensions, and only Boissonnat and Cazals (2002) report the practical performance of their code. Their method is supposed to be added to `CGAL`, but is still not available[8]. It is not really possible to compare the running times they obtained, since they used different computers, operating systems and representation of coordinates (they used integers).

I therefore report in this section an experiment that compares the two-dimensional version of InterpolNN with the most popular algorithm: the mimicking algorithm presented in Watson (1992) (the method of Boissonnat and Cazals (2002) is a generalisation to three dimensions of this algorithm). Both algorithms were implemented with Delphi, and they use the same point location strategy, which is the two-dimensional counterpart of Walk. To implement InterpolNN, an incremental insertion algorithm based on flip22 was implemented, and the interpolation point is deleting by rolling back the flips.

I ran the experiment with two different datasets: (1) a set of 10 000 points randomly distributed in a square; (2) a portion of a map where the elevation is represented with isolines, which have been converted to points (see Figure 6.12). This dataset is akin to the water columns dataset, as the distribution is very anisotropic and formed by lines. The experiment consists of creating

[8]In `CGAL` 3.1, the declaration of the function is there, but the source code is still missing.

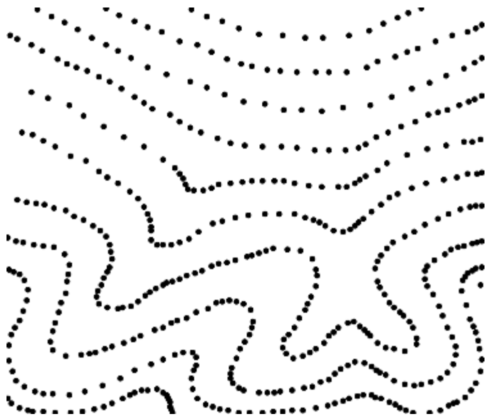
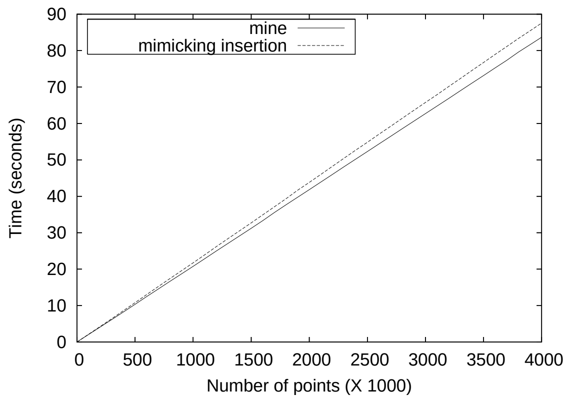
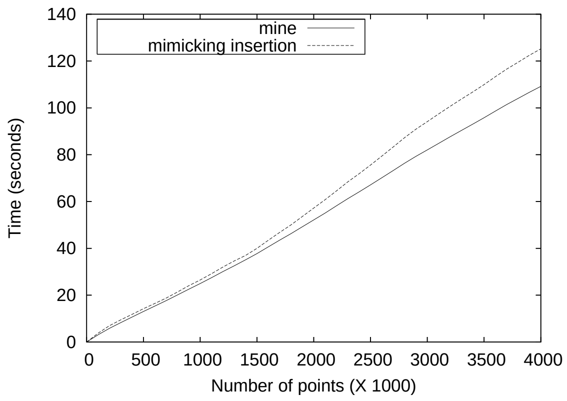


Figure 6.12: The dataset, created from contour lines, used for the interpolation experiment.



(a) 10 000 points randomly distributed in a square.



(b) Dataset of contour lines.

Figure 6.13: Running times for the creation of a 2000x2000 grid with natural neighbour interpolation for two different datasets in two dimensions.

a 2000x2000 grid from these datasets, i.e. 4 millions interpolations are performed at regular locations. Figure 6.13 shows the running time for both datasets. Both algorithms have a linear behaviour and take almost the same time to perform the operation; InterpolNN has a slight advantages in both cases (5% for the randomly distributed points, and 15% for the isolines).

6.4.5 Extraction of Isosurfaces

As mentioned in Section 5.4.1, no additional data structures were used to help with the extraction of isosurfaces. When an isosurface is needed, all the tetrahedra are simple visited and triangular faces are extracted from the intersected tetrahedra. Although this is not theoretically optimal, this simple algorithm yields, in my opinion, satisfactory results.

For instance, when an isosurface is extracted from the ocean dataset, the running time is on average only 0.4s. The time depends on the number of tetrahedra intersected (which itself depends on the value of the isosurface): when it is around 12% of the total, the running time is 0.36s, while when it is around 22%, it is 0.43s. This time is for the traversal of the whole

DT (with TRAVERSE, see page 51) and the extraction of the triangular; the drawing time is not taken into account.

Another experiment was made with the 50k dataset, to which attributes were attached: each point was assigned to a random value between 0 and 5. The extraction of an isosurface takes around 3s: 1.9s when 35% of the 335 731 tetrahedra intersect the isosurface, and 3.6s when 87% of the tetrahedra are intersected.

Chapter 7

Applications in Geoscience

A spatial model based on the Voronoi diagram and the Delaunay tetrahedralization, with the manipulation and analytical functions described in Chapters 4 and 5, is a generic tool that can be used in a wide variety of domains and applications. Indeed, any applications where trivariate fields—represented by their samples—are involved can benefit from the properties of the VD and the DT. Moreover, many application domains where the datasets are point-based can also be modelled and analysed with the VD/DT. The major impediment to the use of the VD/DT to model three-dimensional geoscientific datasets is that faults and/or discontinuities in the data and phenomena are not allowed. As a consequence, geological applications can not use the spatial model directly, although some have used tetrahedralizations for specific problems in geology, see for instance Lattuada (1998) and Xue et al. (2004).

As it was mentioned in the Introduction, most scientific papers related to the use of GIS to model and analyse three-dimensional geoscientific datasets focus on methods to circumvent the static and two-dimensional structures of traditional systems. The analytical methods they present are in two dimensions, as they almost in every case slice the datasets and analyse separately each slice. Examples of such papers are plenty in the surveys of Chapman and Thornes (2003), Sui and Maggio (1999) and Valavanis (2002) concerning the use of GIS in the geosciences. Consequently, discussions concerning 'real' three-dimensional modelling of datasets, and of three-dimensional spatial analysis operations, are scarce in the literature.

This chapter presents some potential applications where the properties of the VD/DT are useful to model fields, both continuous and discrete. Firstly, a few generic applications are presented, and then examples in oceanography, in meteorology and in the study of geological microstructures are presented.

7.1 Generic Applications

7.1.1 Simulations

As briefly discussed in Section 2.6.4, different techniques can be used to perform the simulation of a real-world process. Most simulations of processes in the ocean (e.g. tracking of pollution plumes) and in the atmosphere (e.g. dispersion models) are made with the finite difference method (FDM) on voxel structures. The FDM performed on grids, as used by systems for weather forecasting, is well-known, efficient and mostly accurate. However, the use of grids can sometimes lead to unreliable results (Augenbaum, 1985), and some other technical problems also arise (for instance the curvature of the Earth is problematic for large datasets).

An interesting alternative to FDM is the Free-Lagrange method (FLM) (Fritts et al., 1985). With this method, the flow being simulated is approximated by a set of points (called particles) that are allowed to move freely and interact, and a tessellation of space is kept up-to-date as points are moving (this is fundamental in order to discretise the continuous flow). Each point has a mass and a velocity. Any tessellation is theoretically possible, but the VD has desirable properties because of the shapes of its cells, and also because, as points are moving, the changes in the Voronoi cells are 'smooth'. In other words, topological events arise gradually by the addition/deletion of a facet to a cell (Fritts et al., 1985); by contrast, if a triangulation is used to represent the flow, the bistellar flips cause abrupt changes. Mostafavi and Gold (2004) propose using the VD for the FLM because of the shapes of the cells obtained, but also because of the kinetic properties of the VD. Indeed, earlier implementations of the FLM were very slow because the adjacency relationships between cells had to be rebuilt at each step of the process. With 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 show the advantages of the kinetic VD with the simulation of global tides on the Earth (thus using the VD on a sphere).

The FLM based on the VD could obviously be used 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 results obtained in this thesis have already been used, by Dr Mostafavi at the Université Laval, Québec City, Canada (Mostafavi, 2006), 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.

7.1.2 Real-time Applications

The term 'real-time' refers to using a GIS as data are collected. According to Hatcher and Maher (1999), when used in a marine context, two concepts are involved: when data related to the position of the ship (from a GPS receiver onboard for example) are directly put into the GIS and the position of the boat can be seen according to surrounding objects; and when data are collected, quickly processed and then added to the GIS. The first concept is related to the problems of robot navigation of boats and collision avoidance, while the second permits us to obtain directly at sea a first check of the quality of a survey, and to correct mistakes or collect new data when some are missing. This ensures that a boat will not have to return to a site for further surveys, for oceanographic surveys are very expensive. The concepts are obviously valid for other applications where data are collected by devices.

Because of the dynamic and kinetic properties of the VD, a VD-based GIS could integrate in real-time new data that were just collected, and analyse them at once because the spatial relationships of the objects in a dataset are guaranteed to be valid at all times. No *a priori* knowledge of the complete dataset would be require to view the datasets and query them, as is the case if one wants to use raster and the Kriging interpolation method to construct the grids.

7.1.3 Temporal Data / History Maintenance

The VD permits insertion, deletion and movement of points with local modifications only, thus every operation is reversible. As shown in Gold (1996) and Mioc (2002), by simply keeping a 'log file' of every operation done it is possible to rebuild each 'topological state' of a VD, at any time. This represents one way of solving the problems GISs have with temporal data. There is no need to keep various 'snapshots' of the data at different times for further analysis: when a representation of a field at a specific time is required, it is reconstructed from the original data and from the log file. A dynamic field can also be viewed like a 'movie' of the changes that occurred during a certain period of time, provided of course that such information is available.

7.2 Oceanography

The most common type of volumetric data in the marine environment is CTD data, which is the abbreviated name for data collected with an instrument having sensors for measuring **c**onductivity, **t**emperature and **d**epth of the water (Nichols et al., 2003). Additional properties of the water (e.g. salinity, fluorescence, particle concentration and density) can also be measured when other sensors are mounted on the CTD instrument. The traditional sampling technique in oceanography is the vertical CTD cast: the instrument is lowered from the boat through the water column, and then raised back to the ship. A three-dimensional representation of the water is built with many water columns; resulting datasets therefore have a highly anisotropic distribution.

Volumetric information about the seawater can also be collected with remotely operated vehicles (ROVs) and autonomous underwater vehicle (AUVs) (Nichols et al., 2003). Both are submersible robots that can be positioned with inertial systems and on which CTD-typed sensors can be fixed (George et al., 2002). ROVs are linked to the ship by a tether, which permits an operator to control it and received the datasets in real-time. AUVs are totally autonomous and can operate for weeks, transmitting their data by satellite when they periodically reach the surface of the water. The samples collected with these instruments will obviously also have an anisotropic distribution.

Sonar is also a technology being used to conduct underwater surveys. The technique uses the propagation of sound waves: a pulse of sound is created, and when it hits an object the reflection of the wave goes back to the sensor; the distance sensor-object can be measured, along with other properties of the objects. Notice that when such a sonar is used, datasets also have a highly anisotropic distribution because samples are collected according to the ship's track. To circumvent this, multibeam sonars have been developed in recent years. They permit us to send simultaneously many sound waves across a transversal swath (with perhaps 60 degrees off the nadir); an almost complete representation of a scene can therefore be collected. They have mostly been used to map the seafloor, but it is also possible to fix such a sonar to AUVs/ROVs to scan more accurately the seafloor or objects that lay on it (cables, pipelines, wrecks, etc.). Even with multibeam sonars, interpolation techniques are still required because, as Emery

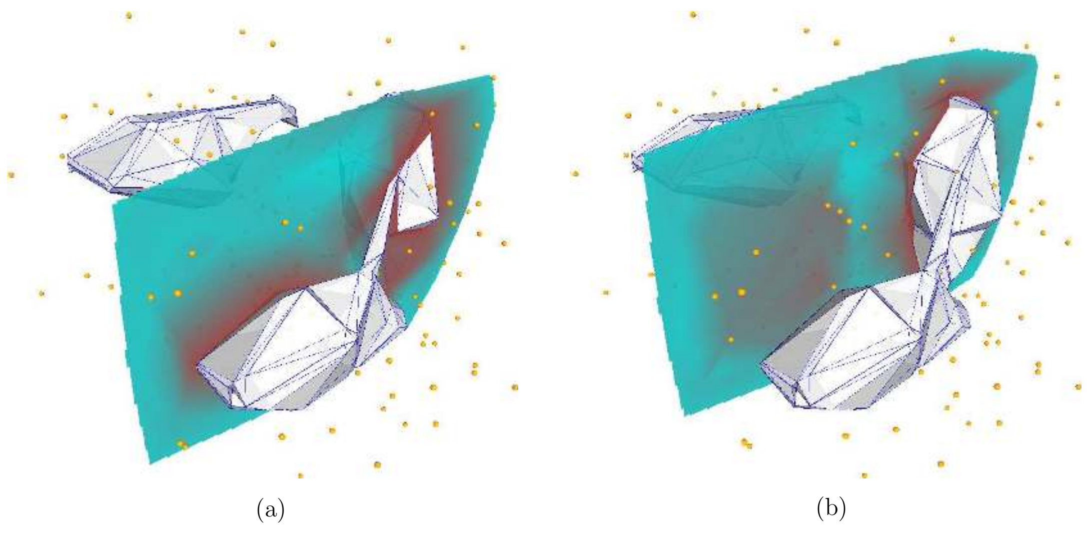


Figure 7.1: Two slices taken at different locations in the same field (red indicates a high value). For both cases, an isosurface for a high value is also shown.

and Thomson (2001) note, "oceanographic research vessels are expensive platforms to operate and must be used in an optimal fashion. As a consequence, it is often impossible to collect observations in time or space of sufficient regularity and spacing to resolve the phenomenon of interest. Efforts are usually made to space measurements as evenly as possible but, for a variety of reasons, station spacings are often considerably greater than desired".

7.2.1 Visualisation of Sea Attributes

Tools that help with the visualisation, as described in Section 5.4, can be combined, and with the help of new techniques developed in recent years in computer graphics, it is possible for instance to draw many isosurfaces and view them all by using 'transparency' techniques, assigning different colours to each, 'peeling off' surfaces and navigating inside and outside to see the shape. For instance, Figure 7.1 shows one case where a cutting plane and an isosurface for a high value of the attribute are displayed. With the help of such tools and visualisation tools to 'navigate' in the scene, a user can understand better the spatial variation of the attribute being studied.

Head et al. (1997) describes several visualisation operations in the context of oceanographic datasets represented with grids, but the same operations are possible and even optimised when the VD/DT is used, as described in the previous chapter.

7.2.2 Study of Upwellings

Upwelling is the "upward movement of cool and nutrient-rich sub-surface waters towards the surface often leading to exceptionally rich areas"[1]. Understanding upwelling is important because 50% of the world's fisheries are found in the upwelling areas, which are only 0.1% of the ocean surface area (Valavanis, 2002). Su and Sheng (1999) studied this process in the

[1]Definition of the Food and Agriculture Organization of the United Nations's glossary (`www.fao.org`).

Monterey Bay in USA by visualising the water properties (CTD data for temperature, salinity and density) over a certain period of time. Different isosurfaces, for each property, were created and then animated; the location and pattern of these isosurfaces helped scientists to determine upwelling characteristics.

They first converted their data to a regular grid, but with a spatial model based on the VD/DT, isosurfaces could be extracted directly from the DT, and no grids would have to be built. Also, if the processes involved were formalised, it could be possible not only to look at datasets collected at different times, but also to simulate the upwelling process and study the shape of isosurfaces, as points are moving over time. This three-dimensional simulation could also be visualised, and stopped at any time to query the data.

7.2.3 Fisheries

While most researchers recognise the benefits of using traditional GISs for fisheries-related applications (Meaden (1999), Meaden (2000) and Valavanis (2002) give numerous examples of applications), they also argue that since fisheries phenomena are dynamic and three-dimensional, so should be the systems for managing and analysing them.

One of the technologies that has helped the most the collection of fisheries-related data is the multibeam sonar because, in recent years, it has been improved and can now record midwater returns. In other words, any object in the water column (e.g. a fish school, or large quantities of seaweed) can now be detected, instead of only bathymetric and seafloor-type data (Hammerstad, 1995). Multibeam datasets are formed by points with three-dimensional coordinates, and some attributes can be attached to each point. These attributes are obtained from parameters returned by the sound wave, such as the backscatter energy, and can help to identify the types of objects; it is for instance possible to extract the type of fish or the density of seaweed in the area (Mayer et al., 2002). While such datasets do not represent a field, the same methods for the extraction of information and the visualisation described in Chapters 5 can be used.

Assessing the fish population and its abundance with multibeam dataset is a complicated task, firstly because the raw data must be processed and analysed to detect the presence of fish schools, and secondly because of the quantity of information that multibeam sonars can collect, which can be up to 400 Mbytes per hour (Mayer et al., 2002). While different techniques to process the raw multibeam data have been proposed (e.g. see Gerlotto et al. (1999) and Mayer et al. (2002)), very little work has been published on the postprocessing of the information, in order to extract information from the processed datasets. Buelens et al. (2005) propose using data mining techniques to identify and assess the size of fish schools from very large datasets. They identify clusters, which represent fish schools, based on the backscatter energy of each point, and use Delaunay tetrahedralization to group different points into a single objects, as Figure 7.2 shows. They are also using alpha shapes to obtain similar results, without the use of the backscatter energy. But, as Buelens (2005) points out himself, the VD could also help in extracting information: “At the moment I am using 3D Delaunay triangulations with some thresholds (like alpha shapes). However, I think Voronoi cells may actually be more accurate because they place the points in the cells rather than at vertices”.

Examples of potential applications of GIS in fisheries are also given by Meaden (2000). Firstly, he states that GISs are useful to identify the relationships between marine populations and environmental variables (seabed type, temperature or salinity of water, seaweed type, etc.), and secondly, the modelling of fish activity and movement (to predict where fish schools will

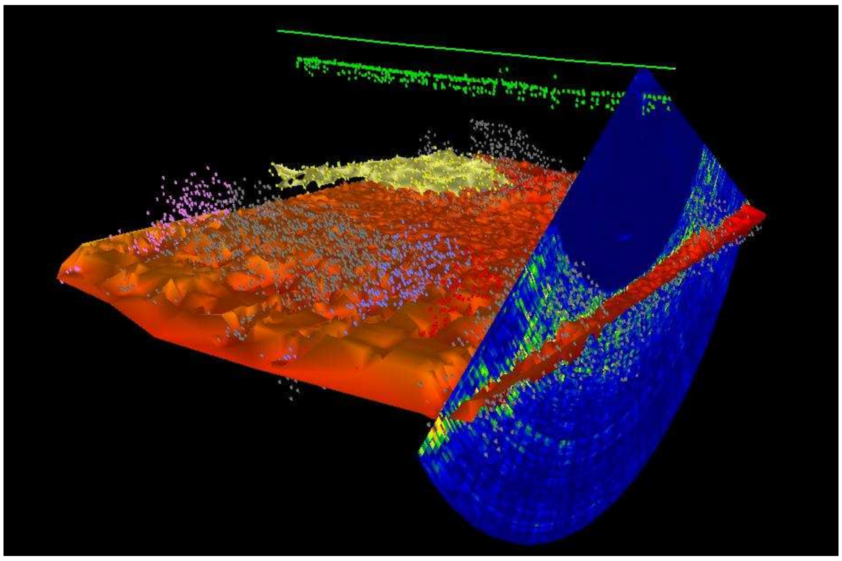


Figure 7.2: Example of processed multibeam data. The orange surface represent the seabed, while the cluster of yellow points represent a fish school. (Figure obtained from Buelens (2005))

migrate) could also be done, provided that a model is available and that a dynamic/kinetic GIS is available.

7.2.4 Biogeography

Biogeography is the science that deals with the spatial distribution of species. It helps us understand where animals and plants live, and tries to understand why they live there. Its link with GIS is obvious. As is the case for many GIS applications related to the ocean, most biogeography projects use commercial two-dimensional GISs, see for instance Schick (2002). Until recently, the use of traditional GISs were not really an issue because methods to track a mammal underwater were not available, and one had to wait until the mammal surfaced to get a position with the GPS. But it is now possible to track the movements of mammals underwater, provided that they have been tagged with devices. Oliver (1995) briefly presents the technology used, and describes the kind of insights he gains from a program where the path of the mammal can be visualised, and combined with other datasets. Analysing the temperature that the animal encountered, and why it went to a specific depth before resurfacing can help predict its future behaviour.

The VD/DT could help for such an application because, first of all, the volumetric data would not need to be gridded (as it is usually done), and secondly, the mammal could simply 'navigate' within the field (by using the adjacency relationships between the tetrahedra, similar to the WALK algorithm in page 55) and use for instance linear interpolation on the faces of the tetrahedron that is being traversed (or natural neighbour interpolation at predefined location along the path of the mammal) to estimate what temperature/salinity the mammal encounters.

7.3 Meteorology

Datasets collected to study the properties of the atmosphere are closely related to those in oceanography. Although collecting data in the atmosphere is somewhat simpler than underwater, obtaining a complete representation of the atmosphere is still difficult. The vast majority of datasets in meteorology have a highly anisotropic distribution similar to that in oceanographic datasets. Only NEXRAD[2], a network of radars capable of measuring precipitation and wind, can produce an almost complete representation of the atmosphere because the radar waves can be scattered in all directions.

Betancourt (2004) surveys the different three-dimensional atmospheric data, which are as follows. Common measurements such the air temperature, pressure and humidity—which are used for weather forecasting—can be collected with disposable instruments attached to balloons either launched from the ground or dropped from an airplane. The measurements are recorded at regular time intervals along the route of the balloon, which will obviously not be completely vertical because of the winds (the position of the balloon is obtained with a GPS). The same instruments can also be attached to aircraft. Moreover, the LIDAR[3] technology, where the distance between the instrument and an object is determine with laser pulses, is used in meteorology to measure the concentration of chemicals (e.g. ozone or pollutants): the instrument is fixed to the ground and points in the vertical direction, so that samples of the concentration of chemicals are collected all along the vertical direction. Similar instruments, called wind profilers, use radar to measure the motion of the atmosphere in the vertical direction. In all cases, the resulting distribution of samples is along the trajectory of the instrument, which is akin to the water columns.

As is the case in oceanography, three-dimensional datasets are usually broken into many two-dimensional datasets to be analysed and visualised (Chapman and Thornes, 2003), or converted to voxels (Bernard et al., 1998; Habermann et al., 2004; Nativi et al., 2004). A spatial model based on the VD/DT to represent and analyse atmospheric data would basically bring the same advantages as in oceanography: the gridding of the samples can be omitted and the data can be analysed directly. The dynamic/kinetic properties of the spatial model are particularly useful here as the atmosphere tends to change quickly over a short period of time; the analysis and weather forecasting must usually be performed quickly.

A early concrete example of application where the VD would be beneficial is given in Barjenbruch et al. (2002). They claim that the study of the trajectories of air parcels is crucial in the weather forecasting process. These trajectories are computed from scientific models, and their visualisation and analysis can inform the operator about significant coming events, such as heavy rain. The visualisation of parcels has traditionally been done in two dimensions, but they show a computer program where the trajectories are viewed in a three-dimensional environment. They also state that understanding where and why the parcel travelled is important (with respect to the attributes of the atmosphere, such as temperature or humidity) so they colour the trajectories based on certain attributes. It is important to state here that all their datasets are first converted to grids. The air parcels moving in the atmosphere are conceptually the same as the mammal moving underwater, and therefore the methods explained in Section 7.2.4 with the VD/DT could be also used.

[2]Next-generation radar, see `http://www.srh.noaa.gov/srh/sod/radar/radinfo/radinfo.html` for a description of the technology.

[3]Light Detection and Ranging.

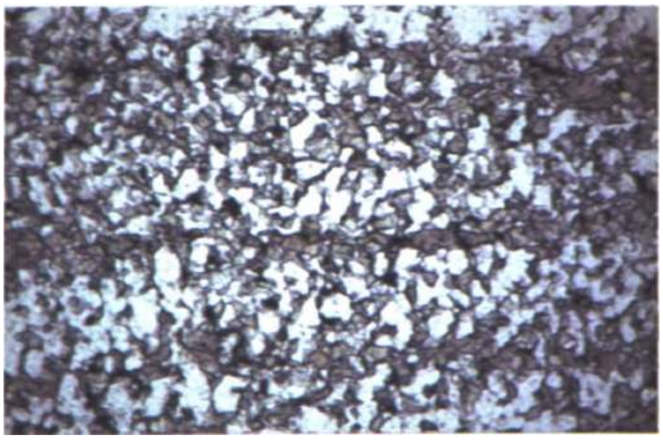


Figure 7.3: A polyphase crystalline rock.

Another example of a meteorological application is presented in Ciolli et al. (2004): the simulation of the temperature and the wind strength near the surface of the Earth, for instance in a valley. That project is actually a good example of the integration of environmental models and GIS. The model in this case is local, i.e. the attributes of the air and the wind are obtained from the slope of the terrain (which is calculated from a digital terrain model) and solar radiation at different times of the day. Ciolli et al. used GRASS to build their model, and output 3D grids of the attributes. The grids at different times can be visualised, and also analysed; they use for example three-dimensional map algebra functions to derive grids of some attributes of the air, and they also extract isosurfaces. In a subsequent paper, they admit themselves that grids have shortcomings: "the (3D) raster approach implies that, while it is possible to choose the resolution along the three axes, the variables must be estimated over the whole domain. Since heavy geometric calculation is involved [...] this can be needless burdensome when the values in only a small set of points are needed" (Vitti et al., 2004). They propose instead a variant of their method where only scattered points are output by the model, using the new 3D vector capabilities of GRASS. However, they fail to explain how the fields could be reconstructed from the data points created. To visualise and analyse the results, they output the points at regular intervals, in fact recreating the grid they wanted to avoid in the first place. The VD/DT could help solve this problem.

7.4 Geological Microstructures

Although the spatial model proposed in this thesis can not be used directly for all the geological applications (because of overfolds and discontinuities), some geology-related problems can clearly benefit from the VD. One of them is the study of the processes of deformation in rocks, which is usually done at the micro-structure level. I describe in the following an ongoing collaborative project with James Mackenzie and Prof. Dr René Heilbronner from the Department of Geosciences, Basel University, Switzerland.

The aim of the research project is to study the deformation mechanisms in crystalline rocks, which are formed by different 'phases', i.e. a rock is an amalgam of different rocks, as in Figure 7.3. Crystalline rocks can therefore be seen as discrete fields, where a value is assigned to each phase in the rock. The claim of Mackenzie et al. (2006) is that the spatial distribution of phases in a rock (the adjacency relationships between phases, and the fact that phases are clustered or not) is a robust indicator of the deformation mechanisms in the rock, because the usual geological processes are not likely to destroy them. Unfortunately, there is a major obstacle to the study of the distribution of phases in three dimensions: only two-dimensional thin sections of the rock (sliced through the rock) can be collected on-site.

This problem has been tackled in different ways. Kretz (1969) quantifies the spatial distribution of two phases by comparing the boundary fractions (with two phases A and B there will be three types of boundaries between phases: AA, BB and AB) on the thin sections with the expected values if the two phases were randomly distributed in three dimensions. With statistical tests, he can assess if the two phases are clustered or not. His work is based on the fact that the area of the surface boundary between two phases can be estimated to be equal to the length of the boundary between the same two phases on a two-dimensional slice (Underwood, 1970). Also, Jerram et al. (1996) build a three-dimensional model of a rock, where the different parts of the rock are represented by spheres (of the same size), which serves as a reference to compare the pattern obtained in the thin sections. This approach has the major shortcoming of not creating a space-filling model, and thus adjacency relationships between phases can not be studied.

The idea of Mackenzie and Heilbronner is to build a computer generated model of the rock in three dimensions, and to offer the possibility to vary the ratio of phases, and their spatial distribution in the volume. Then, two-dimensional slices from the model could be obtained, and they would serve as reference when they are compared to the thin sections of real rocks.

This is where the Voronoi diagram is useful: it produces a space-filling model, two or more phases can easily be created (simply by assigning a value to the Voronoi cells), and, as seen in Section 5.4.2, slices through a VD are easily and efficiently performed. Some functions were added to my prototype to help for such an application. Firstly, it is possible to slice according to any direction the dataset, and visualise the result directly on the screen. It was found that the possibility to view which Voronoi cells are sliced was useful to understand and assess the results; Figure 7.4 shows one slice, one Voronoi cell that was sliced, and the data points whose Voronoi cells were sliced (the ones in green). The prototype also offers the possibility to view many slices and save them as bitmaps: standard image processing software can then be used to calculate the area of phases and the length of the contact boundaries (see Figure 7.5). Notice that the colours on the slides are not meaningful as in that case only two phases were used. I used different colours for each Voronoi cell sliced (so that the user can see the contact boundaries between phases), and the phase (A or B) is represented by the 'red value' of the pixel (i.e. if the number used to code the red value of a pixel (number between 0 and 255) is an even number, then it is phase A, and if it is odd, then it is phase B). This way, the user has access to both the contact boundaries and the value of a phase in the two-dimensional slices. Secondly, functions to calculate the contact boundaries and the volume occupied by each phase in three dimensions were also added to the prototype, to be able to compare them with the ones obtained on the slice. As explained in Section 5.2.5.4 (for computing the natural neighbour coordinates), volumes of Voronoi cells and areas of Voronoi faces are readily computed because they can be decomposed into simplices whose volume/area is simply the determinant of a matrix.

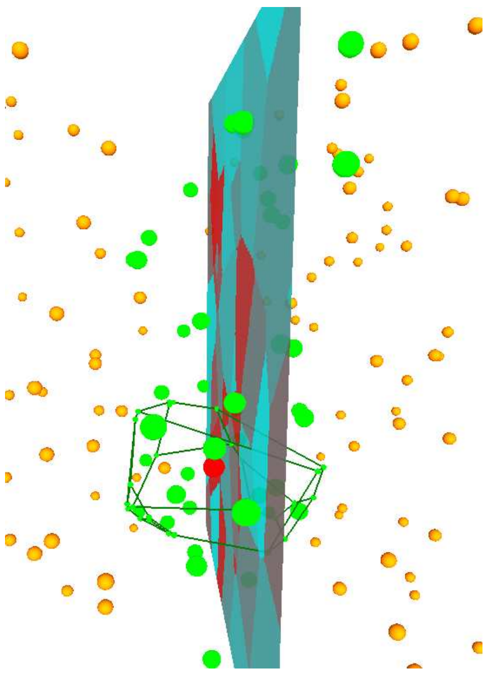


Figure 7.4: Visualisation of one slice in a discrete field, combined with the display of one Voronoi cell that is intersected by the slice. The green data points are the ones whose Voronoi cells are sliced.

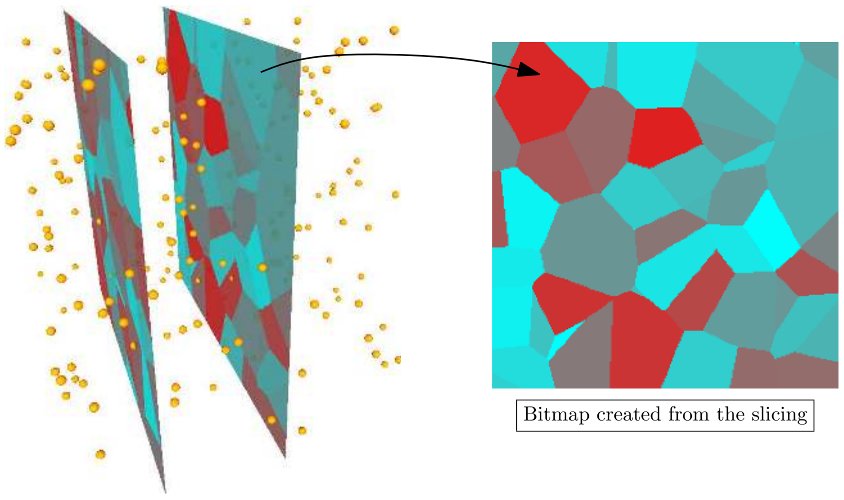


Figure 7.5: Slices can be viewed in the 3D environment, and also saved as bitmaps.

Chapter 8

Conclusions

The choice of a spatial model to represent geographical phenomena is a crucial one that is more than often overlooked by GIS practitioners. As Goodchild (1992b) states: "there exists a multiplicity of possible conceptual data models [spatial models] for spatial data, and the choice between them for a given phenomenon is one of the more fundamental issues of spatial data handling". For a given task, practitioners tend to simply use the spatial models and data structures available in commercial GISs, without assessing the consequences of their choice.

The representation of geoscientific fields in three dimensions is particularly problematic because, regardless of a few notable models developed in academia, the only solutions available are voxel-based models. The popularity of voxel structures is probably due to their conceptual simplicity and to the fact that they are easily and naturally stored in computers. However, as argued in this thesis, voxels have also many shortcomings, both conceptually and technically. Raster structures tessellate arbitrarily the space without taking into consideration the objects embedded in that space (or the phenomenon being studied), which yields an unnatural discretisation. On the technical side, rasters do not scale well and are not rotationally invariant, the size of a grid can become enormous if a fine resolution is wanted, and operations involving the combination of many grids usually mean resampling them, to ensure that every voxel in one grid 'corresponds' to one and only one in the other grids. Each resampling degrades the information stored in the file, and can lead to errors and misinterpretations in the analysis.

8.1 Advantages of a Spatial Model Based on the VD/DT

I have presented in this thesis a new spatial model to represent trivariate fields as found in the geosciences, i.e. the datasets collected to study fields are point-based. Notice that when a field is already represented with a given tessellation, it is nevertheless always possible to extract points from it. The new spatial model is based on the three-dimensional Voronoi diagram (VD) and its dual the Delaunay tetrahedralization (DT). As I have argued throughout this thesis, the tessellations obtained with the VD and the DT have many advantages over other tessellations, particularly over the widely used raster structures (regular grids and octrees). What follows is a summary of the main advantages of using a Voronoi-based spatial model:

1. the VD offers a natural discretisation of space, which is based on the samples that were collected to study the field. The phenomena studied can thus be represented freely, and not enforced by a rigid structure like voxels. The shape and the size of the cells in the VD are adaptive to the spatial distribution of the samples, which is crucial when dealing with highly anisotropic distributions such as the ones found in the geosciences.

2. the one-to-one mapping between the points and the Voronoi cells ensure that the original samples—the meta-field according to Kemp and Vckovski (1998)—are kept and not 'lost', as is the case when gridding. Moreover, the adjacency relationships between the cells define the spatial relationships between unconnected points in space.

3. the reconstruction of a field, a trivariate function, can be done automatically and efficiently with natural neighbour interpolation (for continuous fields) and with nearest neighbour interpolation (for discrete fields). Both methods are entirely based on the VD. In short, no *a priori* knowledge of a dataset or user-defined parameters are necessary, as is the case with other popular methods such as distance-based methods, or Kriging.

4. the knowledge of the VD implies the knowledge of its dual graph the DT, which helps greatly for manipulation operations, and also for some spatial analysis operations such as the extraction of isosurfaces.

5. the construction of the VD/DT for a set of samples is an automatic and efficient process. The storage of the spatial model in a database is also extremely simple for only the samples can be stored. The reconstruction of the VD can also be made with a subset of the samples if only a certain part of the dataset is of interest, or even with for instance 30% of the samples to give a crude approximation of the field.

6. local updates to the spatial model are possible, i.e. insertions, deletions or movements of points in the VD can be made without recomputing the structure from scratch. This is fundamental for the interactive exploration of a dataset, and to model dynamic fields.

7. many GIS and spatial analysis operations in three dimensions are possible and even optimised when the VD/DT of a set of points is constructed. Chapter 5 gave several examples of these operations. Moreover, when many fields need to be combined together to extract information, there is no need to resample the fields, as map algebra functions can be performed directly on the VD.

Admittedly, the algorithms to construct, manipulate and analyse three-dimensional fields represented with the VD are more complex than the ones for raster structures, but, as I have demonstrated in this thesis, they are available, efficient in practice, and can be readily implemented even in the presence of degeneracies. I believe that the benefits of a Voronoi-based spatial model compensate largely for the increased complexity during the implementation.

The usefulness of the VD was highlighted by several examples of potential applications in the geosciences. Moreover, the results of this research and the prototype GIS are already being used in two different projects: a team at the Basel University uses it to study the deformation mechanisms in geological structures (Mackenzie et al., 2006); and a team at the Université Laval has started using it for the simulation of underground water. I am confident that other applications of the tools developed will appear in the future.

8.2 Summary of Contributions

The major contribution of this research is the description of a new spatial model to represent and analyse trivariate fields in geoscience. The spatial model is a viable alternative to using three-dimensional raster structures, viable in the sense that it is not only a concept, but a solution that can be implemented and used for real-world applications. Although the 3D VD has been proposed before as a spatial model (Gold and Edwards, 1992), and although it has been used in other disciplines (see for instance Kim et al. (2004) in chemistry, and Dey and Goswami (2003) for the reconstruction of surfaces), this research contains, to my knowledge, the first detailed description of its use as a spatial model, a description that goes beyond a simple theoretical discussion. The spatial model proposed permits us not only to represent trivariate fields, but also to interactively explore and analyse them with operations that are all based on the properties of the VD and the DT.

The other contributions of this research, without which the major contribution would not have been possible, include:

1. I have shown that the 3D VD/DT can be constructed and manipulated, to become a dynamic and kinetic structure, with the same set of atomic operations—the bistellar flips—and that the practical performance of the resulting algorithms are still satisfactory, despite the fact that flips are theoretically slower. Flip-based algorithms have the main advantages of being numerically more robust, and they are also easier to implement because the maintenance of adjacency relationships is encapsulated in the flip operations.

2. I have demonstrated that the duality concept, in the graph sense of the term, is of significant importance for the modelling of trivariate fields in geoscience. Indeed, the construction and manipulation operations presented are all based on the DT (the bistellar flips are based on simplices), while many GIS operations are based on the properties of the VD.

3. I have presented a new algorithm to delete a vertex in a DT. It uses flips, and the 'unflipping' method was proposed to ensure robustness when the use of a perturbation scheme is impossible. An example of such a case is when a kinetic system is used, because making sure that the perturbations stay coherent with the DT at all times is (almost) impossible.

4. A new algorithm to compute the natural neighbour coordinates (and natural neighbour interpolation) in any dimensions was also proposed. As is the case with the manipulation operations, it is based on the bistellar flips, and yields fast theoretical and practical performances. That permits us to reconstruct a 3D field from the set of samples that were collected to study it.

5. I have offered a detailed description of the algorithms to build the dynamic and kinetic 3D DT, and discussed how to make them robust against all degenerate cases. This research has therefore helped in bridging the gap between theory and practice in computational geometry.

6. To study and extract information from many fields, a variant of the map algebra framework, in which all the fields and all the operation are Voronoi-based, was proposed.

7. A new data structure, the augmented quad-edge, was also developed. To my knowledge, this is the only *implementable* data structure that can store simultaneously a three-dimensional subdivision and its dual (its use is not restricted to the VD and the DT).

8.3 Further Research

The work presented in this thesis answers many questions, but also raises a few more. What follows is a brief summary of the issues that need further research.

Movement of many points in a VD. In the prototype GIS developed, it is only possible to move a single point in the VD at a time. The algorithm to move several at the same time is known and was described, but there are concerns about the speed of a system where all the points would be moving. It would therefore be interesting to implement the method, and test its speed with the simulation of a real process, for instance the simulation of underground water (with the Free-Lagrange method). Comparing the results with other known simulation methods would also be of interest. The implementation of the algorithm with robust arithmetic for the important computations (i.e. intersection of the circumspheres by the trajectory) is, in my opinion, mandatory is one wants to have a program robust in all cases.

DT of large datasets. The prototype GIS developed for this research was not optimised for very big datasets, and the experiments were run for datasets for which all the data structures fit in main memory. In practice, this hinders considerably the use of the prototype for real-world applications since datasets collected to study the Earth, because they are three-dimensional and dynamic, can become enormous.

Creating the DT of very large datasets has become in recent years a topic in itself, with many publications approaching the problem from different perspectives. One way of constructing the DT of a larger input size is simply to use a less space-consuming data structure. Blandford et al. (2005) propose one where only 7.5 bytes per tetrahedron is used, permitting to construct DTs of datasets 4–5 times larger than with a tetrahedron-based data structure. The main drawback of this method is that fewer adjacency and incidence relationships between the elements of the DT are stored explicitly, requiring more work to manipulate the DT, or to extract the VD. Another drawback is that attributes can only be attached to edges, and not to faces and tetrahedra.

For larger point sets—and we are talking about dataset containing many millions of points here—we have to resort to using disk storage. But even with that, problems arise because the memory of modern computers is hierarchical, and works under the assumption that recently used data will be used again soon. This is unfortunately in contradiction with the randomness assumption on which the incremental insertion algorithm is based. When the main memory is exhausted during a computation, paging with disk begins, which slows down tremendously the speed of construction, and can even stop the program. Amenta et al. (2003) circumvent this problem by modifying the insertion order of the points, to find a balance between spatial coherence (points are inserted relatively close to each other) and keeping enough randomness in the input. They show that the performance of `CGAL` and `Pyramid` are for instance considerably increased with their input order: the running time is faster, and the DT of much larger datasets is also possible. Similar ideas were used by Liu and Snoeyink (2005b), but their order is different: input points are ordered along a Hilbert curve. This assumes that the points are evenly distributed in space, and would not be optimal for geoscientific datasets. Also, Isenburg et al. (2006) recently exploited the spatial coherence already existing in datasets, and tweaked it slightly to create an order of insertion that permits them to construct, for instance, the 3D DT of 60 million points in less than one hour.

The good news is that the use of these methods requires only slight modifications to the code of the program built for this research—no rewriting of a new algorithm or data structure is necessary. It would be particularly interesting to see the influence of very anisotropic datasets with these methods.

Discontinuities in geology. The use of the duality between the VD and the DT, or an arbitrary subdivision and its dual, seems to have potential for the modelling of geological structures. The discontinuities and faults could be modelled with one subdivision, while its dual would permit us to represent the spatial variation of certain attributes inside the volume bounded by the discontinuities. The use of VD has already been used to reconstruct geological structures, but it was in two dimensions as cross-sections were used (Boissonnat and Nullans, 1996). Similar ideas could be used directly in three dimensions.

Augmented quad-edge. A preliminary implementation of the AQE was made during its development, to help validate the concepts. The core of the AQE, its navigation operators, and the flips operations were implemented. To test the AQE, the incremental insertion algorithm to construct the VD/DT was used. The code was not optimised in any ways for space or time of construction, and it was therefore meaningless to compare the running time of the construction of the DT with the two different data structures implemented. The development of this new data structure raised several issues that need further investigation.

The construction operators described in this thesis are rather 'crude', and a set of atomic operators permitting us to construct any subdivisions would definitively enhance the use of the AQE. Thus, low-level operators, similar to *Splice* and *MakeEdge* for the quad-edge, are needed.

It would also be worth exploring if the AQE is appropriate for non-manifold objects, i.e. arbitrary subdivisions of 3-manifolds where 'dangling' faces and edges may exist. The quad-edge permits such cases for 2-manifolds, so the AQE could probably do the same for 3-manifolds.

Finally, it would also be interesting to store the AQE in a database, so that big models and the attributes that are attached to each elements be stored and retrieved when needed. Earlier work suggests that quad-edges, being an algebra, may be stored effectively using relational databases, but this is still being examined (Merrett, 2005).

Bibliography

Akima H (1978). A method of bivariate interpolation and smooth surface fitting for irregularly distributed data points. *ACM Transactions on Mathematical Software*, 4(2):148–159.

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

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.

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.

Albrecht JH (1996). *Universal GIS operations.* Ph.D. thesis, ISPA—University of Vechta, Vechta, Germany.

Amenta N, Choi S, and Kolluri RK (2001). The power crust. In *Proceedings 6th ACM Symposium on Solid Modeling and Applications*, pages 249–260. Ann Arbor, Michigan, USA.

Amenta N, Choi S, and Rote G (2003). Incremental constructions con BRIO. In *Proceedings 19th Annual Symposium on Computational Geometry*, pages 211–219. ACM Press, San Diego, USA.

Amenta N, Marshall B, and Eppstein D (1998). The crust and the beta-skeleton: Combinatorial curve reconstruction. *Graphical Models and Image Processing*, 60(2):125–135.

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

Armstrong MP (1988). Temporality in spatial databases. In *Proceedings GIS/LIS '88*, volume 2, pages 880–889. San Antonio, USA.

Aronoff S (1991). *Geographic information systems: A management perspective.* WDL Publications, Ottawa.

Attali D and Boissonnat JD (2002). A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. In *Proceedings 7th ACM Symposium on Solid Modeling and Applications*, pages 139–146. Saarbrücken, Germany.

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

Aurenhammer F (1987). Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16:78–96.

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

Bailey TC and Gatrell AC (1995). *Interactive spatial data analysis.* Wiley, New York.

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.

Bak PRG and Mill AJB (1989). Three dimensional representation in a Geoscientific Resource Management System for the minerals industry. In Raper J, editor, *Three Dimensional Applications in Geographic Information Systems*, pages 155–182. Taylor & Francis.

Barber CB, Dobkin DP, and Huhdanpaa HT (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483.

Barjenbruch DB, Thaler E, and Szoke EJ (2002). Operational applications of three dimensional air parcel trajectories using awips d3d. In *Interactive Symposium on AWIPS*. Orlando, Florida, USA.

Batty M, Couclelis H, and Eichen M (1997). Urban systems as cellular automata. *Environment and Planning B: Planning and Design*, 24(2):159–164.

Baumgart BG (1975). A polyhedron representation for computer vision. In *National Computer Conference*. AFIPS.

Bernard L, Schmidt B, and Streit U (1998). ATMOGIS—integration of atmospheric models and GIS. In *Proceedings 8th International Symposium on Spatial Data Handling*, pages 267–276. Vancouver, B.C., Canada.

Berry JK (1993). Cartographic modeling: The analytical capabilities of GIS. In Goodchild MF, Parks BO, and Steyaert LT, editors, *Environmental Modeling with GIS*, pages 58–74. Oxford University Press, New York.

Bertrand Y, Dufourd JF, Françon J, and Lienhardt P (1993). Algebraic specification and development in geometric modeling. In *Proceedings TAPSOFT'93*, volume 668 of *Lecture Notes in Computer Science*, pages 75–89. Orsay, France.

Betancourt T (2004). Some concepts in atmospheric data. In *The ArcGIS Atmospheric Data Modeling Workshop*. Seattle, USA.

Blandford DK, Blelloch GE, Cardoze DE, and Kadow C (2005). Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24.

Boissonnat JD and Cazals F (2002). Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Computational Geometry—Theory and Applications*, 22:185–203.

Boissonnat JD, Devillers O, Pion S, Teillaud M, and Yvinec M (2002). Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19.

Boissonnat JD and Nullans S (1996). Reconstruction of geological structures from heterogeneous and sparse data. In *Proceedings 4th International Workshop on Advances in Geographic Information Systems*, pages 172–179. ACM Press, Rockville, Maryland, USA.

Boissonnat JD and Yvinec M (1998). *Algorithmic geometry.* Cambridge University Press, Cambridge, United Kingdom.

Boudriault G (1987). Topology in the TIGER file. In *Proceedings 8th International Symposium on Computer Assisted Cartography.* Baltimore, USA.

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

Bowyer A (1981). Computing Dirichlet tessellations. *Computer Journal*, 24(2):162–166.

Breman J, editor (2002). *Marine geography: GIS for the oceans and seas.* ESRI Press, Redlands, USA.

Brisson E (1989). Representing geometric structures in d dimensions: Topology and order. In *Proceedings 5th Annual Symposium on Computational Geometry*, pages 218–227. ACM Press, Saarbrücken, West Germany.

Brown KQ (1979). Voronoi diagrams from convex hulls. *Information Processing Letters*, 9(5):223–228.

Bruggesser H and Mani P (1971). Shellable decompositions of cells and spheres. *Mathematica Scandinavica*, 29:197–205.

Bruns HT and Egenhofer MJ (1997). User interfaces for Map Algebra. *Journal of the Urban and Regional Information Systems Association*, 9(1):44–54.

Buelens B (2005). Personal communication.

Buelens B, Williams R, Sale A, and Pauly T (2005). Model inversion for midwater multibeam backscatter data analysis. In *Proceedings IEEE Oceans'05 Europe.* Brest, France.

Burrough PA (1992). Are GIS data structures too simple minded? *Computers & Geosciences*, 18(4):395–400.

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.

Carlson E (1987). Three-dimensional conceptual modeling of subsurfaces structures. In *Proceedings 8th International Symposium on Computer-Assisted Cartography (Auto-Carto 8)*, pages 336–345. Falls Church, VA, USA.

Carr H, Snoeyink J, and Axen U (2003). Computing contour trees in all dimensions. *Computational Geometry—Theory and Applications*, 24:75–94.

Chapman L and Thornes JE (2003). The use of geographical information systems in climatology and meteorology. *Progress in Physical Geography*, 27(3):313–330.

Chazelle B et al. (1996). Application challenges to computational geometry. Technical Report TR-521-96, Department of Computer Science, Princeton University, USA. (The report is by the Computational Geometry Impact Task Force, which includes a long list of computational geometers).

Cheng SW, Dey TK, Edelsbrunner H, Facello MA, and Teng SH (2000). Sliver exudation. *Journal of the ACM*, 47(5):883–904.

Chew LP (1990). Building Voronoi diagrams for convex polygons in linear expected time. Technical Report PCS-TR90-147, Dartmouth College, Computer Science, Hanover, NH.

Cignoni P, Montani C, Puppo E, and Scopigno R (1996). Optimal isosurface extraction from irregular volume data. In *Proceedings 1996 IEEE Symposium on Volume Visualization*, pages 31–38. San Francisco, USA.

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.

Ciolli M, de Franceschi M, Rea R, Vitti A, Zardi D, and Zatelli P (2004). Development and application of 2D and 3D GRASS modules for simulation of thermally driven slope winds. *Transactions in GIS*, 8(2):191–209.

Clarkson KL (1983). Fast algorithms for the all nearest neighbors problem. In *Proceedings 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 226–232.

Coppock JT and Rhind DW (1991). History of GIS. In Maguire DJ, Goodchild MF, and Rhind DW, editors, *Geographical Information Systems*, pages 21–43. Longman Scientific and Technical, New York.

Couclelis H (1985). Cellular worlds: A framework for modeling micro-macro dynamics. *Environment and Planning A*, 17:585–596.

Couclelis H (1992). People manipulate objects (but cultivate fields): Beyond the raster-vector debate in GIS. In Frank AU, Campari I, and Formentini U, editors, *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, volume 639 of *Lecture Notes in Computer Science*, pages 65–77. Springer-Verlag.

Couclelis H (1999). Space, time, geography. In Longley PA, Goodchild MF, Maguire DJ, and Rhind DW, editors, *Geographical Information Systems*, pages 29–38. John Wiley & Sons, second edition.

Davis BE and Davis PE (1988). Marine GIS: Concepts and considerations. In *Proceedings GIS/LIS '88*. Falls Church, VA, USA.

de Berg M, van Kreveld M, Overmars M, and Schwarzkopf O (2000). *Computational geometry: Algorithms and applications*. Springer-Verlag, Berlin, second edition.

de Berg M, van Kreveld M, van Oostrum R, and Overmars M (1997). Simple traversal of a subdivision without extra storage. *International Journal of Geographical Information Science*, 11(4):359–374.

De Fabritiis G and Coveney PV (2003). Dynamical geometry for multiscale dissipative particle dynamics. *Computer Physics Communications*, 153:209–226.

De Vasconcelos MJP, Gonçalves A, Catry FX, Paúl JU, and Barros F (2002). A working prototype of a dynamic geographical information system. *International Journal of Geographical Information Science*, 16(1):69–91.

Delaunay BN (1934). Sur la sphère vide. *Izvestia Akademia Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7:793–800.

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

Devillers O (2002b). The Delaunay hierarchy. *International Journal of Foundations of Computer Science*, 13(2):163–180.

Devillers O, Pion S, and Teillaud M (2002). Walking in a triangulation. *International Journal of Foundations of Computer Science*, 13(2):181–199.

Devillers O and Teillaud M (2003). Perturbations and vertex removal in a 3D Delaunay triangulation. In *Proceedings 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 313–319. Baltimore, MD, USA.

Dey TK (2004). Curve and surface reconstruction. In Goodman JE and O'Rourke J, editors, *Handbook of Discrete and Computational Geometry*. CRC Press, second edition.

Dey TK and Goswami S (2003). Tight cocone: A water-tight surface reconstructor. In *Proceedings 8th ACM Symposium on Solid Modeling and Applications*. Seattle, Washington, USA.

Dirichlet GL (1850). Über die reduktion der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *Journal für die Reine und Angewandte Mathematik*, 40:209–227.

Dobkin DP and Laszlo MJ (1989). Primitives for the manipulation of three-dimensional subdivisions. *Algorithmica*, 4:3–32.

Douglas D (1974). It makes me so CROSS. Unpublished manuscript from the Harvard Laboratory for Computer Graphics and Spatial Analysis. Reprinted in Peuquet DJ, Marble, DF, editors, *Introductory Readings in Geographic Information System*, pages 303–307. Taylor & Francis, London.

Dwyer RA (1991). Higher-dimensional Voronoi diagrams in linear expected time. *Discrete & Computational Geometry*, 6:343–367.

Edelsbrunner H (1990). An acyclicity theorem for cell complexes in d dimensions. *Combinatorica*, 10:251–260.

Edelsbrunner H (2001). *Geometry and topology for mesh generation.* Cambridge University Press, Cambridge, UK.

Edelsbrunner H, Harer J, Mascarenhas A, and Pascucci V (2004). Time-varying reeb graphs for continuous space-time data. In *Proceedings 20th Annual Symposium on Computational Geometry*, pages 366–372. ACM Press, Brooklyn, New York, USA.

Edelsbrunner H and Mücke EP (1990). Simulation of Simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104.

Edelsbrunner H and Mücke EP (1994). Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72.

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

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

Egenhofer MJ (1995). Topological relations in 3D. Technical report, University of Maine, Orono, USA.

Egenhofer MJ and Herring JR (1990). A mathematical framework for the definition of topological relationships. In *Proceedings 4th International Symposium on Spatial Data Handling*, pages 803–813. Zurich, Switzerland.

Emery WJ and Thomson RE (2001). *Data analysis methods in physical oceanography*. Elsevier, Amsterdam, second and revised edition.

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

Facello MA (1995). Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. *Computer Aided Geometric Design*, 12:349–370.

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

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.

Fischer K (2005). Introduction to alpha shapes. http://n.ethz.ch/student/fischerk/alphashapes/as/.

Fisher PF (1997). The pixel: A snare and a delusion. *International Journal of Remote Sensing*, 18(3):679–685.

Fortune S (1987). A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174.

Fotheringham AS and Wegener M, editors (2000). *Spatial models and GIS : New potential and new models*. Taylor & Francis, London.

Fowler RJ and Little JJ (1979). Automatic extraction of irregular network digital terrain models. In *SIGGRAPH '79: Proceedings 6th Annual Conference on Computer Graphics and Interactive Techniques*, pages 199–207. Chicago, USA.

Frank AU (1992). Spatial concepts, geometric data models, and geometric data structures. *Computers & Geosciences*, 18(4):409–417.

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

Freundschuh SM and Egenhofer MJ (1997). Human conceptions of spaces: Implications for geographic information systems. *Transactions in GIS*, 2(4):361–375.

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

Galton A (2004). Fields and objects in space, time, and space-time. *Spatial Cognition and Computation*, 4(1):39–68.

Garland M and Heckbert PS (1995). Fast polygonal approximation of terrain and height fields. Technical Report CMU-CS-95-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA.

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.

George RA, Gee L, Hill AW, Thomson JA, and Jeanjean P (2002). High-resolution AUV surveys of the eastern Sigsbee escarpment. In *Offshore Technology Conference*. Houston, USA.

Gerlotto F, Soria M, and Fréon P (1999). From two dimensions to three: The use of multibeam sonar for a new approach in fisheries acoustics. *Canadian Journal of Fisheries and Aquatic Science*, 56:6–12.

Gold CM (1989). Surface interpolation, spatial adjacency and GIS. In Raper J, editor, *Three Dimensional Applications in Geographic Information Systems*, pages 21–35. Taylor & Francis.

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.

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

Gold CM (1993). Forestry spatial decision support system classification, and the 'flight simulator' approach. In *Proceedings GIS'93: Eyes on the Future*, pages 797–802. Vancouver, Canada.

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

Gold CM (1999). An algorithmic approach to a marine GIS. In Wright DJ and Bartlett D, editors, *Marine and Coastal Geographic Information Systems*, pages 37–52. Taylor & Francis, London.

Gold CM (2006). What is GIS and what is not? *Transactions in GIS*, 10(4):505–520.

Gold CM, Charters TD, and Ramsden J (1977). Automated contour mapping using triangular element data structures and an interpolant over each triangular domain. In *Proceedings Siggraph '77*, volume 11(2), pages 170–175.

Gold CM and Condal AR (1995). A spatial data structure integrating GIS and simulation in a marine environment. *Marine Geodesy*, 18:213–228.

Gold CM and Cormack S (1987). Spatially ordered networks and topographic reconstructions. *International Journal of Geographical Information Systems*, 1(2):137–148.

Gold CM and Edwards G (1992). The Voronoi spatial model: Two- and three-dimensional applications in image analysis. *ITC Journal*, 1:11–19.

Gold CM, Goralski R, and Ledoux H (2003). Development of a 'Marine GIS'. In *Proceedings 21st International Cartographic Conference*. Durban, South Africa.

Gold CM, Ledoux H, and Dzieszko M (2005). A data structure for the construction and navigation of 3D Voronoi and Delaunay cell complexes. In *Proceedings 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 21–22. Plzeň, Czech Republic.

Gold CM and Maydell U (1978). Triangulation and spatial ordering in computer cartography. In *Proceedings Annual Meeting of the Canadian Cartographic Association*, pages 69–81. Toronto, Canada.

Gold CM, Nantel J, and Yang W (1996). Outside-in: An alternative approach to forest map digitizing. *International Journal of Geographical Information Science*, 10(3):291–310.

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.

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

Gold CM and Snoeyink J (2001). A one-step crust and skeleton extraction algorithm. *Algorithmica*, 30:144–163.

Goodchild MF (1987). Spatial analytical perspective on GIS. *International Journal of Geographical Information Systems*, 1:327–334.

Goodchild MF (1992a). Geographical data modeling. *Computers & Geosciences*, 18(4):401–408.

Goodchild MF (1992b). Geographical information science. *International Journal of Geographical Information Systems*, 6(1):31–45.

Green PJ and Sibson R (1978). Computing Dirichlet tessellations in the plane. *Computer Journal*, 21(2):168–173.

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.

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.

Guibas LJ, Knuth DE, and Sharir M (1992). Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7(4):381–413.

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.

Habermann T, Cartwright J, Schweitzer R, Barrodale I, and Davies E (2004). Integrating science data into geographic information systems. In *Proceedings 20th International Conference on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology*. Seattle, USA.

Haklay M (2004). Map Calculus in GIS: A proposal and demonstration. *International Journal of Geographical Information Science*, 18(2):107–125.

Hammerstad E (1995). Advanced multibeam echosounder technology. *Sea Technology*, 36:67–69.

Hatcher GAJ and Maher N (1999). Real-time GIS for marine applications. In Wright DJ and Bartlett D, editors, *Marine and Coastal Geographic Information Systems*, pages 137–147. Taylor & Francis, London.

Head MEM, Luong P, Costolo JH, Countryman K, and Szczechowski C (1997). Applications of 3-D visualizations of oceanographic data bases. In *Proceedings Oceans '97—MTS/IEEE*, volume 2, pages 1210–1215.

Heller M (1990). Triangulation algorithms for adaptive terrain modeling. In *Proceedings 4th International Symposium on Spatial Data Handling*, pages 163–174. Zurich, Switzerland.

Hill FS (2000). *Computer graphics using OpenGL*. Prentice-Hall, second edition.

Hoffmann CM (1989). The problems of accuracy and robustness in geometric computation. *Computer—IEEE Computer Society Press*, 22:31–42.

Houlding SW (1994). *3D geoscience modeling: Computer techniques for geological characterization*. Springer-Verlag, Berlin.

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.

Isenburg M, Liu Y, Shewchuk JR, and Snoeyink J (2006). Streaming computation of Delaunay triangulations. *ACM Transactions on Graphics*, 25(3):1049–1056.

Jerram DA, Cheadle MJ, Hunter RH, and Elliot MT (1996). The spatial distribution of grains and crystals in rocks. *Contributions to Mineralogy and Petrology*, 125(1):60–74.

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

Joe B (1991). Construction of three-dimensional Delaunay triangulations using local transformations. *Computer Aided Geometric Design*, 8:123–142.

Jones CB (1989). Data structures for three-dimensional spatial information systems in geology. *International Journal of Geographical Information Systems*, 3(1):15–31.

Karamcheti V, Li C, Pechtchanski I, and Yap CK (1999). A CORE library for robust numeric and geometric computation. In *Proceedings 15th Annual Symposium on Computational Geometry*, volume 351–359. ACM Press, Miami, Florida, USA.

Karasick M, Lieber D, and Nackman LR (1991). Efficient Delaunay triangulation using rational arithmetic. *ACM Transactions on Graphics*, 10(1):71–91.

Kaufman AE (1996). Volume visualization. *ACM Computing Surveys*, 28(1):165–167.

Kemp KK (1993). Environmental modeling with GIS: A strategy for dealing with spatial continuity. Technical Report 93-3, National Center for Geographic Information and Analysis, University of California, Santa Barbara, USA.

Kemp KK and Vckovski A (1998). Towards an ontology of fields. In *Proceedings 3rd International Conference on GeoComputation*. Bristol, UK.

Kim D, Cho Y, and Kim DS (2004). Protein structure analysis using Euclidean Voronoi diagram of atoms. In *Proceedings International Workshop on Biometric Technologies, Special Forum on Modeling and Simulation in Biometric Technology*, pages 125–129. Calgary, Canada.

Kretz R (1969). On the spatial distribution of crystals in rocks. *Lithos*, 2:39–66.

Kumler MP (1994). An intensive comparison of triangulated irregular networks (TINs) and digital elevation models (DEMs). *Cartographica*, 31(2).

Langran G (1992). *Time in Geographic Information Systems.* Taylor & Francis, London.

Lattuada R (1998). *A triangulation based approach to three dimensional geoscientific modelling.* Ph.D. thesis, Department of Geography, Birkbeck College, University of London, UK.

Lawson CL (1972). Transforming triangulations. *Discrete Applied Mathematics*, 3:365–372.

Lawson CL (1977). Software for C^1 surface interpolation. In Rice JR, editor, *Mathematical software III*, pages 161–194. Academic Press.

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

Ledoux H (2003). Development of a marine geographical information system based on the Voronoi diagram. Department of Land Surveying & Geo-Informatics, The Hong Kong Polytechnic University. Unpublished transfer report.

Ledoux H and Gold CM (2004a). An efficient natural neighbour interpolation algorithm for geoscientific modelling. In Fisher PF, editor, *Developments in Spatial Data Handling—11th International Symposium on Spatial Data Handling*, pages 97–108. Springer.

Ledoux H and Gold CM (2004b). Modelling oceanographic data with the three-dimensional Voronoi diagram. In *ISPRS 2004—XXth Congress*, volume II, pages 703–708. Istanbul, Turkey.

Ledoux H and Gold CM (2005a). Interpolation as a tool for the modelling of three-dimensional geoscientific datasets. In *Proceedings 4th ISPRS Workshop on Dynamic and Multi-dimensional GIS*, pages 79–84. Pontypridd, Wales, UK.

Ledoux H and Gold CM (2005b). The 3D Voronoi diagram and its dual: Simultaneous construction and storage. In *Topology and Spatial Databases Workshop 2005.* Glasgow, Scotland, UK.

Ledoux H and Gold CM (2006a). A Voronoi-based map algebra. In Reidl A, Kainz W, and Elmes G, editors, *Progress in Spatial Data Handling—12th International Symposium on Spatial Data Handling*, pages 117–131. Springer.

Ledoux H and Gold CM (2006b). La modélisation de données océanographiques à l'aide du diagramme de Voronoï tridimensionnel (in French). *Revue internationale de géomatique*, 16(1):51–70.

Ledoux H and Gold CM (2006c). Simultaneous storage of primal and dual three-dimensional subdivisions. *Computers, Environment and Urban Systems.* In press.

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.

Ledoux H, Gold CM, and Baciu G (n.d.). Deleting a vertex in a Delaunay tetrahedralization by flips. *International Journal of Computational Geometry and Applications.* Submitted.

Lee I and Gahegan M (2002). Interactive analysis using Voronoi diagrams: Algorithms to support dynamic update from a generic triangle-based data structure. *Transactions in GIS*, 6(2):89–114.

Lee J (1989). A drop heuristic conversion method for extracting irregular networks for digital elevation models. In *Proceedings GIS/LIS '89*, pages 30–39. Orlando, USA.

Lee J (1991). Comparison of existing methods for building triangular irregular network models. *International Journal of Geographical Information Systems*, 5(3):267–285.

Li R and Saxena NK (1993). Development of an integrated Marine Geographic Information System. *Marine Geodesy*, 16:293–307.

Li Z (1994). Reality in time-scale system and cartographic representation. *The Cartographic Journal*, 31(1):50–51.

Lienhardt P (1994). N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal of Computational Geometry and Applications*, 4(3):275–324.

Liu Y and Snoeyink J (2005a). A comparison of five implementations of 3D Delaunay tessellation. In Goodman JE, Pach J, and Welzl E, editors, *Combinatorial and Computational Geometry*, volume 52, pages 439–458. Cambridge University Press.

Liu Y and Snoeyink J (2005b). TESS3: A program to compute 3D Delaunay tessellations for well-distributed points. In *Proceedings 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 225–234. Seoul, Korea.

Lockwood M and Li R (1995). Marine Geographic Information Systems—what sets them apart? *Marine Geodesy*, 18:157–159.

Longley PA, Goodchild MF, Maguire DJ, and Rhind DW (2001). *Geographic information systems and science*. Wiley, London.

Lopes H and Tavares G (1997). Structural operators for modeling 3-manifolds. In *Proceedings 4th ACM Symposium on Solid Modeling and Applications*, pages 10–18. Atlanta, Georgia, USA.

Lorensen WE and Cline HE (1987). Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 4:163–168.

Lucieer A and Kraak MJ (2004). Alpha-shapes for visualizing irregular shaped class clusters in 3D feature space for classification of remotely sensed imagery. In *Proceedings of SPIE—The International Society for Optical Engineering*, volume 5295, pages 201–211.

Mackenzie JR, Heilbronner R, and Stünitz H (2006). Simulating crystalline microstructures—an aid in quantifying the spatial distribution of phases in two-phases crystalline rocks. In *Proceedings General Assembly 2006 of the European Geosciences Union (EGU 2006)*. Vienna, Austria.

Mäntylä M (1988). *An introduction to solid modeling*. Computer Science Press, New York, USA.

Mark DM (1992). Spatial metaphors for human-computer interaction. In *Proceedings 5th International Symposium on Spatial Data Handling*, pages 104–112. Charleston, SC, USA.

Mark DM, Chrisman NR, Frank AU, McHaffie PH, and Pickles J (1997). The GIS history project. In *UCGIS Summer Assembly*. Bar Harbor, Maine, USA.

Mark DM, Frank AU, Egenhofer MJ, Freundschuh SM, McGranaghan M, and White RM (1989). Languages of spatial relations (Initiative 2). Technical report, National Center for Geographic Information and Analysis.

Mascarenhas A and Snoeyink J (2005). Implementing time-varying contour trees. In *Proceedings 21st Annual Symposium on Computational Geometry*, pages 370–371. ACM Press, Pisa, Italy.

Mason NC, O'Conaill MA, and Bell SBM (1994). Handling four-dimensional geo-referenced data in environmental GIS. *International Journal of Geographical Information Systems*, 8(2):191–215.

Matheron G (1971). The theory of regionalised variables and its applications. *Les Cahiers du Centre de Morphologie Mathématique de Fontainebleu*, 5.

Mayer LA, Li Y, and Melvin G (2002). 3D visualization for pelagic fisheries research and assessment. *ICES Journal of Marine Sciences*, 59:216–225.

Meaden GJ (1999). Applications of GIS to fisheries management. In Wright DJ and Bartlett D, editors, *Marine and Coastal Geographic Information Systems*, pages 205–226. Taylor & Francis, London.

Meaden GJ (2000). GIS in fisheries management. *GeoCoast*, 1(1):82–101.

Mennis J, Viger R, and Tomlin CD (2005). Cubic map algebra functions for spatio-temporal analysis. *Cartography and Geographic Information Science*, 32(1):17–32.

Merrett TH (2005). Topological structures for spatial databases in two and three dimensions. In *Proceedings 17th Australasian Database Conference (ADC2005)*. Newcastle, Australia.

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

Morehouse S (1985). ARC/INFO: A geo-relational model for spatial information. In *Proceedings 7th International Symposium on Computer Assisted Cartography*. Washington DC, USA.

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

Mostafavi MA (2006). Personal communication.

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.

Mostafavi MA, Gold CM, and Dakowicz M (2003). Delete and insert operations in Voronoi/Delaunay methods and applications. *Computers & Geosciences*, 29(4):523–530.

Mücke EP (1998). A robust implementation for three-dimensional Delaunay triangulations. *International Journal of Computational Geometry and Applications*, 8(2):255–276.

Mücke EP, Saias I, and Zhu B (1999). Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. *Computational Geometry—Theory and Applications*, 12:63–83.

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.

Nichols CR, Porter DL, and Williams RG (2003). *Recent advances and issues in oceanography*. Greenwood Press, Wesport, USA.

O'Conaill MA, Bell SBM, and Mason NC (1992). Developing a prototype 4D GIS on a transputer array. *ITC Journal*, 1992(1):47–54.

Okabe A, Boots B, and Sugihara K (1994). Nearest neighbourhood operations with generalized Voronoi diagrams: A review. *International Journal of Geographical Information Systems*, 8(1):43–71.

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

Oliver GW (1995). Visualizing the tracking and diving behavior of marine mammals: A case study. In *Proceedings 6th IEEE conference on Visualization '95*, pages 397–399.

Oliver MA and Webster R (1990). Kriging: A method of interpolation for geographical information systems. *International Journal of Geographical Information Systems*, 4:313–332.

Owen SJ (1992). *An implementation of natural neighbor interpolation in three dimensions*. Master's thesis, Department of Civil Engineering, Brigham Young University, Provo, UT, USA.

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

Paton M, Mayer LA, and Ware C (1997). Interactive 3-D tools for pipeline route planning. In *Proceedings Oceans '97—MTS/IEEE*, volume 2, pages 1216–1222. Halifax, Canada.

Peucker TK, Fowler RJ, Little JJ, and Mark DM (1978). The triangulated irregular network. In *Proceedings Digital Terrain Models Symposium*, pages 516–532. American Society of Photogrammetry.

Peuquet DJ (1984). A conceptual framework and comparison of spatial data models. *Cartographica*, 21(4):66–113.

Peuquet DJ (1994a). An approach for time-based analysis of spatio-temporal data. In *Proceedings 6th International Symposium on Spatial Data Handling*, pages 489–504. Edinburgh, Scotland.

Peuquet DJ (1994b). It's about time: A conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of American Geographers*, 84:441–461.

Peuquet DJ (2001). Making space for time: Issues in space-time data representation. *GeoInformatica*, 5(1):11–32.

Peuquet DJ, Smith B, and Brogaard B (1999). The ontology of fields: Report of a specialist meeting held under the auspices of the VARENIUS project. Technical report, National Center for Geographic Information and Analysis, Santa Barbara, USA.

Pigot S (1992). A topological model for a 3D spatial information system. In *Proceedings 5th International Symposium on Spatial Data Handling*, pages 344–359. Charleston, SC, USA.

Pigot S and Hazelton NWJ (1992). The fundamentals of a topological model for a four-dimensional GIS. In *Proceedings 5th International Symposium on Spatial Data Handling*, pages 580–591. Charleston, SC, USA.

Pilouk M (1996). *Integrated modelling for 3D GIS.* Ph.D. thesis, ITC, The Netherlands.

Preparata FP and Shamos MI (1985). *Computational Geometry: An Introduction.* Springer.

Pullar D (2001). Mapscript: A map algebra programming language incorporating neighborhood analysis. *GeoInformatica*, 5(2):145–163.

Rajan VT (1991). Optimality of the Delaunay triangulation in $\mathbb{R}^d$. In *Proceedings 7th Annual Symposium on Computational Geometry*, pages 357–363. ACM Press, North Conway, New Hampshire, USA.

Raper J, editor (1989). *Three dimensional applications in Geographic Information Systems.* Taylor & Francis, London.

Raper J (1992). An atlas of three dimensional functions. In Pflug R and Harbaugh JW, editors, *Computer Graphics in Geology*, volume 41, pages 41–49. Springer-Verlag.

Raper J and Livingstone D (1995). Development of a geomorphological spatial model using object-oriented design. *International Journal of Geographical Information Science*, 9(4):359–383.

Ritter GX, Wilson JN, and Davidson JL (1990). Image algebra: An overview. *Computer Vision, Graphics, and Image Processing*, 49(3):297–331.

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

Saalfeld A (1987). It doesn't make me nearly as CROSS: Some advantages of the point-vector representation of line segments in automated cartography. *International Journal of Geographical Information Systems*, 1(4):379–386.

Sambridge M, Braun J, and McQueen H (1995). Geophysical parameterization and interpolation of irregular data using natural neighbours. *Geophysical Journal International*, 122:837–857.

Samet H (1984). The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260.

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

Schick R (2002). Using GIS to track right whales and bluefin tuna in the atlantic ocean. In Wright DJ, editor, *Undersea with GIS*, pages 65–84. ESRI Press, Redlands, USA.

Schönhardt E (1928). Über die zerlegung von dreieckspolyedern in tetraeder. *Mathematische Annalen*, 98:309–312.

Seidel R (1982). *Voronoi diagrams in higher dimensions.* Ph.D. thesis, Diplomarbeit, Institut für Informationsverarbeitung, Technische Universität Graz, Austria.

Seidel R (1998). The nature and meaning of perturbations in geometric computing. *Discrete & Computational Geometry*, 19:1–17.

Shewchuk JR (1997a). Adaptive precision floating point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18:305–363.

Shewchuk JR (1997b). *Delaunay Refinement Mesh Generation*. Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA.

Shewchuk JR (2000). Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *Proceedings 16th Annual Symposium on Computational Geometry*, pages 350–359. ACM Press, Hong Kong.

Shewchuk JR (2003). Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *Proceedings 19th Annual Symposium on Computational Geometry*, pages 181–190. ACM Press, San Diego, USA.

Sibson R (1978). Locally equiangular triangulations. *Computer Journal*, 21:243–245.

Sibson R (1980). A vector identity for the Dirichlet tesselation. In *Mathematical Proceedings of the Cambridge Philosophical Society*, pages 151–155.

Sibson R (1981). A brief description of natural neighbour interpolation. In Barnett V, editor, *Interpreting Multivariate Data*, pages 21–36. Wiley, New York, USA.

Snoeyink J (1997). Point location. In Goodman JE and O'Rourke J, editors, *Handbook of Discrete and Computational Geometry*, pages 559–574. CRC Press, Boca Raton, Florida, USA.

Sprague K, de Kemp E, Wong W, McGaughey J, Perron G, and Barrie T (2006). Spatial targeting using queries in a 3-D GIS environment with application to mineral exploration. *Computers & Geosciences*, 32:396–418.

Stevens SS (1946). On the theory of scales and measurement. *Science*, 103:677–680.

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

Su Y and Sheng Y (1999). Visualizing upwelling at Monterey Bay in an integrated environment of GIS and scientific visualization. *Marine Geodesy*, 22:93–103.

Sugihara K and Hiroshi I (1995). Why is the 3D Delaunay triangulation difficult to construct? *Information Processing Letters*, 54:275–280.

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

Suzuki A and Iri M (1986). Approximation of a tesselation of the plane by a Voronoi diagram. *Journal of the Operations Research Society of Japan*, 29:69–96.

Takeyama M (1996). *Geo-Algebra: A mathematical approach to integrating spatial modeling and GIS*. Ph.D. thesis, Department of Geography, University of California at Santa Barbara, USA.

Takeyama M and Couclelis H (1997). Map dynamics: Integrating cellular automata and GIS through Geo-Algebra. *International Journal of Geographical Information Science*, 11(1):73–91.

Tam R and Heidrich W (2003). Shape simplification based on the medial axis transform. In *Proceedings IEEE Visualization 2003*, pages 481–488. IEEE Computer Society, Seattle, Washington, USA.

Theobald DM (2001). Topology revisited: Representing spatial relations. *International Journal of Geographical Information Science*, 15(8):689–705.

Thiessen AH (1911). Precipitation average for large areas. *Monthly Weather Review*, 39:1082–1084.

Tobler W (1970). A computer movie simulating urban growth in the Detroit region. *Economic Geography*, 46(2):234–240.

Tomlin CD (1983). A map algebra. In *Proceedings of the 1983 Harvard Computer Graphics Conference*, pages 127–150. Cambridge, MA, USA.

Tomlinson RF (1988). The impact of the transition from analogue to digital cartographic representation. *The American Cartographer*, 15(3):249–261.

Tse ROC and Gold CM (2004). TIN meets CAD—extending the TIN concept in GIS. *Future Generation Computer Systems*, 20(7):1171–1184.

Underwood E (1970). *Quatitative stereology.* Addison-Wesley Publishing, Reading, USA.

Valavanis VD (2002). *Geographic information systems in oceanography and fisheries.* Taylor & Francis.

van Kreveld M (1997). Digital elevation models and TIN algorithms. In van Kreveld M, Nievergelt J, Roos T, and Widmayer P, editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 37–78. Springer-Verlag, Berlin.

van Oosterom P (1997). Maintaining consistent topology including historical data in large spatial database. In *Proceedings ACSM/ASPRS Annual Convention and Exposition*, pages 327–336. Auto-Carto 13, Seattle, Washingtion, USA.

van Oosterom P (1999). Spatial access methods. In Longley PA, Goodchild MF, Maguire DJ, and Rhind DW, editors, *Geographical Information Systems*, volume 1, pages 385–400. John Wiley & Sons.

Varma H (1999). Applying spatio-temporal concepts to correlative data analysis. In Wright DJ and Bartlett D, editors, *Marine and Coastal Geographic Information Systems*, pages 75–93. Taylor & Francis, London.

Vitti A, Zatelli P, and Zottele F (2004). 3D vector approach to local thermally driven slope winds modeling. In *Proceedings FOSS/GRASS Users Conference 2004*. Bangkok, Thailand.

Voronoi GM (1908). Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *Journal für die Reine und Angewandte Mathematik*, 134:198–287.

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

Watson DF (1992). *Contouring: A guide to the analysis and display of spatial data.* Pergamon Press, Oxford, UK.

Watson DF (2001). Compound signed decomposition, the core of natural neighbor interpolation in n-dimensional space. Unpublished manuscript, available at http://www.iamg.org/naturalneighbour.html.

Watson DF and Phillip GM (1987). Neighborhood-based interpolation. *Geobyte*, 2(2):12–16.

Wilhems J and van Gelder A (1990). Topological considerations in isosurface generation. *Computer Graphics*, 24(5):79–86.

Wolfram S (1986). *Theory and applications of cellular automata.* World Scientific, Singapore.

Worboys MF (1992). A model for spatio-temporal information. In *Proceedings 5th International Symposium on Spatial Data Handling*, volume 2, pages 602–611. Charleston, SC, USA.

Worboys MF and Duckham M (2004). *GIS: A computing perspective.* CRC Press, second edition edition.

Wright DJ and Goodchild MF (1997). Data from the deep: Implications for the GIS community. *International Journal of Geographical Information Science*, 11(6):523–528.

Xue Y, Sun M, and Ma A (2004). On the reconstruction of three-dimensional complex geological objects using Delaunay triangulation. *Future Generation Computer Systems*, 20(7):1227–1234.

Yap CK and Dubé T (1995). The exact computation paradigm. In Du DZ and Hwang FK, editors, *Computing in Euclidean Geometry*, pages 452–486. World Scientific Press, second edition.

Zlatanova S, Abdul Rahman A, and Shi W (2004). Topological models and frameworks for 3D spatial objects. *Computers & Geosciences*, 30:419–428.

Zubin D (1989). Natural language understanding and reference frames. In Mark DM, Frank AU, Egenhofer M, Freundschuh SM, McGranaghan M, and White RM, editors, *Languages of Spatial Relations: Initiative 2 Specialist Meeting Report*, pages 13–16. National Center for Geographic Information and Analysis, Santa Barbara, USA.