University of West Bohemia
Department of Computer Science and Engineering
Univerzitní 8
306 14 Pilsen
Czech Republic

Regular Triangulation in 3D and Its Applications

The State of the Art and Concept of Ph.D. Thesis

Michal Zemek

Technical Report No. DCSE/TR-2009-03
April 2009

Distribution: Public

The State of the Art and Concept of Ph.D. Thesis
April 2009

Regular Triangulation in 3D and Its Applications

Michal Zemek

The Delaunay triangulation is one of the fundamental data structures of the computational geometry. The regular triangulation is its generalization, which reflects the weights of the input points. This work is focused on three-dimensional regular triangulations within the context of the dynamic variable data, and on the applications of regular triangulations for the biochemistry.

In the first part, we describe several algorithms for construction of regular triangulations and also algorithms allowing to delete points in regular triangulations. Furthermore we discuss the problem of maintaining kinetic and dynamic triangulations. In the second part, we describe in detail how regular triangulations can be used in biochemistry in the search for channels (cavities) in protein molecules. In the third part, we show the results of our research – we describe our method of computation of channels in dynamic proteins and a novel algorithm for point deletion in regular triangulations. Finally we sketch our future work.

This work was supported by

- the Ministry of Education of the Czech Republic, Project No. LC06008,
- the Grant Agency of the Czech Republic, Project No. 201/09/0097.

Contents

Chapter 1

Introduction

Voronoi diagrams denote wide class of geometric constructs dividing a space into a set of cells – for a given set of so-called generators (or sites), each point of a space is associated with a generator which minimizes a certain distance function. The very first application of Voronoi diagrams goes back to the middle of 19th century, when Carl F. Gauss [25] used them for an analysis of quadratic forms. Gustav P. L. Dirichlet continued in his work [18], therefore, Voronoi diagrams are also often called the Dirichlet tessellation. At the beginning of the 20th century, Georgy Voronoi generalized their results in higher dimensions [47]. Since the time, Voronoi diagrams have found their applications in mathematics, crystallography, cartography, geology, biochemistry and also in computer science, where they are used for path planning, collision detection, surface reconstruction, interpolation, iso-surface extraction, etc.

Since their origin, many variations of Voronoi diagrams have been introduced – they vary in the type of generators (points, weighted points, spheres, lines, etc.) and in the used distance function. One of these variations are power diagrams. Power diagrams use weighted points as generators. The weight of a generator is taken into account in the distance function and so it impacts the influence of the generator to its neighborhood. This can be used for example in path-planning – weights of obstacles (generators) are given by their size or dangerousness.

In my research, I focus on the dual structure of power diagrams – on regular triangulations. Thanks to the duality, any computation performed on power diagrams can be also done on regular triangulations. The benefit of regular triangulations is their simpler representation. Consequently, an implementation of an algorithm for constructing or maintaining regular triangulations is mostly simpler than an implementation of the corresponding algorithm working directly with power diagrams. My work is focused on three-dimensional regular triangulations within the context of the dynamic variable data, and on the application of regular triangulations for the analysis of channels in protein molecules. These two topics are closely related – atoms in proteins change their position and so do the channels.

This thesis is organized as follows. In Chapter 2, I recall the definition of regular triangulations. Chapter 3 describes several algorithms allowing the on-line construction of regular triangulations and in the following Chapter 4 the opposite problem is discussed – a deletion of points in regular triangulations. Chapter 5 is focused on triangulations of a variable set of points. How to compute channels in static proteins via a regular triangulation is explained in Chapter 6. In Chapter 7, the results of my research are discussed. First I describe a novel algorithm of computation of channels in dynamic proteins, further I propose my algorithm for point deletion. Chapter 8 concludes this thesis and sketches my future work.

Chapter 2

Regular Triangulation and Voronoi Diagrams

In this Chapter, we give a definition of regular triangulations and recall the relationship between regular triangulations in 3D and convex hulls in 4D. Further we describe power diagrams – the dual structure of regular triangulations and also we mention Euclidean Voronoi diagrams.

2.1 Regular Triangulation

The regular triangulation (RT) is a generalization of well-known Delaunay triangulation (DT). Each point in the regular triangulation is associated with a real number – a weight of the point. If weights of all points are equal, then the regular triangulation and the Delaunay triangulation of this set of points are identical, otherwise the triangulations can be different. We start with a few basic definitions, further we give the exact definition of the regular triangulation and a lemma useful for a construction of the regular triangulation (see [21]).

Triangulation Given a set of points S in E^3, the triangulation $T(S)$ of this set of points is a set of tetrahedra such that:

- A point $p \in E^3$ is a vertex of a tetrahedron in $T(S)$ only if $p \in S$.
- The intersection of two tetrahedra of $T(S)$ is either empty or it is a shared face, a shared edge or a shared vertex.
- The union of all tetrahedra in $T(S)$ entirely fulfills the convex hull of S.

Note that in this definition of $T(S)$ we do not require each $p \in S$ to be a vertex of $T(S)$.

Weighted point A point $p \in E^3$ with an associated weight $w_p \in \mathbb{R}$ is called a weighted point. If the weight w_p is non-negative, then p can be interpreted as a sphere centered at the point p with a radius $\sqrt{w_p}$.

Power distance A power distance of a weighted point p from a point $x \in E^3$ (no matter whether x is weighted or unweighted) is defined as

$$\pi_p(x) = |px|^2 - w_p, \tag{2.1}$$

where $|px|$ denotes the Euclidean distance between the points p and x. The power distance $\pi_p(x)$ can be interpreted as a square of length of a tangent from the point x to a sphere centered at p and with the radius $\sqrt{w_p}$ (if x lies outside this sphere), see Fig. 2.1a.

Orthogonal points Two weighted points p and q are said to be orthogonal if $|pq|^2 = w_p + w_q$, i.e. $\pi_p(q) = w_q$, see Fig. 2.1b.

Orthogonal center Let a, b, c, d be non-coplanar weighted points. A weighted point z is an orthogonal center of a tetrahedron *abcd* if z is orthogonal to the points a, b, c, d.

Global regularity Let z be the orthogonal center of a tetrahedron *abcd*. The tetrahedron *abcd* is globally regular with respect to a set of weighted points S if $\pi_z(p) > w_p$ for each point $p \in S - \{a, b, c, d\}$.

Regular triangulation A triangulation $T(S)$ is a regular triangulation of S ($RT(S)$) if each tetrahedron in $T(S)$ is globally regular with respect to S.

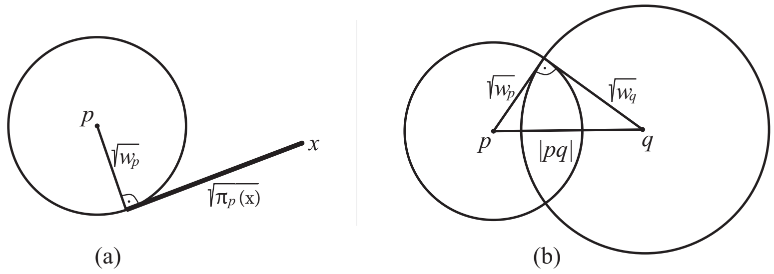


Figure 2.1: (a) The power distance $\pi_p(x)$. (b) Two orthogonal weighted points.

Redundant point A point p, $p \in S$, is called a redundant point if no globally regular tetrahedron *pabc* exists in $RT(S)$. Redundant points are not vertices of any tetrahedron in $RT(S)$, therefore, the vertex set of $RT(S)$ is generally only a subset of S.

Local regularity Let *abcd* and *abce* be two tetrahedra with a common face *abc* and z be the orthogonal center of the tetrahedron *abcd*. The common face *abc* is said to be locally regular if $\pi_z(e) > w_e$.

Lemma 1 *(Regular triangulation) If the vertex set of T(S) contains all non-redundant points from S and all faces of T(S) are locally regular then T(S)=RT(S).*

2.1.1 Basic Characteristics

Similarly to the Delaunay triangulation, the regular triangulation of S is unique if the points of S are in general position. In contrast to DT, it is possible that some point $p \in S$ is not a vertex of $RT(S)$. From the given definition of $T(S)$ it follows that each point $p \in S$ lying on the convex hull of S ($CH(S)$) is a vertex of $RT(S)$ – otherwise the union of tetrahedra would not fulfill $CH(S)$.

Shewchuk [43] proved that in the worst case the number of tetrahedra of $RT(S)$ is $O(n^2)$, where $n = |S|$. This happens if the points of S lie on (or nearby) two nonintersectors. If the points of S are uniformly or nearly uniformly distributed, the expected number of tetrahedra grows with n nearly linearly.

2.1.2 Higher Dimension Embedding

There is a close relation between regular triangulations in d-dimensional space and convex hulls in $(d+1)$-dimensional space. In [4] and [21], this topic is discussed in detail, we restrict our description to 3-dimensional triangulations. First, let us define *a lifted point*. For a point $p = (p_x, p_y, p_z) \in E^3$ with a weight w_p, its lifted point $p^+ \in E^4$ is defined as

$$p^+ = (p_x, p_y, p_z, p_x^2 + p_y^2 + p_z^2 - w_p) \tag{2.2}$$

Assume that P is a paraboloid in 4D given by the equation $f(x, y, z) = x^2 + y^2 + z^2$. A lifted point with a positive weight lies bellow P, a lifted point with a zero weight lies on P and a point with a negative weight lies above P. For a set of weighted points S, we define $S^+ = \{p^+ | p \in S\}$. The following lemma, proved in [21], gives the connection between $RT(S)$ and $CH(S^+)$.

Lemma 2 *Let S be a finite set of weighted points in 3D. The vertical projection of the lower facets of $CH(S^+)$ into 3D gives the tetrahedra of RT(S).*

Let us note that a facet of $CH(S^+)$ in 4D is a tetrahedron and a facet is a lower facet if the hyperplane supporting the facet is non-vertical and $CH(S^+)$ lies vertically above this hyperplane.

Some algorithms exploit the relation described in Lemma 2 to construct $RT(S)$. First, they construct $CH(S^+)$ and then obtain the regular triangulation of S as the vertical projection of the lower facets of $CH(S^+)$. This approach profits from the fact that the computation of a convex hull is a fundamental problem of the computational geometry and effective algorithms are known and well studied. An example of a program computing $RT(S)$ via $CH(S^+)$ is Qhull [6].

Also it is easy to see that $p \in S$ is redundant in $RT(S)$ if and only if its corresponding lifted point p^+ is not a vertex of any lower facet of $CH(S^+)$, see Fig. 2.2.

2.2 Power diagram

Power diagrams ($PD(S)$), also called weighted Voronoi diagrams, are the dual structures of regular triangulations. Because we exploit this duality in the computation of channels in proteins (Section 6.2.3), we give a brief definition of power diagrams in 3D here. The definition in a general dimension and more details can be found in [3] or [21].

Power cell Given a set of weighted points S in E^3. For each weighted point $p \in S$, its power cell is defined as

$$\textit{power cell}(p) = \{x \in E^3 | \forall q \in S - \{p\} : \pi_p(x) \le \pi_q(x)\} \tag{2.3}$$

The point p is the so-called *generator* of *power cell*(p). Observe that *power cell*(p) is a convex polyhedron and the union of all *power cells*(p), $p \in S$, covers E^3. An intersection of two power cells is either empty or forms:

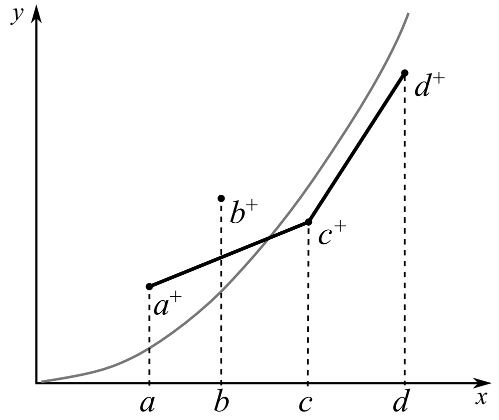


Figure 2.2: Relation between $RT(S)$ in 1D and $CH(S^+)$ in 2D. The weights of a, b are negative, the weights of c, d are positive. The point b^+ lies above the convex hull, therefore b is redundant in $RT(S)$.

- a planar convex polygon – a face of $PD(S)$. A face of $PD(S)$ is the intersection of two power cells.
- a line segment or a half line – an edge of $PD(S)$. An edge of $PD(S)$ is the intersection of at least three power cells.
- a point – a vertex of $PD(S)$. A vertex of $PD(S)$ is the intersection of at least four power cells and the sphere orthogonal to the generators of these power cells is centered in this vertex of $PD(S)$.

Power diagram The power diagram of S is a collection of all power cells, their faces, edges and vertices.

Now let us discuss the duality between $RT(S)$ and $PD(S)$. For $i = 0, ..., 3$, a dual i-dimensional simplex of $RT(S)$ exists for each $(3 - i)$-dimensional object of $PD(S)$:

- a vertex p of $RT(S)$ is the dual of $power\ cell(p)$, p is the generator of $power\ cell(p)$. p is redundant in $RT(S)$ if and only if $power\ cell(p)$ is an empty set.
- an edge of $RT(S)$ is the dual of a face of $PD(S)$ and the edge is perpendicular to the face. Two vertices $p, q \in S$ are connected by an edge in $RT(S)$ if and only if $power\ cell(p)$ and $power\ cell(q)$ share a face.
- a face (a triangle) of $RT(S)$ is the dual of an edge of $PD(S)$. The face is perpendicular to the edge and the edge goes through the orthogonal center of the face.
- a tetrahedron of $RT(S)$ is the dual of a vertex of $PD(S)$. The vertex lies in the orthogonal center of the tetrahedron.

Thanks to the duality, power diagrams could be seen as another representation of regular triangulations and vice versa. Also each of these structures can be derived from the other in linear time. Any computation performed on one of these structures can be also done on the other structure and with the same time complexity. For example, a path in $PD(S)$ – a sequence of vertices and edges – can be represented as a sequence of tetrahedra and faces in $RT(S)$. This will be exploited later in the protein analyses in Section 6.2.3.

2.3 Euclidean Voronoi Diagram

The last data structure of this chapter are Euclidean Voronoi diagrams (EVD). The reason, why we describe them here, lies again in Section 6.2, where Euclidean Voronoi diagrams will be used for a protein analysis.

The formal definition of a cell of EVD is very similar to the definition of a power cell in PD. A sphere is used as a generator instead of a weighted point, and the power distance between a point and a weighted point is replaced by a euclidean distance between a point and a spherical surface.

EVD cell Given a set of spheres S in E^3. For each sphere $s \in S$ with a radius r_s, its EVD cell is defined as

$$EVD\ cell(s) = \{x \in E^3 | \forall s' \in S - \{s\} : |xs| - r_s \leq |xs'| - r_{s'}\}, \tag{2.4}$$

where $|xs|$ denotes the euclidean distance between the point x and the center of s.

The faces, edges and vertices of EVD diagrams can be described as intersections of EVD cells in the same way as in power diagrams. But there is one important difference – a bisector of two generators in PD is a plane, whereas in EVD it is a hyperboloid (or a part of a hyperboloid). In consequence, the geometry of faces and edges of EVD cells is not linear. A face is a connected subset of a hyperboloid, an edge is a conic section. Moreover, faces can contain "holes" and edges can be enclosed ellipses. We recommend Martin Maňák's master thesis [35] as a great source of detail information about EVD, or, e.g. [30].

From the viewpoint of a protein analysis in Section 6.2, the key properties of EVD is that an arbitrary point x of an edge h is equidistant to the spherical surfaces of the three generators defining the edge h and there is no generator whose spherical surface is closer to x.

Chapter 3

Triangulation Construction

Many algorithms of 3D Delaunay triangulation construction are known, most of them can be modified to construct regular triangulations. According to [12], these algorithms can be classified into five groups:

- incremental insertion – points are inserted into the regular triangulation one by one, regularity of the triangulation is restored after each insertion, see e.g. [21], [48].
- incremental construction – $RT(S)$ is constructed by successively building globally regular tetrahedra, see [7].
- higher dimension embedding – $RT(S)$ is obtained as a vertical projection of the lower faces of the convex hull of the lifted points S^+. An example is the Qhull algorithm, see [6].
- divide and conquer – this approach is based on the recursive partition and local triangulation of the point set, and then on a merging phase where the resulting triangulations are joined. An example is the DeWall algorithm [12]. The original DeWall creates Delaunay triangulations but it can be modified for the computation of regular triangulations.
- sweeping – the sweeping algorithms are mainly used in 2D, as an example we can mention Fortune's algorithm for construction of Voronoi diagrams in a plane [24]. The sweeping concept can be extended to higher dimension (e.g. Shewchuk uses sweeping algorithm for the construction of constrained Delaunay triangulations in d-dimension [41]), but an implementation of sweeping algorithms working in higher dimension is often very tricky. We do not know any sweeping algorithm constructing the regular triangulation, nevertheless we think that it is possible to use the sweeping concept to construct the regular triangulation.

The purpose of this section is not to give an exhaustive enumeration of all these algorithms, but to describe the incremental algorithms, with which we have got a personal experience.

3.1 Incremental Flipping Algorithm

In [27], Joe presented an incremental algorithm of 3D Delaunay triangulation construction. Later Edelsbrunner et al. [21] extend Joe's algorithm for a regular triangulation. This algorithm starts with an auxiliary tetrahedron containing all points of S (or alternatively, with a convex hull of S divided into tetrahedra). The points from S are inserted into the triangulation one by one. In each step, a tetrahedron $\mathcal{T}$ containing the point to be inserted is found. Then the point is connected into $\mathcal{T}$ and a regularity of the resulting triangulation is repaired by a sequence of flips. Further the algorithm is described in detail.

3.1.1 Point Location

For location of a tetrahedron containing a given point p, two different approaches can be used. The first is the so called walking algorithm. Its main advantage is that it does not use any auxiliary data structure. The algorithm starts in an arbitrary tetrahedron of $T(S)$ and then traverses $T(S)$ from tetrahedron to tetrahedron until it finds the tetrahedron containing the given point p. In [16], Devillers describes three different walking strategies:

- Straight walk – it visits all tetrahedra along the line segment connecting the starting tetrahedron and the point p (Fig. 3.1a).
- Orthogonal walk – it visits the tetrahedra along the three continuous line segments (each line segment is parallel with one coordinate axis) changing one coordinate at a time (Fig. 3.1b).
- Stochastic visibility walk – from a tetrahedron $\mathcal{T}$ not containing p, the algorithm moves to the neighbor of $\mathcal{T}$ through a *randomly chosen* face f if the plane supporting f separates $\mathcal{T}$ from p (Fig. 3.1c).

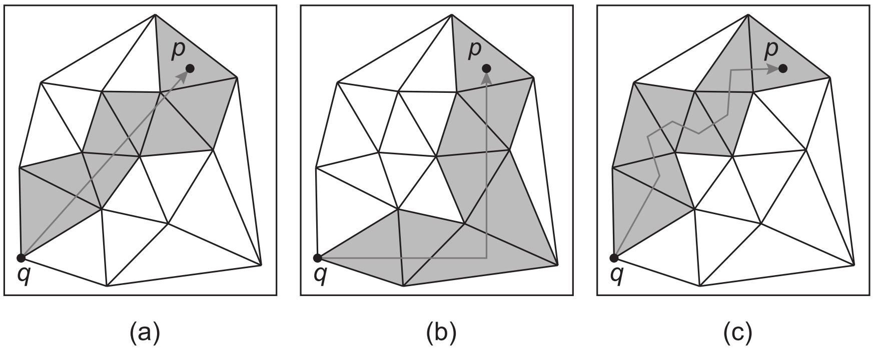


Figure 3.1: Walking algorithms in 2D: (a) straight walk, (b) orthogonal walk, (c) stochastic visibility walk. Images courtesy of Jiří Skála.

All three strategies can be used for a point-location in an arbitrary triangulation. However, Devillers recommends to use the Stochastic visibility walk, because it does not encounter any problem with degenerate cases. Mücke et al. [38] shows that if a tetrahedron, where the walk starts, is chosen "cleverly", then the expected time complexity of the walking algorithm is $O(n^{1/4})$ per one point location, which is slightly worse than optimal $O(\log n)$.

An example of a data structure, which allows a point location in optimal time (in expected case), is the Directed Acyclic Graph (DAG) [13], also called the history dag. DAG maintains the history of changes performed in the triangulation. It has a unique root – the starting auxiliary tetrahedron. The other nodes of DAG are created as follows: if j tetrahedra $\mathcal{T}_i$ of the current triangulation are replaced with k new tetrahedra $\mathcal{T}_i^*$ by a $flip\ jk$, then the k new tetrahedra $\mathcal{T}_i^*$ are connected to DAG as successors of the tetrahedra $\mathcal{T}_i$. So the inner nodes of DAG are the tetrahedra which were destroyed during the triangulation construction and the sinks of DAG store the tetrahedra of the current triangulation. The location of a point p in DAG starts in the root and follows the path of tetrahedra containing p until it reaches a sink of DAG (a tetrahedron of the current triangulation containing p).

As already mentioned, the time complexity of one point location in DAG is $O(\log n)$ in the expected case. DAG has two main disadvantages. First, it consumes $O(n)$ memory in expected case and $O(n^2)$ in the worst case. Second, it is very improper to use DAG in combination with a dynamic triangulation (i.e. a triangulation of a variable set of points – this topic is discussed later in Section 5) – imagine a triangulation of a few points, where one point is repeatedly deleted and inserted. The height of DAG increases with each deletion and insertion, therefore, the time complexity of a location of a point in DAG is dependent not only on the number of points, but also on the number of insertions and deletions a so it can be much worse than a simple search of all tetrahedra. In [46], Vigo et al. describe a data structure based on DAG which allows efficient location, insertion and also deletion of a point, but in our opinion their solution is too tricky.

In [14], Devillers describes another location data structure (it is designated for 2D Delaunay triangulation, but it can be easily extended for a regular triangulation in a general dimension). Devillers's data structure consists of several levels, a level i contains a triangulation of a set $S_i \subseteq S$, where the sets S_i are generated as follows. For each point $p \in S$, a level j is randomly chosen in which the point p firstly appears. Then p is contained in all levels from the level j to the bottom level 1. This bottom level 1 contains all points of S, so it contains the full triangulation of S. In consequence, the following term holds: $S_k \subseteq S_{k-1} \subseteq ... \subseteq S_2 \subseteq S_1 = S$. The point with the highest level determines the number of levels k. Each point $p \in S$ has a link to one incident triangle in each level containing p (so a point which firstly appears in the level j has j links to the incident triangles). The location of the triangle containing a given query point q starts in the highest level k (which contains only a few points and triangles). First, the nearest point $p \in S_k$ of q is found (using an algorithm similar to the straight walk described above). Then a triangle incident to p in the lower level $k-1$ is determined (using the link of p). This triangle is the starting place of the next search of the nearest point of q in the lower level $k-1$. This process repeats, until the algorithm reaches the level 1. In this level, the triangle containing the point q is found. The time complexity of one point location is $O(\log n)$ in the expected case. This data structure allows not only insertion, but also deletion of a point – if a point p should be deleted from S, p is deleted from all levels where it appears. Another possibility is to use a bucketing data structure, see e.g. [45].

3.1.2 Flips

A flip is a well-known operation introduced by Lawson (see [27]). Flips can be classified as non-degenerate and degenerate. First, we define the non-degenerate flips – here we make an exception and give the definition of flips in general dimension d. Degenerate flips are described later and only in 3D.

Non-degenerate Flips

Given the set Q of $d+2$ points in E^d in general position, the convex hull of Q can be triangulated with exactly two different triangulations. If one of these triangulations consists of k simplices, the other triangulation has exactly $d+2-k$ simplices (see [31]). The operation replacing one of these triangulations with the other is called a flip. Flips are denoted by the number of simplices of $T(Q)$ before and after the flip. So, back in 3D, four types of non-degenerate flips exist (see Fig. 3.2):

- $flip$ 14 – replaces 1 tetrahedron by 4. It inserts a point into the triangulation.
- $flip$ 41 – replaces 4 tetrahedra by 1. It removes a point from the triangulation.
- $flip$ 23 – replaces 2 tetrahedra by 3. It destroys an inner face shared by the two flipped tetrahedra and replaces it by three new inner faces.
- $flip$ 32 – replaces 3 tetrahedra by 2. It destroys three inner faces, each shared by two of three flipped tetrahedra, and replaces them by one new inner face.

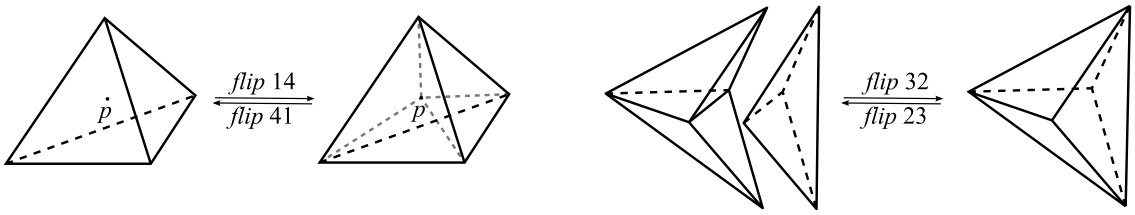


Figure 3.2: Flips.

Clearly, the $flip$ 14 is the inverse of the $flip$ 41 and the $flip$ 23 is the inverse of the $flip$ 32. Except the $flip$ 14, the described flips are used for the repairing of locally non-regular faces. But not every face can be flipped. Now we give the conditions of flippability as described in [21].

Flippability

Let f be a face shared by two tetrahedra $\mathcal{T}_1$ and $\mathcal{T}_2$ and let e_1, e_2, e_3 be the edges of f. An edge e_i is called convex if there is a plane that contains e_i and $\mathcal{T}_1$ and $\mathcal{T}_2$ lie on the same side of the plane. Otherwise the edge e_i is called reflex. The face f is flippable (i.e. can be transformed by a flip) if all edges of f are convex or if each reflex edge e_i is incident to exactly three tetrahedra.

Note that if a face is flippable, then the number of reflex edges determine the type of flip to be used: a face without reflex edges is flippable with the $flip$ 23, a face with one reflex edge is flippable with the $flip$ 32 and a face with two reflex edges is flippable with the $flip$ 41.

Degenerate Flips

If the points in S are not in general position, the above mentioned flips can create flat tetrahedra in $RT(S)$. In most cases, this is unacceptable. One solution is to simulate a general position using a perturbation method (see [20]). Other popular solution (used e.g. in [27] and [42]) is to use degenerate flips to deal with points in non-general position. In 3D, three degenerate flips are used (see Fig. 3.3):

- $flip$ 26 is used, if the point p to be inserted lies in a face f. Two tetrahedra (sharing the face f) are replaced with six new tetrahedra, each incident to p.
- $flip$ $m2m$ is used, if the point p to be inserted lies in an edge e. The edge e is divided and also each tetrahedron $\mathcal{T}$ containing e is divided in two new tetrahedra, so m tetrahedra containing the edge e are replaced with $2m$ new tetrahedra.
- $flip$ 44 – let $abcd$ and $abce$ be two tetrahedra with a shared face abc. Let us suppose that the vertices b, c, d, e are coplanar (thus abc cannot be flipped by the $flip$ 23, because it would create the flat tetrahedron $bcde$.) If there exist two neigbouring tetrahedra $bcdf$ and $bcef$, then the face abc can be flipped by the $flip$ 44. This flip replaces the two faces abc and bcf by two faces ade and def.

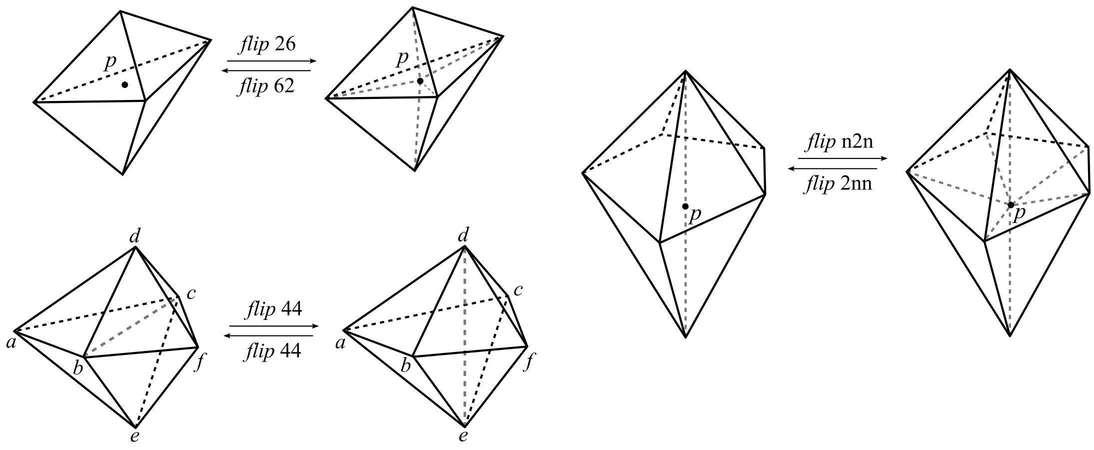


Figure 3.3: Degenerate flips.

3.1.3 Predicates

The incremental flipping algorithm uses two basic tests - an orientation test and a regularity test. Both these tests can be realized as a determinant computation (see [22]). The orientation test of four points a, b, c, d checks the mutual position of the point d and a plane ϱ passing through the points a, b, c.

$$orient(a, b, c, d) = \det \begin{pmatrix} a_x & a_y & a_z & 1 \\ b_x & b_y & b_z & 1 \\ c_x & c_y & c_z & 1 \\ d_x & d_y & d_z & 1 \end{pmatrix} \tag{3.1}$$

If the result of the test $orient(a, b, c, d)$ is positive, negative, respectively, the point d lies above, bellow, respectively, the plane ϱ. If the result is equal to zero, d lies in the plane ϱ.

The regularity test of five weighted points a, b, c, d, e checks, whether the face abc of the tetrahedron $abcd$ is locally regular with respect to the point e. This test can be also seen as the orientation test of the lifted points a^+, b^+, c^+, d^+, e^+ in E^4. In the following equation, $a_t = a_x^2 + a_y^2 + a_z^2 - w_a$ (b_t, c_t, d_t and e_t are computed in the same way).

$$regular(a,b,c,d,e) = \det\begin{pmatrix} a_x & a_y & a_z & a_t & 1 \\ b_x & b_y & b_z & b_t & 1 \\ c_x & c_y & c_z & c_t & 1 \\ d_x & d_y & d_z & d_t & 1 \\ e_x & e_y & e_z & e_t & 1 \end{pmatrix} \cdot orient(a,b,c,d) \tag{3.2}$$

If the result of the test $regular(a, b, c, d, e)$ is negative, the face abc is locally regular. Otherwise is non-regular.

A special case of the regularity test is the so-called *in_sphere* test. This test ignores the weights of points and so it is computed as follows.

$$in\_sphere(a,b,c,d,e) = \det\begin{pmatrix} a_x & a_y & a_z & a_x^2+a_y^2+a_z^2 & 1 \\ b_x & b_y & b_z & b_x^2+b_y^2+b_z^2 & 1 \\ c_x & c_y & c_z & c_x^2+c_y^2+c_z^2 & 1 \\ d_x & d_y & d_z & d_x^2+d_y^2+d_z^2 & 1 \\ e_x & e_y & e_z & e_x^2+e_y^2+e_z^2 & 1 \end{pmatrix} \cdot orient(a,b,c,d) \tag{3.3}$$

If the result of the test $in\_sphere(a, b, c, d, e)$ is negative, positive, respectively, the point e lies outside, inside, respectively, the circumscribed sphere of the points a, b, c, d. If the result is equal to zero, e lies on the circumscribed sphere. The *in_sphere* test is used in algorithms for Delaunay triangulation construction.

These determinant tests are relatively robust, nevertheless floating point errors can cause that the determinant test returns an incorrect result. One possible solution of this problem is to use libraries for an arbitrary precision floating-point arithmetic or for an adaptive precision floating-point arithmetic – we will call them exact arithmetics for short. These libraries should guarantee that the result is always correct, but at the cost of slower evolution of the tests. In our implementation, we use Shewchuk's adaptive precision floating-point arithmetic [40] for the computation of the orientation and in_sphere tests[1].

3.1.4 Algorithm

Now we are finally able to describe Edelsbrunner's algorithm of incremental insertion (see [21]). As already mentioned, the points are inserted into the triangulation one by one. Let p be the point to be inserted. First, the tetrahedron $\mathcal{T}$ containing p is located (using a walking algorithm or a location data structure). The tetrahedron $\mathcal{T}$ is checked – if $\mathcal{T}$ is regular with respect to the point p, then p is redundant and the algorithm continues with another point. If p is not redundant, p is connected into $\mathcal{T}$ by the $flip$ 14 (if p lies inside $\mathcal{T}$), by the $flip$ 26 (if p lies in a face of $\mathcal{T}$), or by the $flip\ m2m$ (if p lies in an edge of $\mathcal{T}$). In consequence, some faces in the triangulation can become non-regular. Edelsbrunner et al. proved that it is sufficient to check only regularity of the *link* faces of p (a face f is a link face of p, if f is a face of some tetrahedron containing p but f does not contain p, see Fig. 4.1). So all the link faces of p are checked – if a flippable and non-regular face is found, it is flipped by a proper

[1] Shewchuk's C-code of the predicates can be downloaded from http://www.cs.cmu.edu/~quake/robust.html

flip. All the link faces of p of the newly created tetrahedra have to be also checked. When no non-regular link face of p exists, the triangulation is regular and the algorithm continues with another point. We summarize this in the Algorithm 3.1.

Algorithm 3.1: Incremental flipping insertion

Input: Set of weighted points S
Output: $RT(S)$
$L \leftarrow$ the empty stack of faces;
for *each point p in S* **do**
    Locate the tetrahedron $\mathcal{T}$ containing p;
    if *$\mathcal{T}$ is not regular with respect to p* **then**
        Connect p into $\mathcal{T}$ by $flip$ 14 (by $flip$ 26 or $flip$ $n2n$ in a degenerate case);
        Push each face f of the new tetrahedra, f is a link face of p, into L;
        while *L is not empty* **do**
            Pop a face f from L;
            if *f exists, is not regular and is flippable* **then**
                Flip the face f;
                Push each face f' of the new tetrahedra, f' is a link face of p, into L;
            end
        end
    end
end

The time complexity of the described algorithm is $O\big(n^2\big)$ in the worst case. In the average case, the time complexity is $O(n\ log\ n)$, $O\big(n^{5/4}\big)$, respectively, if a location data structure, a walking algorithm, respectively, is used for a point location.

3.2 Bowyer-Watson Algorithm

Another incremental algorithm is known as the Bowyer-Watson algorithm. In 1981, Adrian Bowyer [8] and David F. Watson [48] devised and published it independently in the same issue of The Computer Journal. The algorithm computes Delaunay triangulations, but can be easily generalized for the computation of regular triangulations by adding weights to the input points and by replacing the in_sphere test by the test of regularity. The algorithm works in d-dimension and its expected time complexity is $O\big(n^{(2d-1)/d}\big)$. For simplicity, we describe the algorithm in 3D.

The algorithm starts with one auxiliary tetrahedron containing all the input points from the given set S (in the same way as the incremental flipping algorithm), or alternatively, with a convex hull of S divided into tetrahedra. The points of S are inserted into the triangulation one by one. In each step, all tetrahedra whose circumscribed spheres contain the point p to be inserted (the so-called conflicting tetrahedra) are removed from the triangulation. This creates a cavity inside the triangulation – a star-shaped polyhedron $\mathcal{P}$ with triangular faces. The polyhedron $\mathcal{P}$ is then retriangulated using the new point p – each face of $\mathcal{P}$ is connected with p, together forming a new tetrahedron of the triangulation (see Fig. 3.4).

Note that the tetrahedra conflicting with the given point p can be found efficiently. First, a tetrahedron $\mathcal{T}$ containing p is found, using e.g. the walking algorithm. In the Delaunay

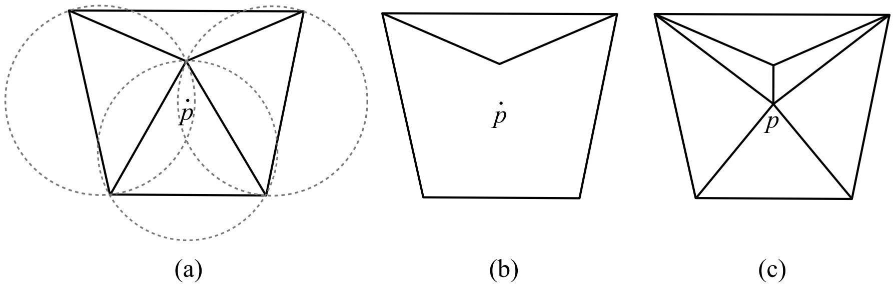


Figure 3.4: The Bowyer-Watson algorithm in 2D – insertion of the point p: (a) the conflicting triangles are marked by the circumscribed circle, (b) the triangulation with the star-shaped cavity, (c) the resulting triangulation.

triangulation, $\mathcal{T}$ is always the conflicting tetrahedron of p. In the regular triangulation, $\mathcal{T}$ can be non-conflicting, i.e. $\mathcal{T}$ can be regular with respect to p. This means that p is redundant and is not inserted into $RT(S)$. Because the conflicting tetrahedra form one continuous polyhedron, they can be found by a simple breadth-first or depth-first search. The search starts in the conflicting tetrahedron $\mathcal{T}$. Then in each step, it explores one neighbor tetrahedron $\mathcal{T}^*$ of the currently known conflicting tetrahedra. If the circumscribed sphere of $\mathcal{T}^*$ contains p, then $\mathcal{T}^*$ is added to the known conflicting tetrahedra.

3.3 Incremental Construction

The algorithm of incremental construction (see Beyer et al. [7]) builds $RT(S)$ a tetrahedron by a tetrahedron. Once a tetrahedron is created, it is never changed in the future (in contrast to the two previous incremental algorithms). The first step is to create a globally regular tetrahedron (or a globally regular triangle). All faces of this tetrahedron are inserted into an active face list (AFL). For each face f in the AFL, the point $p \in S$ minimizing the signed Delaunay distance from f is found. Note that this search considers only points lying in the half space which is given by the plane supporting the face f and does not contain the tetrahedron generating the face f. The signed Delaunay distance of a weighted point p from a face abc is defined as

$$SD(p, abc) = (z_p - z_{abc}) \cdot \frac{(a-b) \times (a-c)}{||(a-b) \times (a-c)||}, \tag{3.4}$$

where z_p is the orthogonal center of the weighted points p, a, b, c (see the definition in 2.1) and z_{abc} is the orthogonal center of the weighted points a, b, c (z_{abc} lies in the plane supporting the face abc).

After the point p minimizing the signed Delaunay distance from the face f is found, f is expanded – the point p is connected with f, thereby a new globally regular tetrahedron $\mathcal{T}$ is created. Each face f' of $\mathcal{T}$ is checked – if f' has already been in the AFL, f' is removed from the AFL and the adjacency between the new tetrahedron $\mathcal{T}$ and the "older" tetrahedron containing f' is established, otherwise f' is added into the AFL. If no point which can expand f is found, then f lies on the $CH(S)$ and is removed from the AFL. The algorithm continues until the AFL is empty.

The time complexity of this algorithm is $O(n^3)$. According to Beyer et al., the time complexity in expected case can be significantly improved if a few speed-up techniques are used. First, the check if a new face is already in the AFL can be done in expected time $O(1)$ using a hash table. Second, the location of the point minimizing the signed Delaunay distance can be accelerated by using a uniform grid. The second speed-up technique can be used only if the points are uniformly distributed and the weights of points are relatively small with respect to the average distance between the points.

3.4 Comparison

The algorithm of incremental construction is easy to implement, but it has a few serious drawbacks. First, it does not allow to insert a new point into $RT(S)$ after the triangulation is created (if the new point lies inside $CH(S)$). The second drawback is its numerical stability. By our experience, if no exact arithmetic is used, the algorithm often fails even in 2D – it creates intersecting triangles.

Both the Bowyer-Watson algorithm and the incremental flipping algorithm allow the on-line construction of triangulation if an approximate area of all points to be inserted is known. The advantage of the Bowyer-Watson algorithm in comparison with the incremental flipping algorithm is its simpler implementation. And according to our tests, Bowyer-Watson algorithm is also faster than the incremental flipping. For example, Bowyer-Watson is about 20%–25% faster for a set of points with Gaussian distribution, see Fig. 3.5.

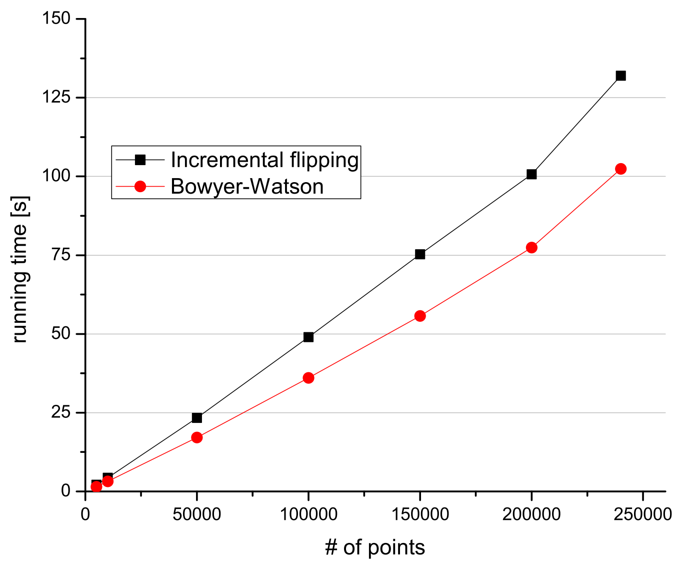


Figure 3.5: Time of the construction of $RT(S)$.

The drawback of Bowyer-Watson algorithm is its lack of numerical robustness. According to our experience, the Bowyer-Watson is much more robust than the incremental construction, but still less robust than the incremental flipping. If the test of regularity numerically fails in Bowyer-Watson, then the conflicting tetrahedra can form a non-star-shaped polyhedron $\mathcal{P}$. If this polyhedron $\mathcal{P}$ is retriangulated as described in 3.2, some of the new tetrahedra intersect some of the tetrahedra lying outside $\mathcal{P}$, see Fig. 3.6. In contrast, if the regularity test numerically fails in the incremental flipping, the resulting triangulation can contain non-regular tetrahedra, but no mutually intersecting tetrahedra. These can appear only if the orientation test numerically fails. Orientation tests are computed as a sign of a determinant of a matrix 4x4 and regularity tests as a sign of a determinant of a matrix 5x5, therefore, we can expect that orientation tests are more reliable.

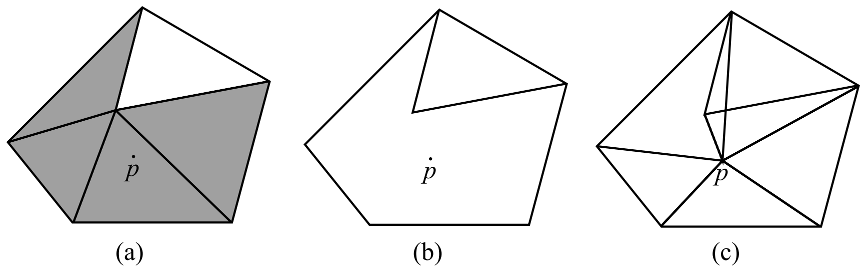


Figure 3.6: An error in the Bowyer-Watson algorithm in 2D. (a) The triangles which are conflicting with the point p according to the regularity tests are shaded. The upper triangle has been mistakenly marked as non-conflicting. (b) The triangulation without the conflicting triangles. (c) The resulting (bad) triangulation.

Chapter 4

Triangulation Destruction

As showed in the previous chapter, many algorithms are known for the construction of 3D regular triangulations (and even more for the construction of Delaunay triangulations). Many of them work incrementally, inserting points one by one. But only few algorithms are focused on the "inverse" problem – the deletion of vertices in 3D RT or DT, and as far as we know, all algorithms for deletion allow to delete only one point in time – if more points have to be removed, they are deleted successively.

So in this chapter, we are focused on obtaining $RT(S\text{-}\{p\})$ from $RT(S)$ by a deletion of a point p. Before we can continue, a few new definitions are needed:

Star(p) A set of tetrahedra incident to p in $RT(S)$ is denoted as $star(p)$, $deg(p)$ denotes a number of tetrahedra in $star(p)$.
Star face A face f is a *star face of* p if f is a face of some tetrahedron $\mathcal{T} \in star(p)$, and f contains p.
Link face A face f is a *link face of* p if f is a face of some tetrahedron $\mathcal{T} \in star(p)$, and f does *not* contain p.
Polyhedron $\mathcal{P}(p)$ A star-shaped polyhedron formed by the link faces of p is denoted as $\mathcal{P}(p)$, *the initial* $\mathcal{P}(p)$ denotes $\mathcal{P}(p)$ before the beginning of a deletion of p.
Ear An *ear* ε of polyhedron $\mathcal{P}$ is a set of 2 or 3 neighboring faces of $\mathcal{P}$, which determine a tetrahedron $\mathcal{T}$ (we say that ε defines $\mathcal{T}$). An ear containing exactly two faces of $\mathcal{P}$ is called a *2-ear*, and an ear sharing exactly three faces with $\mathcal{P}$ is called a *3-ear* (see Fig. 4.2). An ear ε is called *valid* if the tetrahedron defined by ε lies inside $\mathcal{P}$. Note that each ear of $\mathcal{P}$ is given by exactly four vertices of $\mathcal{P}$. Cutting of an ear ε means to create a tetrahedron $\mathcal{T}$ defined by the ear ε, and to connect $\mathcal{T}$ to the triangulation outside $\mathcal{P}$. In consequence, the polyhedron $\mathcal{P}$ shrinks.

Now let us discuss the problem of point deletion. The first question can be, how much $RT(S)$ changes, if one point is deleted. Lemma 3 gives a partial answer (the idea of this lemma can be found in [17] or [33]).

Lemma 3 *If points in S are in a general position, then it is always possible to delete an arbitrary point $p \in S$ in RT(S) without modifications of any tetrahedron not containing p (i.e. if $\mathcal{T} \in RT(S)$ and $p \notin \mathcal{T}$, then $\mathcal{T} \in RT(S\text{-}\{p\})$).*

Proof It is well known that a regular triangulation of points in general position is unique. Consider the triangulation $RT(S\text{-}\{p\})$, points in S are in a general position. If p is inserted

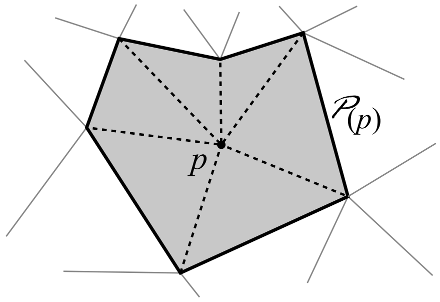
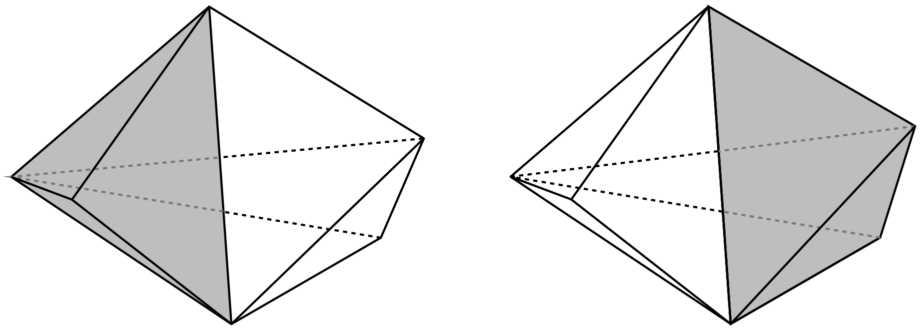


Figure 4.1: Definitions in 2D: “tetrahedra” of $star(p)$ are shaded, star “faces” are dashed, link “faces” are black solid and together form $\mathcal{P}(p)$.



(a) (b)

Figure 4.2: Ears (shaded) of a polyhedron. (a) a 3-ear, (b) a 2-ear.

into this triangulation by the Bowyer-Watson algorithm (see 3.2), all tetrahedra in $RT(S\text{-}\{p\})$ which are non-regular with respect to p are deleted, and the “hole” is retriangulated with new tetrahedra *all incident to* p. Note that the tetrahedra in $RT(S\text{-}\{p\})$ which are regular with respect to p remain unchanged. Now if p is immediately deleted in $RT(S)$, then the resulting triangulation have to be identical with the “original” triangulation $RT(S\text{-}\{p\})$, and so only tetrahedra incident to p have to be modified. Because $RT(S)$ is not affected by the order of insertion of points, we can generalize this idea for any point in S.□

Unfortunately, Lemma 3 does not hold generally – if points in S are in non-general position, then it is possible that p cannot be deleted in $RT(S)$ without changes of one or more tetrahedra not containing p. Consider the triangulation shown in Fig. 4.3a. Let the points a, b, c, d, p be cospherical and the points a, b, c, d coplanar. At this moment, p can be deleted without a change of a tetrahedron not containing p. Now we insert a point q into the triangulation. Assume that all tetrahedra incident to p are regular with respect to q, but in consequence of a non-general position, the faces acp and acq are flipped by the $flip$ 44. The result is shown in Fig. 4.3b (without the point q). Now p cannot be “easily” deleted – if all tetrahedra incident to p are removed, the resulting “hole” is the Schönhardt polyhedron, which cannot be triangulated (see Fig. 4.3c). So to delete p, some edge of the polyhedron $\mathcal{P}(p)$ has to be flipped, i.e. some tetrahedra outside $\mathcal{P}(p)$ have to be modified.

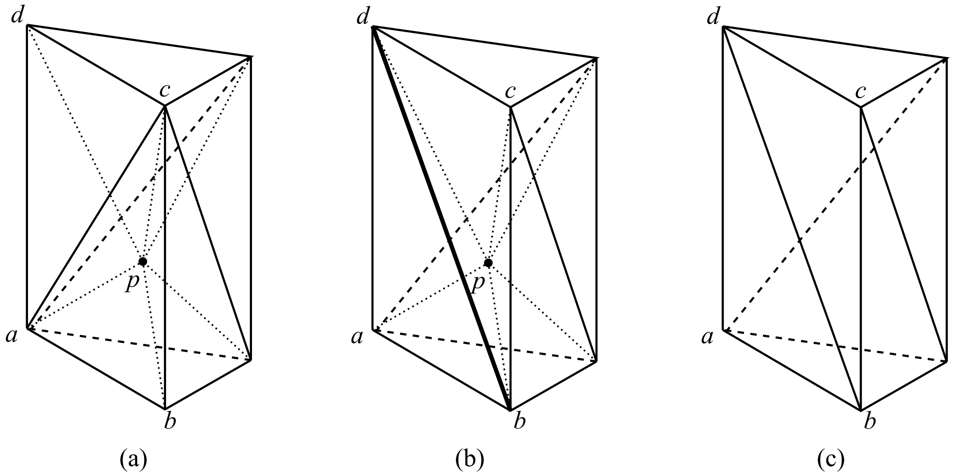


Figure 4.3: (a) triangulation after the insertion of p, (b) p cannot be deleted without a flip of an edge of the polyhedron $\mathcal{P}(p)$, (c) Schönhardt polyhedron.

Now let us discuss methods for point deletion. Three approaches can be distinguished:

- All tetrahedra incident to p are removed from $RT(S)$ and the created star-shaped polyhedron $\mathcal{P}(p)$ is retriangulated by cutting ears.
- The $deg(p)$ is reduced to 4 by a sequence of flips and then p is removed by the $flip$ 41.
- All tetrahedra incident to p are removed from $RT(S)$. A triangulation of vertices of $\mathcal{P}(p)$ is computed separately, and then $\mathcal{P}(p)$ is "filled" with this triangulation.

Further, three algorithms for point deletion are described, each uses a different approach. The algorithms are originally designed for a deletion in the Delaunay triangulations. But they can be easily modified to work in a regular triangulation (in fact, it is sufficient to replace the *in_sphere* test by the regularity test). We design and present the modified versions.

4.1 Cutting of Ears

In [15], Devillers described an algorithm of the point deletion in two-dimensional DT. Later, Devillers and Teillaud [16] extend this algorithm to 3D. The main idea of the algorithm is as follows. All tetrahedra which are incident to a point p to be deleted are removed. It creates a "hole" in the triangulation – a star-shaped polyhedron $\mathcal{P}(p)$. The goal is to retriangulate $\mathcal{P}(p)$ in such a way that the resulting triangulation is regular. Devillers proved that this can be done by successive cutting of valid ears in a particular order – in each step, a valid ear ε satisfying a certain condition (see bellow) is found and cut, i.e. a tetrahedron $\mathcal{T}$ defined by the ear ε is created, and connected to the triangulation (and in consequence, the polyhedron $\mathcal{P}(p)$ shrinks).

For the choosing of an ear to be cut, Devillers offers two different conditions – we call them *the condition of minimal ear-power* and *the condition of global regularity*.

4.1.1 Condition of Minimal Ear-power

This condition was introduced by Devillers in [15] for a deletion in 2D Delaunay triangulation, but it can be extended to 3D. First, let us define the ear-power of an ear ε with respect to a point q as

$$ear\text{-}power(\varepsilon, q) = -\pi_z(q), \tag{4.1}$$

where z is the orthogonal center of the tetrahedron defined by the ear ε and $\pi_z(q)$ is the power distance defined in the equation 2.1. The condition of minimal ear-power is based on the following Lemma 4, proved by Devillers in [15].

Lemma 4 *Let $\mathcal{E}$ be a set of all valid ears of $\mathcal{P}(p)$. If the ear $\varepsilon\in\ \mathcal{E}$ with the minimal ear-power(ε,p) is cut off, then the tetrahedron $\mathcal{T}$ defined by ε is globally regular in RT(S-$\{p\}$).*

By this lemma, a valid ear whose ear-power is minimal with respect to the point p should be cut. This can be done repeatedly, until $\mathcal{P}(p)$ is retriangulated. This implies the following algorithm, summarized in the pseudocode 4.1.

First, all ears of $\mathcal{P}(p)$, no matter if valid or not, are found. A priority of a valid ear ε is set to *ear-power*(ε,p), a priority of a non-valid ear is set to $+\infty$. All the ears are inserted into a priority queue L according to their priority. Second, the ear with a minimal priority is removed from the queue L and it is cut off. The remaining ears in L and their priorities are updated (see bellow). This process is repeated until L contains only four ears. These last four ears are retriangulated with one tetrahedron.

The algorithm looks simple, but the updating of ears (after a cut of an ear) is tricky to implement. If an ear ε is cut, some ears perish, and must be removed from L, some new ears arise, and have to be inserted into L and even some 2-ears can change to 3-ears (and vice versa), and have to be modified in L.

Let k be the total number of cut-off ears. A cut of one ear takes $O(1)$ time. After a cut of an ear, $O(1)$ ears have to be modified and the update of the priority queue L takes $O(log\ k)$ time. So the time complexity is $O(k\ log\ k)$. In expected case, the number of cut-off ears k is $O(deg(p))$. However, in the worst case, the number of ears k can be $O\big((deg(p))^2\big)$ – but this happens only for specially generated set of points and is highly improbable in practice.

4.1.2 Condition of Global Regularity

The condition of global regularity is based on the following Lemma 5 (proved in [17]).

Lemma 5 *Let ε be a valid ear of $\mathcal{P}(p)$. If the tetrahedron $\mathcal{T}$ defined by ε is regular with respect to all vertices (weighted points) of $\mathcal{P}(p)$, then the tetrahedron $\mathcal{T}$ defined by ε is globally regular in RT(S-$\{p\}$).*

By this condition, a valid ear ε of $\mathcal{P}(p)$ is cut, if the tetrahedron $\mathcal{T}$ defined by ε is regular with respect to all vertices of $\mathcal{P}(p)$. So instead of maintaining the priority queue, the ears are stored in a list, and regularity of each ear with respect to all vertices of $\mathcal{P}(p)$ is tested. Anytime a valid ear fulfilling the condition of global regularity is found, it is cut off and ears in the list are updated (some ears perish, and must be removed from the list, some new ears arise, and have to be inserted into the list and even some 2-ears can change to 3-ears and vice versa).

Algorithm 4.1: Cutting of ears with minimal ear-power

Input: $RT(S)$, a point $p \in S$
Output: $RT(S\text{-}\{p\})$
$L \leftarrow$ the priority queue;
for *each ear* ε *of* $\mathcal{P}(p)$ **do**
  if ε *is valid then* **then**
    prior$\leftarrow$ *ear-power*(ε, p);
  end
  else
    prior$\leftarrow +\infty$;
  end
  L.Insert(ε, prior);
end
while $L.count > 4$ *do* **do**
  $\varepsilon_{min} \leftarrow L$.Minimum();
  Cut(ε_{min});
  Update ears and their priorities in L with respect to ε_{min};
end
Triangulate the remaining $\mathcal{P}(p)$ with one tetrahedron;

This yields to the time complexity $O(k\ t)$, where k denotes the number of cut off ears and t the number of vertices of $\mathcal{P}(p)$.

4.1.3 Handling Points in Non-general Position

As already mentioned, if the points are in non-general position, it is possible that $\mathcal{P}(p)$ cannot be triangulated. Devillers et al. [17] briefly discuss, how to modify an algorithm of incremental insertion to avoid a rise of such non-triangulated polyhedra. Further, they describe a perturbation of the in_sphere test guaranteeing that a triangulation after the deletion of a point p is exactly the same as p has never been inserted (and therefore any other point q can be deleted without modifications of tetrahedra outside $\mathcal{P}(q)$). Note that this can be used only if the algorithm is based on the condition of global regularity.

4.2 Flipping of Ears

The second method of a point deletion uses a different approach. Instead of removing all tetrahedra incident to a point p to be deleted and a retriangulation of the created hole, a sequence of flips is used to decrease $deg(p)$ to 4 and then p is removed by the $flip$ 41. An algorithm based on this method is described by Ledoux et al. in [33].

Assume that p is the point to be deleted. Analogously to the Devillers's algorithm, a list of ears of $\mathcal{P}(p)$ is maintained. But in this algorithm the tetrahedra inside $\mathcal{P}(p)$ (i.e. the tetrahedra incident to p) are not removed from $RT(S)$, and ears are not cut but *flipped.* Two different flips of two different types of ears can be distinguished:

- Let ε be a 2-ear given by two faces *abc* and *bcd* of $\mathcal{P}(p)$. This means that two tetrahedra *abcp* and *bcdp* (incident to p) exist in the triangulation. To flip ε means to flip these two tetrahedra by the $flip$ 23. It creates three tetrahedra – one tetrahedron not incident to p and two incident to p. Not every 2-ear can be flipped – the ear ε is said to be flippable if and only if the two tetrahedra *abcp* and *bcdp* can be flipped with the $flip$ 23 (see 3.1.2), i.e. if the union of these two tetrahedra is a convex polyhedron.

- Let ε be a 3-ear given by three faces *abc*, *acd*, and *adb* of $\mathcal{P}(p)$. This means that three tetrahedra *abcp*, *acdp*, and *adbp* (incident to p) exist in the triangulation. To flip ε means to flip these three tetrahedra by the $flip$ 32. It creates two tetrahedra – one not incident to p and one incident to p. Again, not every 3-ear can be flipped – the ear ε is said to be flippable if and only if the three tetrahedra *abcp*, *acdp*, and *adbp* can be flipped with the $flip$ 32 (see 3.1.2), i.e. if the union of these three tetrahedra is a convex polyhedron.

So each flip of an flippable ear modifies the tetrahedra in $star(p)$, and creates one tetrahedron which is not incident to p. The algorithm does not flip each flippable ear – a flippable ear ε is flipped if the tetrahedron $\mathcal{T}$ defined by ε is regular with respect to all vertices (weighted points) of $\mathcal{P}(p)$, i.e. if $\mathcal{T}$ fulfills the condition of global regularity.

The ears are not processed in a particular order. If an ear ε is flippable and fulfills the condition of global regularity, ε is flipped, and the ears in the list are updated (some ears perish, some new ears arise, and some 2-ears can change to 3-ears and vice versa). If ε is not flippable or does not fulfills the condition of global regularity, the next ear in the list is tested. After a number of flips, only four ears remain in the list. That means only four tetrahedra are incident to p and p can be deleted with the $flip$ 41.

Ledoux et al. claim, but without proof, that if ears are flipped according to this strategy and points in S are in general position, then a flippable ear fulfilling the condition of global regularity always exists. This means that the algorithm cannot get stuck with a triangulation, where each ear fulfilling the condition of global regularity is non-flippable.

The method is summarized in the Algorithm 4.2. The time complexity of the algorithm is $O(k\ t)$, where k denotes the number of flipped ears and t the number of vertices of $\mathcal{P}(p)$.

Algorithm 4.2: Flipping of ears

```
Input: RT(S), a point p ∈ S
Output: RT(S-{p})
L ← list of all ears of P(p);
while L.count > 4 do do
    ε ← L.FirstNonTestedEar();
    if ε is flippable then
        T ← tetrahedron defined by ε;
        if T is regular with respect to all vertices of P(p) then
            Flip ε with the proper flip;
            Update ears in L with respect to ε;
        end
    end
end
```

4.2.1 Handling Points in Non-general Position

As discussed in the beginning of this chapter, if points in S are not in general position, $RT(S)$ is not unique and it is possible that a point $p \in S$ cannot be deleted without modifying some tetrahedra outside $\mathcal{P}(p)$. Ledoux et al. offer two approaches to solve this problem. The first is based on a perturbation (similar to the Devillers's approach, described in 4.1) – the perturbed in_sphere test is used both for the triangulation construction and the deletion.

The second approach is based on recovering from untriangulatable $\mathcal{P}(p)$. By Ledoux's experience, if $\mathcal{P}(p)$ cannot be triangulated, then $\mathcal{P}(p)$ always contains two neighboring coplanar faces, whose diagonal can be flipped with the $flip$ 44 – this flip modifies two tetrahedra outside $\mathcal{P}(p)$ and two tetrahedra inside $\mathcal{P}(p)$. After one or more these flips, the resulting $\mathcal{P}(p)$ can be triangulated.

4.3 Sewing

The third method of a point deletion is significantly different from the previous two – instead of flipping or cutting of ears, the triangulation of a "hole" is computed separately and then the hole is filled with this triangulation. Let p be the point to be deleted. The algorithm works as follows:

1. All tetrahedra incident to p (tetrahedra inside $\mathcal{P}(p)$) are removed from $RT(S)$.
2. A "stand alone" regular triangulation $RT(S^*)$ of vertices (weighted points) of $\mathcal{P}(p)$ is computed.
3. In $RT(S^*)$, all tetrahedra which are outside $\mathcal{P}(p)$ are removed.
4. The resulting $RT(S^*)$ is "sewn" into $\mathcal{P}(p)$ in $RT(S)$.

The idea of this method is known (e.g. Devillers mentioned it in [17]), but as far as we know, no article is devoted to it, and no one discussed this method in detail – therefore the rest of our description is slightly fuzzy.

Let k be the number of vertices of $\mathcal{P}(p)$. The time complexity of the creation of $RT(S^*)$ is $O(k^2)$ in the worst case. The steps 3 and 4 can be probably also done in $O(k^2)$. $O(k^2)$ is also the number of tetrahedra, which are necessary for a re triangulation of $\mathcal{P}(p)$ in the worst case, therefore, the algorithm is worst-case optimal. But much work is done uselessly – many tetrahedra created in the step 2 are removed in the step 3.

If five or more points in S are orthogonal to the same weighted point (an analogy to cospherical points in DT), then the described algorithm can fail – e.g. a tetrahedron in $RT(S^*)$ can intersect $\mathcal{P}(p)$ and thus $RT(S^*)$ cannot be sewn into $\mathcal{P}(p)$ without some modifications. Again, this can be solved by using a perturbation technique.

4.4 Comparison

According to our experience, if an exact arithmetic is not used for the regularity tests, the flip-based algorithm is more robust than the algorithm of cutting of ears. If a regularity test numerically fails in the algorithm of cutting of ears, an ear to be cut can be chosen badly. In consequence, the polyhedron $\mathcal{P}(p)$ can degenerate into two polyhedra, which share only one edge. In such a case the algorithm fails and the triangulation is damaged. In contrast, the flip-based algorithm keeps a complete triangulation and if a regularity test numerically fails, a resulting triangulation could be non-regular but it is still valid, i.e. tetrahedra do not intersect each other and the topology is correct. If someone wants to use the algorithm of cutting of ears, we recommend to base the algorithm on the condition of global regularity, although using the condition of minimal ear-power yields to an algorithm with better time complexity. The condition of minimal ear-power was introduced for a deletion in 2D, and although it can be theoretically extended to 3D, it is not robust.

Chapter 5

Triangulation of Variable Data

In this chapter we focus on triangulations of a variable set of points. Such triangulations are useful in many areas. For example, they are used for a collision detection and a path planning in a non-static environment. Also they are useful for a simulation of some environmental processes or modeling of cells in tissues. Last but not least they can be used for a tracking of channels in dynamic proteins. According to the movements of points, two types of triangulations are typically distinguished:

- Kinetic triangulations – points move along continuous curves.
- Dynamic triangulations – points appear, disappear or discreetly change their position.

So in a kinetic triangulation, a position of each point is changing continuously in time. This continuous movement of points causes a sequence of discrete changes of the topology of the triangulation, each change happens in a particular moment. To maintain a kinetic triangulation means to detect successively these moments and to update the topology correctly.

The situation in a dynamic triangulation is different. The moments of changes are given, and to maintain a dynamic triangulation means to handle these changes.

5.1 Kinetic Triangulation

Several authors address the problem of kinetic Delaunay triangulations in three or more dimensional space, let us mention Schaller [39], Ledoux [32] and first of all Albers. Albers has published several papers about the kinetic Delaunay triangulation, in this section we focus of his algorithm described in [1]. A generalization of this algorithm for regular triangulations is not straightforward – because of the motion of points, in the regular triangulation some non-redundant points can become redundant and vice versa – therefore we make an exception and describe the original algorithm designed for the Delaunay triangulation. A principle of the algorithm is as follows. Given a set of points, each point moves continuously along its own *polynomial* curve in time. At the beginning, the Delaunay triangulation of the given points in their "starting positions" is computed. Now if the points start to move, the topology of the triangulation remains unchanged until a point moves to a circumscribed sphere of a tetrahedron of $DT(S)$, in other words, until five points become cospherical. In this moment, which is called *a topological event*, the topology of $DT(S)$ has to be modified (with one exception which will be described later). So to maintain a kinetic $DT(S)$, all topological events have to be detected and handled in the order of their occurrence in time.

5.1.1 Topological Event

Assume that $p(t)$ denotes a position of a point p at the time t. A topological event arises if five points become cospherical – it means that the following in_sphere test of the five points a, b, c, d, e is equal to zero (in the following equation, $a_q(t) = a_x^2(t) + a_y^2(t) + a_z^2(t)$).

$$in\_sphere(a(t), b(t), c(t), d(t), e(t)) = \det \begin{pmatrix} a_x(t) & a_y(t) & a_z(t) & a_q(t) & 1 \\ b_x(t) & b_y(t) & b_z(t) & b_q(t) & 1 \\ c_x(t) & c_y(t) & c_z(t) & c_q(t) & 1 \\ d_x(t) & d_y(t) & d_z(t) & d_q(t) & 1 \\ e_x(t) & e_y(t) & e_z(t) & e_q(t) & 1 \end{pmatrix} \quad (5.1)$$

As the trajectories are polynomial curves, a determinant of the matrix in Eq. 5.1 is a polynomial of the variable t and we look for zero values of the polynomial, i.e. we solve a polynomial equation of high degree (the degree depends on the degrees of the trajectories). This cannot be done analytically, an iterative numerical approach must be used. The equation can have more roots, i.e. movements of the five points can cause more topological events, but we are interested only in the nearest "future" root, i.e. in the nearest future topological event. This root will be called the time of execution of the topological event.

Of course, there is no need to solve the in_sphere test for every 5-tuple of points of S. A moving point p has always first to penetrate into a circumscribed sphere of some tetrahedron, which is a neighbor of some tetrahedron of $star(p)$ but is not incident to p, before p can move into a circumscribed sphere of some "farther" tetrahedron (see Fig. 5.1). Therefore, we can restrict our search for topological events to the pairs of neighboring tetrahedra. In consequence, each topological event is represented by a triplet consisting of a pair of indices (or pointers) of the two associated tetrahedra and the scheduled time of execution.

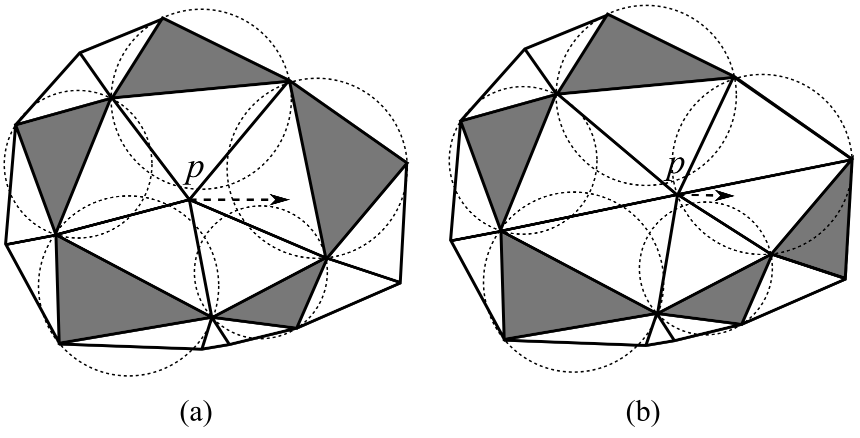


Figure 5.1: Topological event in 2D. The tetrahedra neighboring with $star(p)$ are shaded. (a) Point p moves into a circumscribed circle of a tetrahedron neighboring with $star(p)$. (b) The updated triangulation.

5.1.2 Handling of Topological Event

Assume that the movement of vertices of two neighboring tetrahedra $abcd$, $bcde \in DT(S)$ causes a topological event, i.e. the points a, b, c, d, e, which were in a general position before a moment, become cospherical. Now two cases have to be distinguished:

1. In the next moment, the point a will be outside the circumscribed sphere of $bcde$, i.e. the trajectory of a is tangent to this sphere, see Fig. 5.2a.
2. In the next moment, the point a will be inside the circumscribed sphere of $bcde$, i.e. the trajectory of a intersects this sphere, see Fig. 5.2b.

In the first case, the topology of $DT(S)$ remains unchanged. Therefore such topological events are ignored. In the second case, the topology of $DT(S)$ has to be updated – the shared face bcd has to be flipped by the $flip$ 23 or $flip$ 32. According to [1], one of these flip is possible.

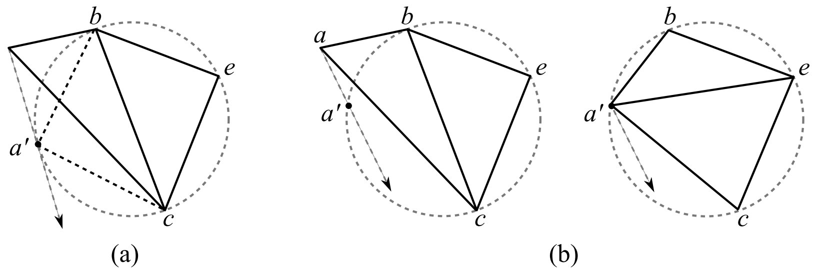


Figure 5.2: Topological event in 2D. (a) Point a moves along a trajectory tangent to the circumscribed circle, the topology remains unchanged. (b) Point a moves into the circumscribed circle, the topology changes when a becomes cocircular with b, c, d (this position is marked as a').

5.1.3 Algorithm

The algorithm consists of an initialization step and of a lifecycle. In the initialization step, a triangulation $DT(S)$ of the points in their starting position is constructed. Then for each pair of neigbouring tetrahedra in $DT(S)$ its topological event is computed. The found topological events are scheduled (not each pair of tetrahedra causes an event), i.e. the topological events are stored in a priority queue according to their time of execution, the nearest topological event is at the top of the queue. In the lifecycle the algorithm repeats the following. First, the nearest topological event is popped from the queue and the topology of $DT(S)$ is modified – a face shared by the two tetrahedra associated with the popped event is flipped. In consequence of this flip, all events which are associated with at least one of the tetrahedra destroyed by the flip become invalid and have to be removed from the queue. Also new future topological events can arise – a topological event is computed for each pair of neighboring tetrahedra containing at least one new tetrahedron and if it exists, it is inserted into the queue according to its priority (time of execution).

In the described algorithm, the nearest topological event is found in $O(1)$, the topology of $DT(S)$ is also updated in $O(1)$ and the update of the priority queue takes $O(log\ n)$, where n is a number of points in S.

5.2 Dynamic Triangulation

A life of a point (a vertex) in the dynamic triangulation is discrete – in certain given moments the point appears, disappears or changes its position. Obviously, an appearance and a disappearance of a point can be handled by the algorithms for insertion and deletion, which are discussed in Chapter 3 and 4 in detail. A change of a position of a point can be also processed as a combination of a deletion and an insertion of the point. But these operations are slow. Guibas et al. shown in [26] that if a position of a point changes only slightly, it is faster to move the point from its old position to its new position "in the kinetic way", i.e. the point is moving continuously and each time it penetrates into a circumscribed sphere of a tetrahedron, the triangulation is repaired by a flip.

Schaller et al. offered in [39] a slightly different approach – a point p is moved to its new position in several steps. A size of each step is chosen in such a way that the tetrahedra incident to p do not intersect their neighboring tetrahedra (but p can be moved into a circumscribed sphere of some tetrahedra). After each step, the triangulation is repaired by flips, see Fig. 5.3. Here is a weak point of this algorithm – Schaller does not prove that the necessary flips are always possible and we have doubt about it.

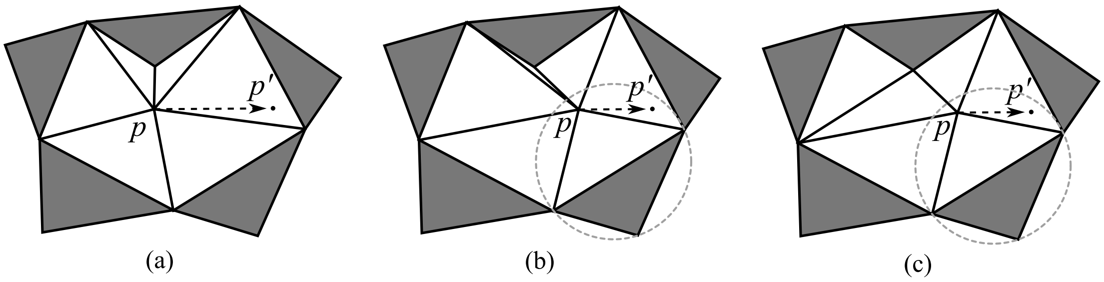


Figure 5.3: Schaller's algorithm in 2D – a movement of the point p. (a) The starting configuration. (b) p is moved as far as possible without a rise of intersecting tetrahedra. Note that p can be inside a circumscribed circle, but we do not care. (c) The triangulation is repaired by flips.

Chapter 6

Applications of Regular Triangulations

In this Chapter, we describe the applications of regular triangulations. We focus on the topic on which we directly participate – the analysis of channels in protein molecules. The other interesting applications of RT are mentioned just briefly.

6.1 Applications

Ferrez [23] uses regular triangulations to simulate behavior of granular materials and to detect collisions between grains. Maur et al. [34] describe how to use regular triangulations to construct constrained Delaunay triangulations in 2D. Cheng et at. [11] use careful assignment of weights of points to eliminate slivers (nearly planar tetrahedra) in triangulations. The algorithm Power crust [2] uses the dual structure of RT (power diagrams) to reconstruct a surface from a set of sample points. Brož [9] employs regular triangulation to find a shortest path in a dynamic environment with obstacles. Each obstacle is represented as a weighted point, the weight is given by a size or by a dangerousness of the obstacle. Beside this, RT can naturally replace DT in all its applications.

6.2 Channels in Proteins

Biochemists study the protein properties, modify them and try to create "improved proteins". These improved proteins are then used in designs of new drugs and agrochemicals or for detection and neutralization of dangerous chemicals. This research involves a huge number of time consuming tests, therefore, biochemists widely use a visualization and a computer simulation of the proteins and of their behavior to make their job easier. An example of an interest of biochemists is a molecular surface. A molecular surface can be computed via a spherical probe rolling around the molecule. In [5], a dynamic maintenance of a molecular surface is described. A power diagram (the dual structure of RT) is used for a representation of a molecule and a molecular surface is represented as the NURBS patches.

A similar problem is the construction of "a pocket". A pocket is a depressed region on the surface of a protein and is often used for an interaction between a protein and some small molecule. In [19], the pockets are constructed via a Delaunay triangulation and an alpha shape of a protein. In [28], a Euclidean Voronoi diagram (also called a Voronoi diagram of spheres) and a convex hull of a protein are used for the pocket computation and recognition.

Not all chemical reactions between a protein and another molecule happen on a protein surface. Many of them proceed deep inside proteins. Here a channel analysis is useful. A channel is a cavity, leading from a biochemically significant place (so called *active site*) inside a protein to its surface. Medek et al. (see [36]) proposed an algorithm which finds a channel in a protein as a sequence of tetrahedra in the 3D Delaunay triangulation of the given protein. We have modified this algorithm for the computation of channels in regular triangulations and describe it in detail in Section 6.2.3, because in Section 7.1 we propose an algorithm which speeds up the channel computation in *dynamic* proteins.

6.2.1 Geometric Model of Proteins and Channels

We omit the chemical aspects of channels and consider only the protein geometry. A protein is simplified to a set of spheres – each atom is approximated by exactly one sphere, where a radius of each sphere is equal to the van der Waals radius[1] of the corresponding atom (the van der Waals radius of an atom is the radius of an imaginary hard sphere which is used to model the atom in various applications). Also we restrict ourselves to channels with circular perpendicular cross-section (this greatly simplifies the algorithm). This is a strong simplification, but the resulting channels still provide relevant information and are used by the biochemists. This model of a protein implies the following formal definition of the channel.

Channel A channel $\mathcal{R}$ in a protein leading from a point p to a point q is defined by its centerline and its volume. The centerline is a continuous curve starting in p and ending in q. The volume of the channel is formed by the union of spheres (so-called channel spheres) whose centers lie on the centerline and which are tangent to at least one atom but do not intersect any atom. The sphere with the minimal radius is called the bottleneck sphere of $\mathcal{R}$, the center of this sphere is called the bottleneck point of $\mathcal{R}$ and the radius of this sphere is called the bottleneck radius of $\mathcal{R}$ (see Fig. 6.1). We call the channel $\mathcal{R}$ *ideal*, if no other channel leading from p to q with a bigger bottleneck radius exists.

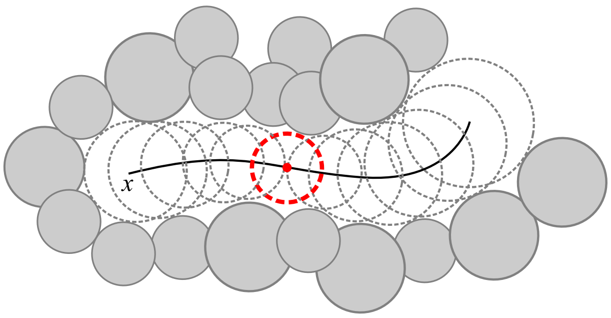


Figure 6.1: The 2D channel, “x” marks the active site, the bottleneck sphere and the bottleneck point are shown.

Channels could be optimized according to various criteria. For example, our interest can be to minimize a length of a channel, to minimize the curvature of a channel centerline, or to

[1] http://en.wikipedia.org/wiki/Van_der_Waals_radius

maximize a channel volume. But our todays interest is to maximize the bottleneck radius of a channel.

In the following text we commit a certain inaccuracy. Often we treat a weighted point as a sphere. In such a case, a weighted point p with a weight w_p is interpreted as a sphere with the radius $r_p = \sqrt{w_p}$ centered in the point p. If p is a (weighted) point and s is a sphere, the term $|ps|$ always denotes the Euclidean distance between p and the center of the sphere s. Further let us define the distance of an unweighted point from a set of spheres.

Distance from spheres Let S be a set of spheres and p be a point. The distance of p from S will be denoted as $dist(p,S)$ and is defined as follows:

$$dist(p,S) = min\{|ps| - r_s \mid s \in S\}, \tag{6.1}$$

i.e. $dist(p,S)$ is the distance between p and the nearest spherical surface.

6.2.2 Channels in Euclidean Voronoi Diagram

A channel could be hardly constructed directly from its definition. But it can be computed effectively in a Euclidean Voronoi diagram via the following Lemma 6 (see [36]).

Lemma 6 *Let S be a set of spheres in 3D. Consider an ideal channel $\mathcal{R}$ with a centerline leading from a point p to a point q. The bottleneck point of $\mathcal{R}$ is situated on some Voronoi edge of Euclidean Voronoi diagram of S or in the point p or q.*

By this lemma, we can restrict a search for a bottleneck of a channel to edges of a Euclidean Voronoi diagram and hence we can construct a channel centerline as a subset of its edges. Now the channel computation is easy. First, we assign to each edge h of $EVD(S)$ a real weight – the radius of the biggest sphere, which can move along h without an intersection with any sphere of S (we describe this in detail below). $EVD(S)$ together with these weights of edges can be interpreted as a weighted graph G – the vertices of $EVD(S)$ are the nodes of G, the edges of $EVD(S)$ are the edges of G. An ideal channel leading to a given active site is computed in G via a simple greedy algorithm. The greedy algorithm maintains a set D of "known" edges. At the beginning, only the edge closest to the active site is inserted into D. Then in each step, the algorithm adds to D an edge h, $h \notin D$, which is connected with some edge $h' \in D$ (i.e. h and h' share a Voronoi vertex in $EVD(S)$) and has the maximal weight. This process continues until an edge ending outside the protein is found. Then the channel centerline is reconstructed using a backward traversal from these ending edge to the active site as a sequence of edges. An edge of this sequence which has the minimal weight determines the bottleneck of the channel, the bottleneck radius is the minimal weight. When the channel centerline is known, the channel volume is obtained as the union of spheres whose centers lie on the centerline and which are tangent to the nearest atoms. Because the centerline consists of edges of $EVD(S)$, a computation of such a sphere in $EVD(S)$ takes $O(1)$ time. The weight of an edge h is the radius of a bottleneck sphere of the edge h which is defined as follows:

Bottleneck sphere of an edge The bottleneck sphere of an edge h is the sphere s satisfying:

- the center of s lies on h,
- s is tangent to at least one sphere of S,

- s does not intersect or contain any sphere of S,
- there is not any smaller sphere s' satisfying the three previous conditions.

In other words, a bottleneck sphere of an edge h is the biggest sphere which can move along h and does not intersect any sphere of S. The computation of a bottleneck sphere of h is simple. The bottleneck point of h has to be located in the point of intersection of h and the plane ϱ passing through the centers of the three spheres determining the edge h, see Fig. 6.2a. Or, if this intersection does not exist, the bottleneck point lies in one of the endpoints of h, see Fig. 6.2b. The radius of the bottleneck sphere is the distance between the bottleneck point and the nearest sphere of S (the nearest atom).

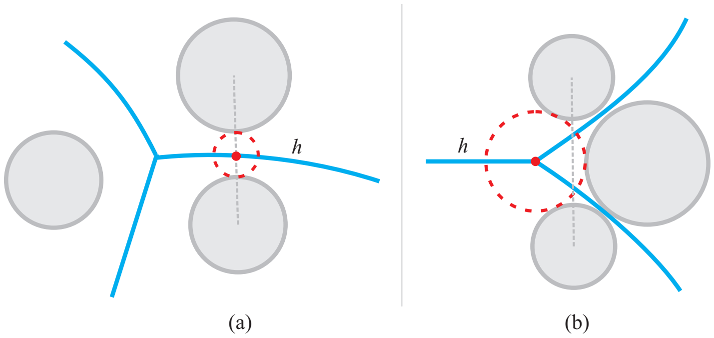


Figure 6.2: Bottleneck sphere of the edge h in EVD in 2D. (a) The bottleneck point of h is located in the point of intersection of h and the line segment joining the centers of the two circles determining the edge h. (b) The bottleneck point lies in one of the endpoints of h.

6.2.3 Channels in Regular Triangulation

Euclidean Voronoi diagrams are very intricate and algorithms of their construction are time consuming and hard to implement. For example, the time complexity of the algorithm described in [29] is $O(n^3)$, where n is the number of spheres. Also the differences between atom radii in proteins are small (a ratio of the van der Waals radii of the biggest and the smallest atom in proteins is 1,5), hence some approximation of $EVD(S)$ should be sufficient. Medek et al. use Voronoi diagrams of points. This type of diagram ignores the atom radii completely. We employ power diagrams – they reflect weights of points and can be constructed effectively. Not surprisingly, we use the dual of power diagrams – regular triangulations. An edge of a power diagram is dual to a face of a regular triangulation, a vertex of $PD(S)$ is dual to a tetrahedron in $RT(S)$. Hence to traverse an edge of $PD(S)$ from one its endpoint to the other is equivalent to a traverse from one tetrahedron in $RT(S)$ to its neighbor through their common face. So in $RT(S)$, channels are computed via a traversing of tetrahedra, where a traversing order is given by weights of faces. A weight of a face f in $RT(S)$ is the radius of a bottleneck sphere of f which is defined as follows:

Bottleneck sphere of a face Let $abcd$ and $bcde$ be two neighboring tetrahedra in $RT(S)$ with a common face bcd. The bottleneck sphere of the face bcd is the sphere computed as follows (in the following text s_1, s_2, s_3 denote three spheres with the radii r_{s1}, r_{s2}, r_{s3}, see Fig. 6.3):

1. Place a center of a sphere s_1 into the orthogonal center of the tetrahedra *abcd* and set the radius r_{s1} to $dist(s_1,\{a,b,c,d\})$.

2. Analogously compute the sphere s_2 in the tetrahedron *bcde*. Note that the line joining the centers of the spheres s_1 and s_2 is the edge of $PD(S)$ dual to the face *bcd*.

3. If the centers s_1 and s_2 lie on one side of the plane supporting the face *bcd*, then the bottleneck sphere of the face *bcd* is the smaller sphere of s_1 and s_2.

4. Otherwise, place a center of a sphere s_3 in the intersection of the plane supporting the face *bcd* and the line joining centers of s_1 and s_2, and set the radius r_{s3} to $dist(s_1,\{a,b,c,d,e\})$.

5. The bottleneck sphere of the face *bcd* is the smallest sphere of s_1, s_2 and s_3.

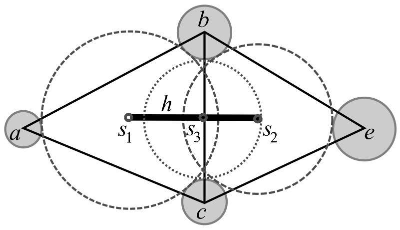


Figure 6.3: Bottleneck sphere of a face in 2D.

A channel in $RT(S)$ is computed similarly as in $EVD(S)$. $RT(S)$ is interpreted as a weighted graph G. Each node in G is equivalent to exactly one tetrahedron in $RT(S)$, two nodes in G are connected with an edge h if their two corresponding tetrahedra in $RT(S)$ share a face, and the weight of h is equal to the weight of this shared face. The graph G is traversed via a greedy algorithm (very similar to the algorithm described above in 6.2.2). The greedy algorithm maintains a set D of "known" edges. At the beginning, a node of G corresponding to the tetrahedron containing a given active site is found and the edges ending in this node are inserted into D. Then in each step, the algorithm adds to D an edge h, $h \notin D$, which has a common node with some edge $h' \in D$ and has the maximal weight. This process continues until an edge ending in a node with the degree less than 4 is added into D (such a node corresponds to a tetrahedron lying on the triangulation border). Then the channel is reconstructed using a backward traversal from the ending node to the starting node as a path P in G (a sequence of nodes and edges of G). The minimal weight of an edge in the path P determines the bottleneck radius of the channel. The channel centerline is computed as a polyline whose vertices are the orthogonal centers of the tetrahedra corresponding to the nodes of P (see Fig. 6.4). Two ways of a computation of a channel volume are described later in Sections 6.2.4 and 6.2.5.

This channel computation in $RT(S)$ or $PD(S)$ has a drawback. The points of the edges of $PD(S)$ maximalize their power distance from the weighted points of S, but not their Euclidean distance. Assume that h is a line joining the orthogonal centers of two neighboring

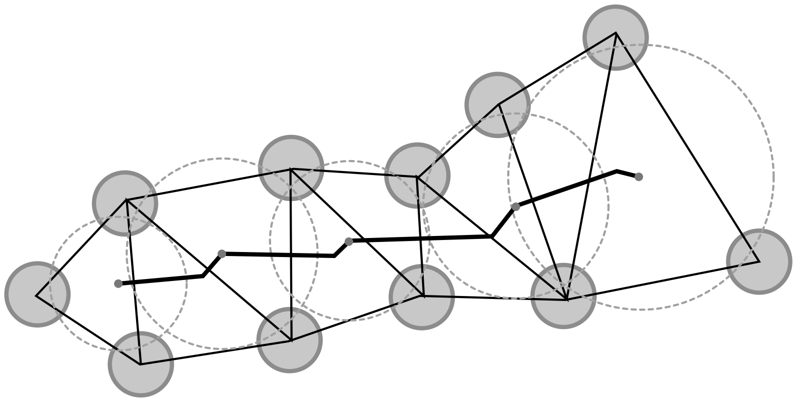


Figure 6.4: Centerline of a channel in the 2D regular triangulation. Only odd orthogonal circles are shown for a better clarity.

tetrahedra *abcd* and *bcde* of $RT(S)$, s is a sphere centered on h with the radius r_s equal to $dist(s,\{a,b,c,d,e\})$. The sphere s does not intersect the spheres a, b, c, d, e, but it still can intersect some other spheres of S. Therefore the described computation of a bottleneck sphere of a face in $RT(S)$ is not accurate and resulting bottleneck spheres can intersect some spheres of S (atoms). Similar problem appears in the computation of a channel volume. Therefore, we distinguish two types of channels:

- a pessimistic channel – it must not contain any intersection with any atom.
- an optimistic channel – it can contain small intersections with some atoms, but its bottleneck radius is greater than or equal to the radius of a corresponding pessimistic channel.

6.2.4 Optimistic Channels

Assume a channel centerline is computed by the description given in 6.2.3. The channel volume is formed by the union of spheres centered on the channel centerline. Let h be a segment of the channel centerline joining the orthogonal centers of tetrahedra *abcd* and *bcde*. The radius of each channel sphere s centered on h is set to $dist(s,\{a,b,c,d,e\})$. Clearly, the resulting channel can intersect some spheres of S (atoms). A size of this intersection is formalized in the following definition.

Channel error If a sphere s of a channel in $RT(S)$ intersects a sphere $a \in S$, the so-called *channel error* is given as

$$channel\ error = r_s + r_a - |as|, \tag{6.2}$$

where r_s is the radius of the sphere s, r_a is the radius of the sphere a and $|as|$ denotes the Euclidean distance between the centers of a and s.

In optimistic channels, we ignore the channel errors and hope that they are insignificant. An upper bound of a channel error is shown in the following section.

6.2.5 Pessimistic Channels

Pessimistic channels must not intersect any atom. Their construction is based on the following lemma, which gives an upper estimate of a channel error. The proof of this lemma can be found in author's master thesis [49].

Lemma 7 *Given a regular triangulation RT(S) and two tetrahedra abcd, bcde $\in$RT(S) with the common face bcd. Let s be a sphere centered on the line joining the orthogonal centers of the tetrahedra abcd and bcde with the radius r_s =dist$(s, \{a, b, c, d, e\})$. Further assume that the radius of the sphere a is greater then or equal to the radii of the spheres b, c, d, e and that r_{max} is the radius of the biggest sphere in S. Then the maximal possible channel error of the sphere s fulfills the following inequality:*

$$channel\ error \leq r_{max} + r_s - \sqrt{(r_s^2 + 2r_a r_s + r_{max}^2)} = f(r_s, r_a, r_{max}) \tag{6.3}$$

So if the radius r_s of the sphere s from Lemma 7 is reduced by the error estimate $f(r_s, r_a, r_{max})$, then the resulting sphere cannot intersect any sphere of S (atom). Let us note that the channel error estimate is dependent only on the radii r_s, r_a, r_{max}. The time complexity of one error estimate calculation is clearly $O(1)$.

To construct a pessimistic channel, the computation of a bottleneck sphere of a face f is modified according to Lemma 7. First, the spheres s_1, s_2 and s_3 are computed according to the definition of the bottleneck of a face. For each of these spheres, the radius is reduced by the upper estimate of the channel error calculated according to the formula (6.3). The bottleneck sphere of the face f is the smallest sphere of these "reduced" spheres.

The channel volume is computed in the same way as a volume of an optimistic channel, but the radius of each channel sphere is reduced by its channel error estimate.

We have made extensive tests of the channels computation – we have compared optimistic and pessimistic channels and channels computed in regular and Delaunay triangulations. The conclusion is that neither a type of channel nor a type of the used triangulation significantly affects the resulting channels. As a matter of interest, we show the comparison of the average bottleneck radii of optimistic and pessimistic channels in four proteins. In each protein, optimistic and pessimistic channels leading to one thousand active sites were constructed. Tab. 6.1 shows that the optimistic channels are slightly wider than the pessimistic channels. The values of radii are given in angstroms, 1 angstrom = $1A = 10^{-10}$metre.

Average bottleneck radius [A]

Protein	Optimistic channels	Pessimistic channels
1awj	1,398	1,270
1dbja	1,536	1,468
dhaa	1,100	0,986
2f61	1,829	1,761

Table 6.1: The average bottleneck radii of optimistic and pessimistic channels in four proteins.

Chapter 7

Our Contribution

This chapter describes the main results of the author's research. The research has been mostly focused on regular triangulations within the context of the dynamic variable data. So the first topic deals with the computation of channels in dynamic proteins, the second topic addresses the more general problem – the deletion of a point in a regular triangulation.

7.1 Channels in Dynamic Proteins

In Section 6.2, we have described the computation of channels in *static* proteins. But the geometry of a protein is not static – the atoms move in time and these movements can influence the shape, the centerline or even an existence of the channel leading to a specific active site. The typical goal is to find a wide and time-stable channel. Therefore a long sequence of "molecule snapshots" has to be explored to see the changes in the channel properties.

A recent method of the channel computation [36] creates a triangulation of the whole protein for each snapshot. This process can be very time-consuming, because the sequences contain hundreds or thousands of snapshots and the number of atoms in a protein varies from thousands to hundreds of thousands.

Our proposed method (published in [51]) speeds up the channel computation in a sequence of snapshots significantly. The acceleration is based on two observations. First, a channel goes usually only through a small part of a protein. Second, those channels, which are wide enough to be usable, do not change their centerline between two following snapshots dramatically. Because of the first observation, we involve a clustering of atoms to our method. With regard to the second observation, we exploit the topology information about the channel in a snapshot for the computation of the channel in the following snapshot. Our resulting channels are almost identical with the channels computed in the triangulation of the whole protein and the total computing time falls to thirty percent and less.

7.1.1 Survey of the Proposed Solution

Here we briefly introduce our proposed method, its principle is explained in detail in Section 7.1.2. To compute a channel in a single protein snapshot, we use the algorithm of the channel computation described in 6.2. So, a channel is found as a sequence of tetrahedra in a regular triangulation and is given by its centerline and volume. Our interest is to maximize the bottleneck radius of the channel.

Let us remind that the goal of our method is to accelerate the channel computation in a sequence of molecule snapshots. The recently used approach creates a triangulation of all points (atoms positions) for each snapshot. Our proposed method speeds up the sequence processing so that it tries to create the triangulation only of those points which affect the channel. It profits from the fact that the positions of atoms as well as the channel centerline do not change very much between successive snapshots. But the proposed method is still able to compute the optimal channel in each snapshot even if the channel completely changes between two successive snapshots. The method works as follows.

As the first step, the points are clustered, each cluster can be represented with one of its points – a cluster center. In the first snapshot processing, all points are triangulated and a channel is found. In each successive snapshot, we create a triangulation only of the points which were near the channel in the previous snapshot and of the cluster centers (the cluster centers substitute the remaining, non-triangulated points). A channel is computed in this "subtriangulation" and is checked. If the channel leads through tetrahedra with cluster centers (it means that the channel has changed because of shifts of atoms), the clusters corresponding to these cluster centers are expanded, i.e. all the points in these clusters are added into the triangulation. Then the channel is computed again. This process is repeated, until the channel is correct.

Before we continue with the detailed description of the proposed algorithm, let us describe the clustering algorithm, which is used inside our proposed method[1].

Clustering

Clustering in general is a process of grouping elements with similar properties. In this thesis, clustering is used to identify groups of 3D points lying geometrically close together.

Perhaps the best known approach to clustering is the k-means clustering. It partitions the points into exactly k clusters, so that the sum of distances from each point to its cluster center is minimized. This technique has a drawback that we need to know the number of clusters in advance. Therefore, we solve clustering as a facility location problem. Every point becomes a potential cluster center (so called facility). For opening a facility, the facility cost c must be paid. All other points are connected to the closest open facility. The problem is to determine at which points a facility should be opened to minimize the overall clustering cost

$$C = k \cdot c + \sum_{i=0}^{n-1} d_i$$

where k is the number of open facilities, n is the number of all points, and d_i is the distance of point i to its facility.

The local search technique [10] can be used to solve the facility location. The method first generates a coarse initial solution which is then iteratively refined by local improvements. Suppose we have such an initial solution. One local improvement proceeds as follows. A random point p is selected and it is determined whether opening a facility at p can improve the solution. If there is not a facility already open at p, the facility cost c would have to be paid for opening it. Then some points may be closer to p than to their current facility. Such points can be reassigned to p, thereby decreasing their distance to facility. After that some facilities may have just a few points remaining connected. If such facilities were closed, their

[1] I would like to thank to Jiří Skála for providing his dll library, which implements the clustering.

facility cost would be spared. The remaining points would have to be reassigned to some more distant facility, but the spared facility costs could overweight this.

If the above mentioned reassignments and perhaps closures are found to improve the solution (i.e., to decrease the clustering cost C), they are actually performed. Local improvements should be repeated $n \log n$ times [10] to get a constant-factor approximation. In [44] it can be seen that the number of iterations could be reduced to about $0{,}01n$ without any major impact on the clustering quality.

A method proposed in [37] is used to generate an initial solution. Points are taken in random order. At the first one, a facility is always opened. For each following point, the distance d to the closest open facility is measured. A new facility is opened at the point with probability d/c (or 1 if $d > c$). Otherwise the point is connected to the closest open facility.

The initial solution can be generated in $O(n^2)$ time. A single local search step needs to inspect all n points and should be repeated $n \log n$ times. The time complexity of the clustering is thus $O(n^2 \log n)$, but it could be reduced to $O(n^2)$ by decreasing the number of local search iterations. The iterative nature of the algorithm allows updating the clustering by simply performing additional iterations. This is useful for quickly updating the clustering after points have been shifted slightly.

The facility cost parameter specifies how expensive is to create a cluster. It can be used to tune the clustering algorithm. A high facility cost means expensive clusters, so the algorithm will create just a few large clusters. A low facility cost will result in many small clusters.

7.1.2 Proposed Solution in Detail

First, let us remind that a channel in $RT(S)$ is found via the traversing of the triangulation by a greedy algorithm (see Section 6.2.3). Assume that the channel $\mathcal{R}$ has been computed in $RT(S)$. All the tetrahedra which have been explored by the greedy algorithm together with all their neighbor tetrahedra will be referred as the set $T_{\mathcal{R}}$. The set of the vertices of the tetrahedra from $T_{\mathcal{R}}$ will be referred to as $S_{\mathcal{R}}$ and also as *the touched vertices*, see Fig. 7.1.

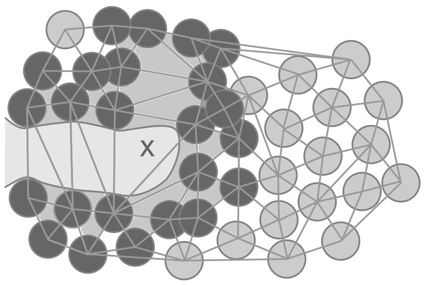


Figure 7.1: The channel in the triangulation (in 2D). The points of $S_{\mathcal{R}}$ are dark gray, the tetrahedra of $T_{\mathcal{R}}$ are shaded. Note that $T_{\mathcal{R}}$ contains *all* tetrahedra which have been traversed during the channel computation and their neighbors, not only the tetrahedra of the resulting channel.

As mentioned above, a long sequence of molecule snapshots has to be processed to see the channel changes. But maybe there is no need to compute for each snapshot the triangulation of the whole protein. Perhaps some points (atoms) do not have any effect on a channel and could be omitted. In this Section we follow this idea and we design a method which speeds up the channel computation.

Assume that the triangulation of a protein has been created and the channel $\mathcal{R}$ has been computed. Typically, the tetrahedra from $T_{\mathcal{R}}$ are only a small part of the triangulation. The rest of the triangulation does not affect the channel computation. We formalize this idea in the two following lemmas.

Lemma 8 *Let S be a set of weighted points in E^3, RT(S) is the regular triangulation of S, H is a subset of the tetrahedra in RT(S) and S^* is a set of the points associated with the vertices of all tetrahedra in H. Then each tetrahedron from H appears in RT(S^*).*

Proof It can be proved that the regular triangulation $RT(S)$ contains each tetrahedron which is globally regular. Let us suppose that there is a tetrahedron $\mathcal{T}$ in H which does not belong to $RT(S^*)$. The tetrahedron $\mathcal{T}$ is an element of H, therefore, it belongs to $RT(S)$, therefore, $\mathcal{T}$ is globally regular with respect to the set S. S^* is a subset of S, therefore, $\mathcal{T}$ is also globally regular with respect to S^*, therefore, $\mathcal{T}$ belongs to $RT(S^*)$. This is a contradiction of the premise that $\mathcal{T}$ does not belong to $RT(S^*)$.□

Lemma 9 *Let $\mathcal{R}$ be a channel found in RT(S), leading to an active site q. Let RT($S_{\mathcal{R}}$) be a regular triangulation of $S_{\mathcal{R}}$. Consider a channel $\mathcal{R}^*$ found in RT($S_{\mathcal{R}}$), leading to the same active site q as the channel $\mathcal{R}$. Then the channels $\mathcal{R}$ and $\mathcal{R}^*$ are equal (they are given by the same tetrahedra and have the same centerline and volume).*

Proof According to Lemma 8, $RT(S_{\mathcal{R}})$ contains all the tetrahedra from $T_{\mathcal{R}}$. $RT(S_{\mathcal{R}})$ can contain some other tetrahedra, which do not appear in the $RT(S)$, but these tetrahedra are not used during the channel computation (only tetrahedra from $T_{\mathcal{R}}$ are) and therefore they cannot affect the resulting channel.□

According to Lemma 9, it is possible not to insert certain points into the triangulation without affecting the channel computed in this triangulation. There is an obvious problem – these points are not known until a channel is constructed. It is true, if the channel is computed in a single molecule snapshot. But in the case of a sequence of snapshots, we can profit from the fact that the channel does not change very much in two successive snapshots.

With regard to this fact, we can exploit the information about the set $S_{\mathcal{R}}$ in a snapshot to speed up the channel computation in the following snapshot. The indices of the points from $S_{\mathcal{R}}$ in a snapshot are saved. In the following snapshot, only the points with these indices (but with new coordinates) are triangulated. A channel on this "subtriangulation" may be incorrect as it does not consider a possibility of some bigger protein change between two snapshots. We can distinguish two cases:

1. The channel does not lead to the protein surface (it runs out from "the area of points $S_{\mathcal{R}}$", see Fig. 7.2 a, b).
2. The channel intersects some points (atoms), which have not been triangulated (see Fig. 7.2 a, c).

To solve the first case, we use the point clustering. Before a channel is computed, all the points in the snapshot are clustered and a center of each cluster (one point from the cluster

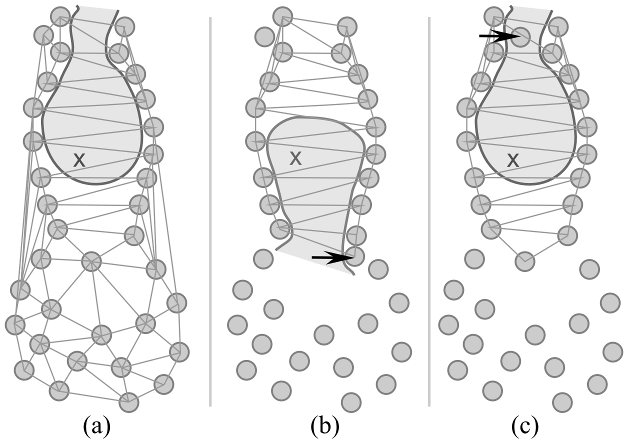


Figure 7.2: (a) The correct channel in the triangulation of the whole snapshot, (b, c) the incorrect channels in the subtriangulation of the successive snapshot. (b) One triangulated atom (marked with an arrow) has moved, the computed channel does not lead to the real protein surface. (c) One non-triangulated atom (marked with an arrow) has moved and lies inside the channel.

representing the whole cluster) is inserted into the triangulation. These centers are marked with the flag *_cluster*. After the channel is computed, all the tetrahedra of the channel are checked whether they contain some cluster centers as vertices (we will call this test the *topology test* of the channel). If so, the channel could be incorrect – it could run out from "the area of the points $S_{\mathcal{R}}$". In such a case the clusters corresponding to these certain cluster centers are expanded, i.e. all the points of these clusters are added into the triangulation (except the points which have been used as the cluster centers – these points have been already inserted), the flags *_cluster* are removed from the expanded cluster centers and the channel is computed again (see Fig. 7.3). This process is repeated until the correct channel is found.

The solution of the second case is simple. All the channel spheres are checked whether they intersect any non-triangulated atoms (we will call this test the *geometry test* of the channel). If so, the points representing the intersected atoms are added into the triangulation and the channel is computed again. This process repeats until the channel does not intersect any atom.

In fact, these two corrective techniques are combined together in a corrective loop. First, the topology test is used. If it fails, some clusters are expanded and the channel is computed again. This repeats until the topology test is satisfied. Then the geometry test is used. If the channel intersects some non-triangulated atoms, these atoms are inserted into the triangulation and the corrective loop continuous with the topology test. This repeats until both test are satisfied. Note that a geometry test is called only once per one corrective loop iteration whereas the topology test can be called repeatedly. This is not necessary, but the topology test takes less time and typically causes bigger changes in the triangulation and in the channel. Thanks to the corrective loop, the proposed method is able to compute the optimal channel in the subtriangulation even if the channel completely changes between two successive snapshots. The whole algorithm is summarized in a pseudocode 7.6 and a visualization of a channel in a subtriangulation is shown in 7.6.

Algorithm 7.1: The computation of channels in a dynamic protein

```
Input: a sequence of protein snapshots and an active site
Output: a sequence of channels
The first snapshot;
  Triangulate the whole first snapshot;
  Compute the clusters;
  Compute the channel;
  I ← the list of the indices of the touched vertices;
for all following snapshots do
    Update clusters;
    Triangulate cluster centers and vertices with indices from I;
    Compute the channel;
    channel_is_correct ← false;
    while not channel_is_correct do
        topology_is_correct ← false;
        while not topology_is_correct do
            Test the channel topology;
            if the channel topology is bad then
                Expand the clusters corresponding to the touched cluster centers;
                Compute the channel;
            else
                topology_is_correct ← true;
            end
        end
        Test the channel geometry;
        if the channel geometry is bad then
            Insert the intersected atoms (vertices) into the triangulation;
            Compute the channel;
        else
            channel_is_correct ← true;
        end
    end
    I ← the list of the indices of the touched vertices;
    Send the channel to output;
end
```

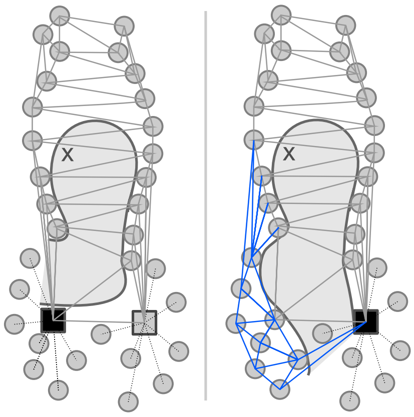


Figure 7.3: The topology test. (a) The channel goes through a tetrahedra whose vertex is a cluster center (marked with the dark square). (b) The cluster corresponding to the cluster center has been expanded and the channel has been recomputed. In this case, the new channel goes around another cluster center.

7.1.3 Details and Issues

The clustering of points is time consuming, but there is no need to compute the clusters for each snapshot again. The used clustering algorithm (see Section 7.1.1) enables to modify the positions of points and then "to update" the clusters. This takes much less computing time.

In the proposed method, some points could be inserted into the triangulation repeatedly (for example, the first time as a point from $S_{\mathcal{R}}$, next time during the cluster expansion). This is not a fundamental problem, because the insertion algorithm recognizes these points and does not insert them into the triangulation again. But first it must locate the point in the triangulation and it consumes some time. To avoid this, each point is marked with the flag *_added* after its insertion into a triangulation and only points without this flag are inserted. This flag can also be used to speed up the geometry test of a channel – only points without the flag *_added* have to be tested (the points with this flag have been inserted into the triangulation and cannot intersect a channel).

Now let us analyze the only reason why the channel computed by the proposed method may not be correct. If the channel changes between two snapshots, the computed channels need not to lead to the real protein surface, it can end inside the protein on the false surface of the subtriangulation (see Fig. 7.4a). In most cases this situation is recognized in the topology test (the tetrahedra near the false surface contain the centers of cluster) or in geometry test (some unused atom intersects the channel). If the problem appeared, it could be reduced by using larger amount of smaller clusters (see Fig. 7.4b). Another solution, safer but slower, is to insert all convex hull points of the snapshot into the triangulation. Nevertheless, we have never noticed such an incorrect channel during our tests.

7.1.4 Time and Space Complexity

In this Section we give the time and space complexity of the proposed method for the channel computation. Let n denote a number of atoms in a snapshot. The time and space complexity

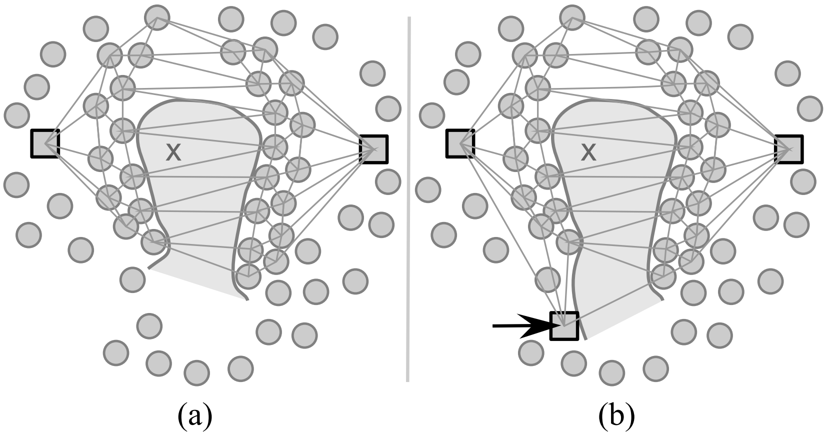


Figure 7.4: The two incorrect channels in the subtriangulations. The cluster centers are marked by the squares. (a) The channel tetrahedra do not contain any cluster center as a vertex and the channel is wrongly considered to be correct. (b) More clusters are used, some channel tetrahedron contains a cluster center (marked with an arrow) and the channel is recognized as incorrect.

of the computation of the regular triangulation in E^3 is $O(n^2)$ in the worst case. However, supposing that atoms in a protein molecule are almost uniformly distributed, we can expect $O(n)$ space and $O(n^{4/3})$ or $O(n \log n)$ time complexity (depending on the used triangulation algorithm). The time complexity of the initial point clustering is $O(n^2 \log n)$, the updating of clusters after the shifts of points runs in $O(n^2)$. Here the O notation hides a constant 0,01, which makes the clusters usable for the protein processing. The time complexity of one channel computation in one snapshot is $O(m \log m)$, where m is the number of the tetrahedra used during the channel computation (typically $m << n$). The topology test of a channel runs in $O(m)$, the geometry test of a channel takes $O(rn)$ time, where r denotes the number of spheres in the channel (typically $r << n$).

The expected time complexity of one channel computation (including the triangulation computation) of the classic method is $O(n^{4/3} + m \log m)$ per snapshot. Our method runs in $O(\bar{n}^{4/3} + n^2 + (k + l)(m \log m) + krn + lm)$, where $\bar{n}$ is the number of triangulated points ($\bar{n} \leq n$) and k, l, respectively, denotes the number of iterations of geometry test, topology test, respectively. According to our test (see Section 7.1.4), the values of k and l are very low in average (less than 6) and the channel computation is very fast; the time complexity roughly is $O(\bar{n}^{4/3} + n^2)$.

7.1.5 Experimental Results

In this Section, the channels computed by our method will be referred as *our channels*, the non-accelerated method will be called the *classic method* and the channel computed by the classic method will be called the *classic channels*.

The algorithm was implemented in C# 2.0, experiments were done on Intel Pentium 4 3.2 GHz with 2 GB RAM, running Windows XP.

Now we are going to present the experiments made with our method of channel computation. We show computing times of our channels and compare it with computing times of the classic channels. For various sequences, we present the behavior of our method – computing time, the number of points inserted into the triangulation, and the number of topology and geometry tests of the channel. Also we show the distribution of computing time among the particular parts of the proposed method.

For all the tests we have used two different "real" sequences of the protein dhaa computed by the molecular dynamics simulation and two generated sequences of protein 1z2y and 1bgl. The sequences of protein 1z2y and 1bgl were generated as follows. To create a new snapshot from the previous one, each atom was shaken – a random number with a uniform distribution from range (-0,9A; 0,9A) was added to each of its coordinates (the differences between the atom positions in the two following snapshots in a real sequence are similar). The characteristics of the used sequences are in Table 7.1.

Protein	# of atoms	Length of sequence
dhaa1	4706	1100
dhaa2	4706	1050
1z2y	13619	50
1bgl	65584	50

Table 7.1: The tested sequences.

The comparison of our method and the classic method is the most important result. We have run both methods for the four sequences. In each sequence, channels to ten different active sites have been computed independently and computing times have been averaged. We have chosen the active sites so that the channels leading to them in the first snapshot are about forty tetrahedra long. According to our experience, this is a reasonable type of a channel (longer channels are typically unstable and therefore useless for the biochemists). Table 7.2 shows that our method computes the channels approximately 4 times faster for the small proteins and 8 times faster for the large protein 1bgl than the classic method. According to Table 7.3, our method uses in average 25% of points from the small proteins and only 9% of points from the large protein.

Running time [s]			
Protein	Proposed method	Classic method	Speed up ratio
dhaa1	0,58	2,3	4,0
dhaa2	0,56	2,3	4,1
1z2y	2,1	8,9	4,2
1bgl	11,2	94	8,4

Table 7.2: Time of channel computation per snapshot.

The time distribution for the four sequences is shown in Figure 7.5. The topology test has a negligible cost. For the small proteins, most of the time is consumed during the triangulation computation. For the large data, the update of clusters and the geometry test become expensive (but the acceleration ratio is still better for the larger data).

The numbers of topology and geometry tests are shown in Table 7.4. The average numbers

Protein	Proposed method ($\bar{n}$)	Classic method (n)	$\bar{n}/n$ [%] ratio
dhaa1	1092	4706	23,2
dhaa2	1097	4706	23,3
1z2y	3470	13619	25,5
1bgl	6065	65584	9,2

Table 7.3: The average number of triangulated points.

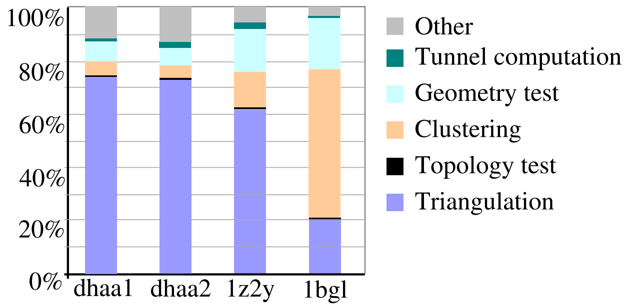


Figure 7.5: The time distribution of the proposed method. The part "Other" involves reading of the input points and the data conversion.

of topology and geometry tests are very low, although in some snapshots the number of topology tests exceeds 15. Note that each topology (geometry) test checks the topology (geometry) of the entire channel (see Section 7.1.4 for the time complexity).

Protein	# of topology tests	# of geometry tests
dhaa1	3,06	1,14
dhaa2	3,22	1,1
1z2y	5,35	1,55
1bgl	3,81	1,71

Table 7.4: The average number of topology and geometry tests per a snapshot.

We have found that our channels are very similar to the classic channels, but not completely the same. We define a similarity of two channels based on the channel topology. For both channels we construct the set of vertex indices of the channel tetrahedra and then we compare these two sets. The similarity of two channels is given as

$$\text{similarity} = \frac{2\,|I_1 \cap I_2|}{|I_1| + |I_2|},$$

where I_1, I_2, respectively, denotes the indices set of our channel, classic channel, respectively. Table 7.5 presents the measured similarity of the channels. With respect to the results, we can say that our proposed method computes the channels nearly identical with the channels computed by the classic method.

Protein	Average similarity [%]
dhaa1	93,7
dhaa2	90
1z2y	95,4

Table 7.5: The similarity of our and classic channels.

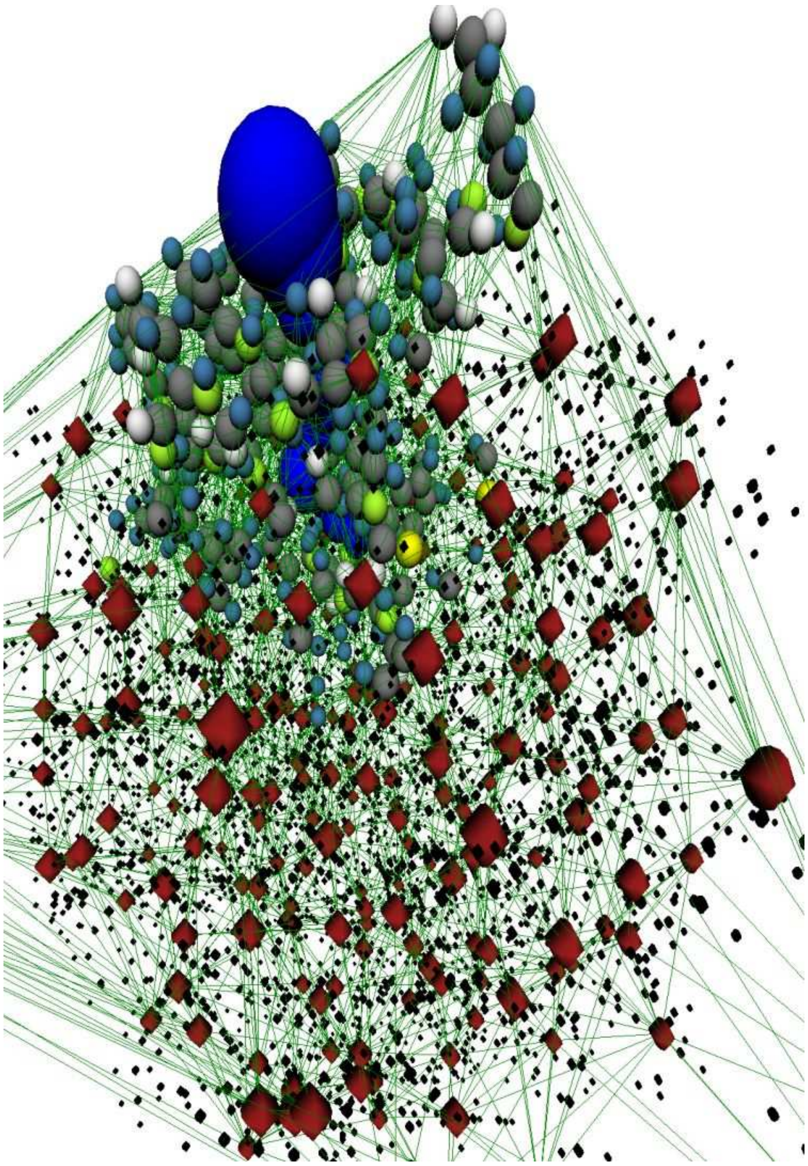


Figure 7.6: A channel in the protein dhaa. The channel is denoted by the dark blue spheres. The other spheres mark the triangulated atoms, the red-brown polyhedra denote the triangulated centers of clusters and the small dark dots are the non-triangulated atoms.

7.2 Hybrid Algorithm for Deletion of a Point

Here we propose a randomized algorithm that allows to delete a point in three–dimensional regular or Delaunay triangulation by a sequence of flips (published in [50]).

Assume that p is the point to be deleted. A test of regularity of a certain tetrahedron with respect to $O(deg(p))$ points is called *the global check*. A test of regularity of a certain tetrahedron with respect to $O(1)$ vertices is called *the local check*.

The previous algorithms (see Chapter 4) check regularity of each new tetrahedron with respect to *all vertices* of tetrahedra incident to the point, which is being deleted (the global check). In contrast, the proposed algorithm uses a combination of local and global checks, therefore, we call it *the hybrid algorithm*. First, the hybrid algorithm tries to delete a point by a randomized sequence of flips of faces satisfying a certain local condition. This simple approach always deletes the point successfully, but theoretically in an unbounded time in the worst case. Therefore we combine it with the global checks – if the point is not deleted after a certain number of flips, the hybrid algorithm replaces the local checks by the global checks. In practice, the local checks are sufficient in most cases and the global checks are used only rarely. In consequence, the hybrid algorithm needs less tests of regularity in average than the previous algorithms. Note that the test of regularity is the most frequent operation in the algorithms for point deletion and a number of these tests determines their asymptotic complexity.

Before we describe the hybrid algorithm, we propose a non-deterministic auxiliary algorithm for point deletion (Section 7.2.1), which uses only the local checks. This algorithm will be called *the basic algorithm*. In Section 7.2.2 we prove that the basic algorithm always deletes p successfully, but in an unbounded time in the worst case. Finally, in Section 7.2.3 we describe the hybrid algorithm – we show how to combine the basic algorithm with the global checks to ensure that the resulting hybrid algorithm always terminates in a finite time. Note that both algorithms can be used for the deletion in Delaunay triangulations. First possibility is to compute $DT(S)$ as $RT(S)$, where all points in S have the same weight. Second possibility is to compute $DT(S)$ directly and to replace regularity tests by the in_sphere tests in the hybrid algorithms. Now let us recall a few definitions from Chapter 4.

Star(p) A set of tetrahedra incident to p in $RT(S)$ is denoted as $star(p)$, $deg(p)$ denotes a number of tetrahedra in $star(p)$.
Star face A face f is a *star face of* p if f is a face of some tetrahedron $\mathcal{T} \in star(p)$, and f contains p.
Link face A face f is a *link face of* p if f is a face of some tetrahedron $\mathcal{T} \in star(p)$, and f does *not* contain p.
Polyhedron $\mathcal{P}(p)$ A star-shaped polyhedron formed by the link faces of p is denoted as $\mathcal{P}(p)$, *the initial* $\mathcal{P}(p)$ denotes $\mathcal{P}(p)$ before the beginning of a deletion of p.
Boundary face Let Y be a set of arbitrary tetrahedra – a face f is a *boundary* face of Y, if f is a face of exactly one tetrahedron $\mathcal{T} \in Y$.

7.2.1 Basic Algorithm

Let $RT(S)$ be a regular triangulation of a set S of weighted points in general position and let $p \in S$ be the point to be deleted. In [21] it is shown that p can always be inserted into $RT(S\text{-}\{p\})$ by a sequence of flips. Therefore p can also be deleted in $RT(S)$ by a sequence of flips. During the insertion of a point, a question whether a face should or should not be flipped can be answered by a simple regularity test of the face. However, a similar local condition which could be used for the point deletion is not known.

The proposed basic algorithm uses only the local checks but at the price of non-determinism – it tries to delete p by a sequence of flips satisfying a certain local condition. But there exist more such sequences, not each of them converges, and it is not known how to find a convergent sequence using only some local conditions. Therefore the basic algorithm performs the flips satisfying the certain local condition in a random order. If the resulting sequence does not delete p, the basic algorithm performs a few "back" flips and then tries other random sequence.

First, we introduce the Necessary Conditions – in the basic algorithm, only faces satisfying these local conditions are flipped.

Necessary Conditions Let f be a star face flippable with a $flip\ mn$, and let Y be a set of n tetrahedra created by the flip. Let $Y' \subset Y$ be a set of tetrahedra which do not contain p. Let Z_p be a set of tetrahedra, $Z_p \cap Y = \emptyset$, which contain p and share some boundary face of Y. Analogously, Let $Z_{!p}$ be a set of tetrahedra, $Z_{!p} \cap Y = \emptyset$, which *do not* contain p and share some boundary of Y. Let Q_p be a set of points, where each $q \in Q_p$ is a vertex of some $\mathcal{T} \in Z_p$ and is not a vertex of any $\mathcal{T} \in Y$. Analogously, let $Q_{!p}$ be a set of points, where each $q \in Q_{!p}$ is a vertex of some $\mathcal{T} \in Z_{!p}$ and is not a vertex of any $\mathcal{T} \in Y$ (see Fig. 7.7). We say that the face f satisfies the Necessary Conditions if it holds:
Condition 1 Each tetrahedron $\mathcal{T} \in Y'$ is regular with respect to the points $Q_{!p}$.
Condition 2 Each tetrahedron $\mathcal{T} \in Y'$ is regular with respect to the points Q_p.

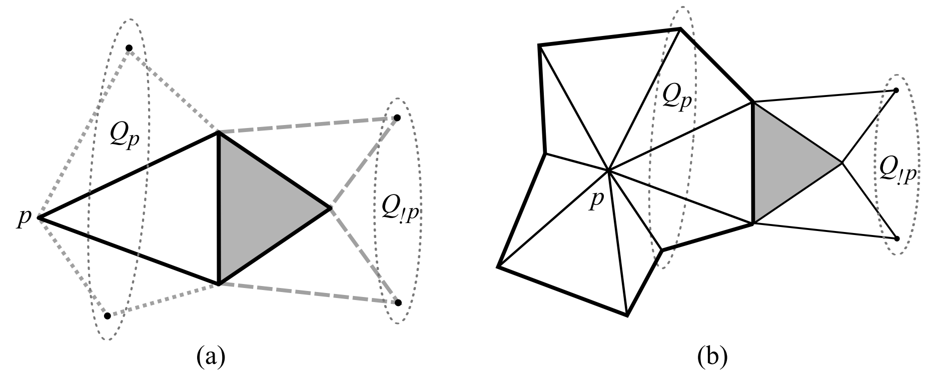


Figure 7.7: (a) The Necessary Conditions (in $2D$) – tetrahedra $\in Y$ are solid (tetrahedra $\in Y'$ are shaded), tetrahedra $\in Z_p$ dotted, tetrahedra $\in Z_{!p}$ dashed. The sets Q_p, $Q_{!p}$ are marked. (b) The Necessary Conditions in the context of $star(p)$.

Remark 1 Let $\mathcal{T}$ be a tetrahedron created by a flip of a star face, $\mathcal{T} \notin star(p)$. The Condition 1 checks, whether the faces of $\mathcal{T}$ which $\mathcal{T}$ does not share with $star(p)$ are regular. Due to this condition, the resulting $RT(S\text{-}\{p\})$ is regular, as shown in Section 7.2.2. The Condition 2 is a heuristic, which significantly accelerates a convergence of the basic algorithm in practice. Note that the verification whether a face f satisfies Necessary Conditions requires $O(1)$ tests of regularity, i.e. it is a local check.

Now we can give the main idea of the algorithm. Let p be again the point to be deleted.

1. A weight w_p of p is set to $-\infty$. In consequence, some star faces of $RT(S)$ become non-regular.
2. *All* flippable non-regular star faces in $RT(S)$ which satisfy both the Necessary Conditions 1 and 2 are flipped in a *random* order. When no such face exists any more, $deg(p)$ is checked. If $deg(p)= 4$, the algorithm continues with the step 4.
3. m randomly chosen link faces are flipped (rules how to choose m will be specified later). In consequence, some flippable non-regular star faces arise in $RT(S)$. After that, the algorithm continues with the step 2.
4. The point p is deleted in $RT(S)$ by a $flip$ 41. Some faces of the tetrahedron created by this flip can be non-regular. These are flipped, as well as the new non-regular faces, which arise due to these flips. Then the algorithm ends.

Before we discuss the algorithm, we need the following Lemma 10.

Lemma 10 *Let $T(S)$ be an arbitrary triangulation of a set of weighted points S and $p \in S$. Assume that the weight w_p is $-\infty$ and $w_q \in \mathbf{R}$ for each $q \in S - \{p\}$. Then each link face is regular. Regularity of a star face abp shared by the two tetrahedra abcp and abdp depends on the angle α between the faces abc and abd. If α is convex, concave, respectively, then abp is non-regular, regular, respectively. If abc and abd are coplanar, then the regularity of the face abp depends both on positions and weights of a, b, c, d, p. A minimal count of non-regular star faces is 6.*

Proof A test of regularity of a face can be done as a determinant computation (see [22]). A (non)regularity of star and link faces follow directly from the expansion of the determinant. The minimal count of non-regular star faces follows from the fact that a minimal count of convex angles between neighboring faces of an enclosed polyhedron is 6 (e.g. a tetrahedron).□

Now let us discuss each step of the algorithm.

Step 1 A change of the weight w_p affects regularity of faces. Clearly, it cannot influence regularity of any face shared by tetrahedra which are not incident with p, in other words, it cannot influence regularity of any face outside $\mathcal{P}(p)$. By Lemma 10, all link faces are regular and only star faces can be non-regular.

Step 2 Also, before any face is flipped, only star faces can be non-regular. Due to the flips of these faces, new non-regular faces can appear. As will be proved in Section 7.2.2, each new non-regular face has to be also a star face. In practice, a list L of potential non-regular star faces is maintained. At the beginning of the step 2, all star faces are added into L. Then, after each flip, the boundary star faces of the created tetrahedra are added into L. The faces are taken out of L in a random order (to prevent an endless loop of the steps 2 and 3). Each chosen face f is tested – if f exists, is flippable, non-regular and fulfills the Necessary Conditions, then f is flipped. Because the algorithm uses only the local checks, in this step a tetrahedron $\mathcal{T}$, $\mathcal{T} \notin star(p)$, can be created, whose all faces are regular but $\mathcal{T}$ is globally non-regular. In such a case p cannot be deleted – after a number of flips the algorithm reaches a state when no flippable star face satisfying the Necessary Conditions exists and $deg(p) > 4$. In this case the algorithm continues with the step 3.

Step 3 In this step m link faces are flipped. A number m is random – we suggest an upper limit of this random value is 1 at the beginning and is increased by 1 each time the step 3 is entered. Link faces to be flipped are chosen randomly, only flips modifying tetrahedra outside *the initial* $\mathcal{P}(p)$ are not allowed.

Step 4 As already mentioned, some faces of the tetrahedron created by the $flip$ 41 can be non-regular and have to be flipped. Some boundary faces of tetrahedra created by these flips can be also non-regular. Therefore in this step, regularity of each boundary face of tetrahedra created by a flip has to be checked and eventually repaired by a flip.

Remark 2 The basic algorithm does not change any tetrahedron outside *the initial* $\mathcal{P}(p)$. But if the points S are not in general position, the deletion of p without a modification of some tetrahedra outside *the initial* $\mathcal{P}(p)$ could be impossible (see [33]). A solution of such a case is discussed in Section 7.2.5.

Remark 3 The algorithm cannot be used if p lies on the boundary of the convex hull of S. This can be easily avoided if four auxiliary points which form a tetrahedron containing all the points of S are added into S before a construction of $RT(S)$.

7.2.2 Correctness of Basic Algorithm

To show the correctness of the basic algorithm, we have to prove two its properties. First, the algorithm always terminates, i.e. the algorithm does not repeat the steps 2 and 3 in an endless loop and always achieves the step 4 (but theoretically in an unbounded time). Second, all faces in $RT(S\text{-}\{p\})$ are regular after the algorithm ends. We begin with the first property.

Lemma 11 *Let S be a set of weighted points in general position. Any $p \in S$ can be deleted in RT(S) by a sequence of flips of star faces satisfying the Necessary Conditions and by a $flip$ 41.*

Proof It is well known that a regular triangulation of points in general position is unique, and not affected by the order of insertion of points. In [21] it is proved that a point p can always be inserted into $RT(S\text{-}\{p\})$ by a $flip$ 14 and by a following sequence of flips of non-regular link faces, where each flip destroys a tetrahedron $\mathcal{T} \in RT(S\text{-}\{p\})$. If each $flip\ mn$ of a link face in this sequence is replaced with a $flip\ nm$ of a certain star face, then these flips applied in the reverse order (with the last $flip$ 41) delete p in $RT(S)$. Each tetrahedron $\mathcal{T}$, $p \notin \mathcal{T}$, created by these flips is regular with respect to all points $S - \{p\}$ and therefore fulfills the Necessary Conditions.□

By the Lemma 11, there always exists a sequence of flips of star faces which deletes p in $RT(S)$. But the problem is, how to find this sequence – all faces are regular, so it is not clear, which faces should be flipped (in contrast to insertion of a point into RT, where only non-regular faces are flipped). Here the Lemma 10 is helpful. By this Lemma, if w_p is set to $-\infty$, at least 6 star faces become non-regular (it can be also proved that at least one of them is flippable). A sequence of flips of these non-regular star faces (step 2) is shrinking $\mathcal{P}(p)$. A $flip$ 32 of a non-regular star face reduces $deg(p)$ by 2, a $flip$ 23 does not change $deg(p)$, only shrinks the volume of $\mathcal{P}(p)$. Unfortunately this flipping strategy does not assure that $deg(p)$ is always reduced to 4 – it is possible that after several flips each star face is non-flippable and $deg(p)> 4$. If this happens, a few link faces have to be flipped (step 3). A flip of a link face expands $\mathcal{P}(p)$ – a $flip$ 23 increases $deg(p)$, a $flip$ 32 expands the volume of $\mathcal{P}(p)$. A loop of the steps 2 a 3 could be seen as a backtracking looking for a sequence of flips which successfully reduces $deg(p)$ to 4. Because flips are applied in a random order in the steps 2 and 3, the algorithm cannot get stuck and endlessly repeat one bad sequence of flips, but step by step (flip by flip) it randomly traverses all potentially regular triangulations of $\mathcal{P}(p)$ and sooner or later it deletes p successfully. Unfortunately, if the algorithm is very unlucky, the number of performed flips is unbounded.

Now we show that $RT(S\text{-}\{p\})$ obtained by the basic algorithm is regular.

Lemma 12 *Let p be successfully deleted in RT(S) by the basic algorithm. The resulting RT(S-{p}) is regular.*

Proof We will show that all faces of $\mathcal{T}$ created in the step 2 or 3, $\mathcal{T} \notin star(p)$, are regular. Let $\mathcal{T}$, $\mathcal{T} \notin star(p)$, be a tetrahedron created in the step 2 by a flip of a star face satisfying the Necessary Condition 1. Due to Lemma 10, the faces of $\mathcal{T}$ which $\mathcal{T}$ shares with $star(p)$ are regular. The remaining faces of $\mathcal{T}$ have to be also regular, because $\mathcal{T}$ fulfills the Necessary Condition 1 (see Remark 1). So all faces of $\mathcal{T}$ are regular (although $\mathcal{T}$ can be globally non-regular). In the step 3, all tetrahedra created by a flip of a link face are in $star(p)$. So during the steps 2 and 3, only star faces can be non-regular. When $deg(p)= 4$, all the 6 remaining star faces are removed by a $flip$ 41. Some faces of the tetrahedron created by this flip can be non-regular, but all the other faces in $RT(S\text{-}\{p\})$ are regular. It can be proved that all the non-regular faces are destroyed by flips in the step 4 (the proof is based on the algorithm of Incremental topological flipping [21]).$\square$

Let k be the initial $deg(p)$. In the worst case, k can be $O(|S|)$ for each point $p \in S$ – Shewchuk [43] proved that each point in S can be incident to $O(|S|)$ tetrahedra if the points of S lie on (or nearby) two nonintersectors. But in practice k is usually constant. As already mentioned, the number of performed flips can be unbounded in the worst case, but we have never noticed such a case and the expected number of flips is $O(k^2)$ in practice (see Section 7.2.6).

7.2.3 Hybrid Algorithm

The hybrid algorithm combines the basic algorithm with global checks. Its goal is to ensure that each point p is deleted in a finite time and to keep a low average number of the tests of regularity at the same time. First, we define the Sufficient Condition.

Sufficient Condition Let f be a flippable star face. We say that f satisfies the Sufficient Condition if each tetrahedron $\mathcal{T}$ created by the flip, $\mathcal{T} \notin star(p)$, is regular with respect to all vertices (weighted points) of *the initial* $\mathcal{P}(p)$.

The idea of the hybrid algorithm is simple. At the beginning, the hybrid algorithm tests only the Necessary Conditions (their verification is the local check). If a point p is not deleted by a certain number of flips, the algorithm replaces the Necessary Conditions by the Sufficient Condition. A verification of the Sufficient Condition involves more tests or regularity (it is the global check), but ensures that the hybrid algorithm terminates in a finite time.

The hybrid algorithm begins in exactly the same way as the basic algorithm – thus the weight w_p of a p is set to $-\infty$ (step 1), star faces satisfying the Necessary Conditions are flipped (step 2), if no such a face exists and $deg(p) > 4$, then a random number of link faces is flipped (step 3), and the algorithm continues with the step 2. Now, if the step 3 is entered "too many times" (i.e. we have bad luck and the algorithm progresses slowly), the steps 2 and 3 are slightly modified:

- In the step 2, the local check of the Necessary Conditions is replaced by the global check of the Sufficient Condition, i.e. a star face is flipped only if each tetrahedron $\mathcal{T}$ created by the flip, $\mathcal{T} \notin star(p)$, is regular with respect to all vertices (weighted points) of *the initial* $\mathcal{P}(p)$.
- In the step 3, the number of flipped link faces is not random any more, but it is increased by 1 each time the hybrid algorithm enters the step 3 (this formal change is used in Section 7.2.4 to prove that the algorithm terminates in a finite time).

There is a question, what "too many times" is. In our tests we have used a simple criterion – the algorithm starts to use the Sufficient Condition if the step 3 is entered more than $deg(p)/2$ times.

7.2.4 Hybrid Algorithm Terminates

Here we discuss that the hybrid algorithm always terminates in a finite time. Assume that p was not deleted "quickly enough" and the hybrid algorithm starts to check the Sufficient Condition instead of the Necessary Conditions. Two cases can be distinguished.

First, all tetrahedra outside $\mathcal{P}(p)$ are regular with respect to all points in S-$\{p\}$. According to [33], in such a case each sequence of flips of star faces satisfying the Sufficient Condition always deletes p successfully (step 2). Note that in this case no link face is flipped at all.

Second, some tetrahedra outside $\mathcal{P}(p)$ are non-regular with respect to some points in S-$\{p\}$. Because the hybrid algorithm does not modify tetrahedra outside *the initial* $\mathcal{P}(p)$, these non-regular tetrahedra have to lie inside *the initial* $\mathcal{P}(p)$. It can be proved that in such a case p cannot be deleted by flips of star faces satisfying the Sufficient Condition, i.e. the step 2 of the hybrid algorithm fails. Then the algorithm continues with the step 3. In this step, link faces are flipped, each flip destroys a tetrahedron outside $\mathcal{P}(p)$. We have to show that all non-regular tetrahedra lying outside $\mathcal{P}(p)$ (and inside *the initial* $\mathcal{P}(p)$) would be destroyed after a finite number of iterations of the steps 2 and 3. Consider the following:

- The number of link faces to be flipped is increased each time the step 3 is entered.
- The maximum possible number of tetrahedra inside *the initial* $\mathcal{P}(p)$ is $O(k^2)$, where k is the initial $deg(p)$.
- Flips modifying tetrahedra outside *the initial* $\mathcal{P}(p)$ are not allowed.

- If $\mathcal{P}(p)$ is not equal to *the initial* $\mathcal{P}(p)$, there always exists a *flippable* face f of $\mathcal{P}(p)$ which is not a face of *the initial* $\mathcal{P}(p)$ (this follows from [21]).

So after a finite number of iterations of the steps 2 and 3, the number of link faces to be flipped would be greater than a number of tetrahedra lying outside $\mathcal{P}(p)$ and inside *the initial* $\mathcal{P}(p)$, therefore, all these tetrahedra would be destroyed in one certain step 3 and $\mathcal{P}(p)$ becomes equal to *the initial* $\mathcal{P}(p)$. In this moment, all tetrahedra outside $\mathcal{P}(p)$ are regular and p can be deleted by any sequence of flips of star faces satisfying the Sufficient Condition. Note that this is the worst case. In an average case, the non-regular tetrahedra lying outside $\mathcal{P}(p)$ would be destroyed before $\mathcal{P}(p)$ becomes equal to *the initial* $\mathcal{P}(p)$.

7.2.5 Degenerate Cases

The hybrid algorithm assumes that all points in S are in general position, which is often not true in real datasets. A common solution is to simulate general position using a perturbation method. But perturbation methods require an exact arithmetic, which is not suitable for us. We use another approach – in this Section we describe a few modifications, which improve the robustness of the hybrid algorithm for points in non-general position.

The incremental flipping algorithm of construction of DT [27],[42] uses degenerate flips (see Section 3.1.2) to deal with points in non-general position. The main purpose of these flips is to avoid a creation of flat tetrahedra. A $flip$ 26, $flip$ $m2m$, respectively, is used, if a point should be inserted into a face, an edge, respectively. A $flip$ 44 is used to handle coplanar points in S. The hybrid algorithm uses the inversions of these flips as described below.

Flip 62 Assume that p is the last point inserted into $RT(S)$ (via an incremental flipping algorithm). If p was included into $RT(S)$ by the $flip$ 26, then $deg(p)$ cannot be reduced to 4 without creating a flat tetrahedron or a non-regular face. But $deg(p)$ can be reduced to 6 and then p can be deleted by the $flip$ 62. The hybrid algorithm should be modified as follows: if $deg(p)= 6$ at the end of the step 2 and p together with 3 points of $\mathcal{P}(p)$ are coplanar, then p is deleted by the $flip$ 62.

Flip 2mm If p were included into $RT(S)$ by the $flip$ $m2m$, the situation is similar to the previous one. Again, if $deg(p)> 6$ at the end of the step 2, $\mathcal{P}(p)$ can be checked, whether p should be deleted by the $flip$ $2mm$. But this is tricky to implement, therefore, we use another solution in practice. If the step 2 has already run several times and still $deg(p)> 6$, than p is perturbed (so that any tetrahedron in $star(p)$ does not intersect another). Due to this, p stops being collinear with two other points of $\mathcal{P}(p)$ and $deg(p)$ can be decreased as usual.

Flip 44 The $flip$ 44 avoids creation of a flat tetrahedra in $RT(S)$. Let f be a star face satisfying the Necessary Conditions and flippable by the $flip$ 44. If the flip of f creates two tetrahedra $\mathcal{T}_1$, $\mathcal{T}_2 \notin star(p)$, then the flip is performed only if the face shared by $\mathcal{T}_1$,$\mathcal{T}_2$ is regular (in fact, this is an extension of the Necessary Condition 1 and it guarantees that the resulting $RT(S\text{-}\{p\})$ is regular).

When five or more points in S are orthogonal to the same weighted point (an analogy of cospherical points in DT), $RT(S)$ is not unique and it is possible that p cannot be deleted from $RT(S)$ without modifications of tetrahedra outside *the initial* $\mathcal{P}(p)$, i.e. *the initial* $\mathcal{P}(p)$

cannot be triangulated. A typical example is the Schönhardt polyhedron (see Fig. 4.3). In [17], this problem is solved using a perturbation. [33] use a different approach. By their experience, when *the initial* $\mathcal{P}(p)$ cannot be triangulated, then *the initial* $\mathcal{P}(p)$ always contains two neighboring coplanar faces, whose diagonal can be flipped using a $flip$ 44 of a star face – this flip modifies two tetrahedra outside *the initial* $\mathcal{P}(p)$. After one or more these flips, the resulting $\mathcal{P}(p)$ can be triangulated. Our tests agree with this claim and therefore we use a similar approach – in the step 2, we allow a $flip$ 44 modifying two tetrahedra inside *the initial* $\mathcal{P}(p)$ and two tetrahedra outside *the initial* $\mathcal{P}(p)$.

7.2.6 Experimental Results

We have compared the basic and the hybrid algorithms with our implementation of Ledoux's algorithm. The algorithms have been tested on a few hundreds of datasets, together with nearly 400 000 points to be deleted. The normalized histogram of $deg(p)$ of the used datasets is showed in Fig. 7.8. The most frequent values of $deg(p)$ in the used datasets are in a range 20–28.

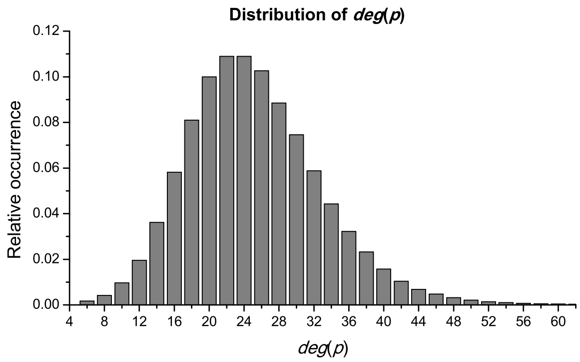


Figure 7.8: Normalized histogram of $deg(p)$.

We have omitted the comparison of the running times of the algorithms, because we have not optimized their implementations yet. Instead, we have focused on the average number of flips and first of all on the average number of regularity tests. Also we have been interested in how often the hybrid algorithm uses the global checks.

Because it is possible that the basic algorithm does not terminate in a finite time, we limited its maximal number of flips to $100 * k^2$, where $k = deg(p)$. If a number of flips performed by the basic algorithm exceeded this bound, the deletion of a point was skipped. In the used datasets, this happened only to 30 points from 400 000. The hybrid algorithm terminates always in a finite time thanks to the use of the global checks. The Fig. 7.9 shows that the global checks were used only by the deletion of very small percent of points – altogether by the deletion of only 661 points $\approx$ 0.17%).

The average number of performed flips is shown in Fig. 7.10. The number of flips performed by Ledoux's algorithm needs to delete a point p is approximately $O\left((deg(p))^{1.4}\right)$. The basic and the hybrid algorithm perform significantly more flips – approximately $O\left((deg(p))^{2}\right)$. This is not pleasant, but for the time complexity of the algorithm is crucial the number of performed regularity tests, not the number of flips.

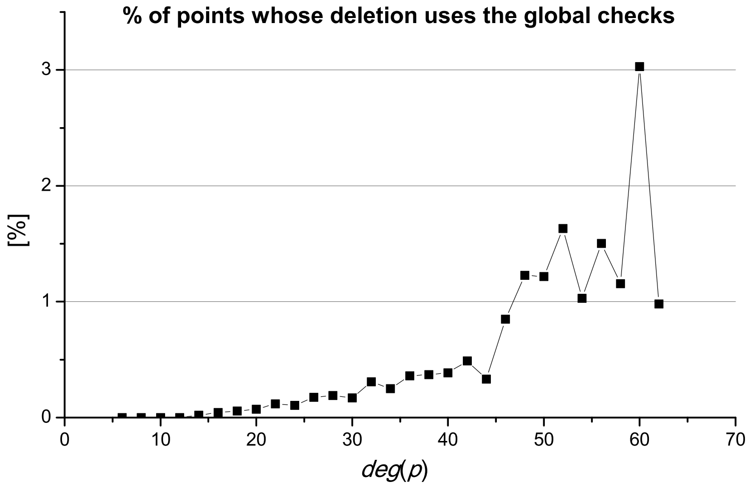


Figure 7.9: Percentage of points whose deletion uses the global checks in the hybrid algorithm.

Finally, Fig. 7.11 shows the average number of regularity tests of all three algorithms. We remind that the regularity test is a basic operation of the algorithms for point deletion and a number of these tests determines their time complexity. Because Devillers's algorithm (described in Section 4.1) checks regularity of each ear in the same way as Ledoux's algorithm checks regularity of each flip, we can expect that Devillers's algorithm performs the same number of regularity tests as Ledoux's algorithm. In our comparison, the hybrid and the basic algorithms perform approximately $O\left((deg(p))^2\right)$ regularity tests. Ledoux's algorithm needs more tests – approximately $O\left((deg(p))^{2.4}\right)$.

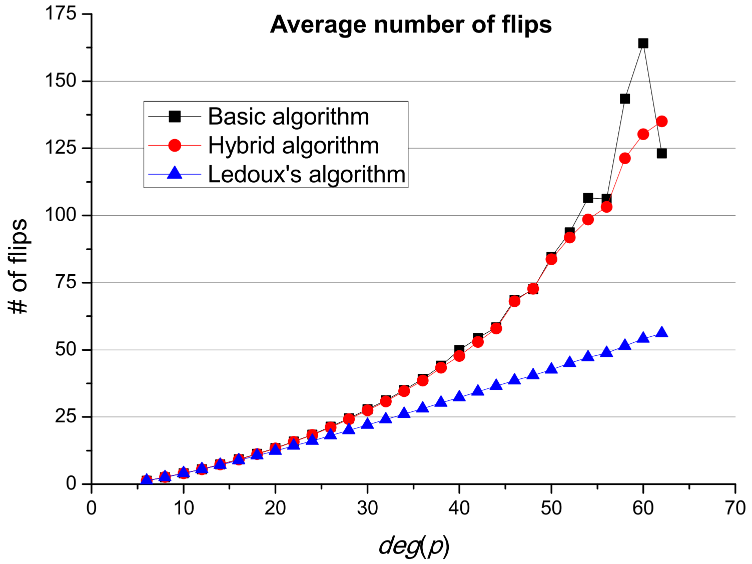


Figure 7.10: Average number of flips.

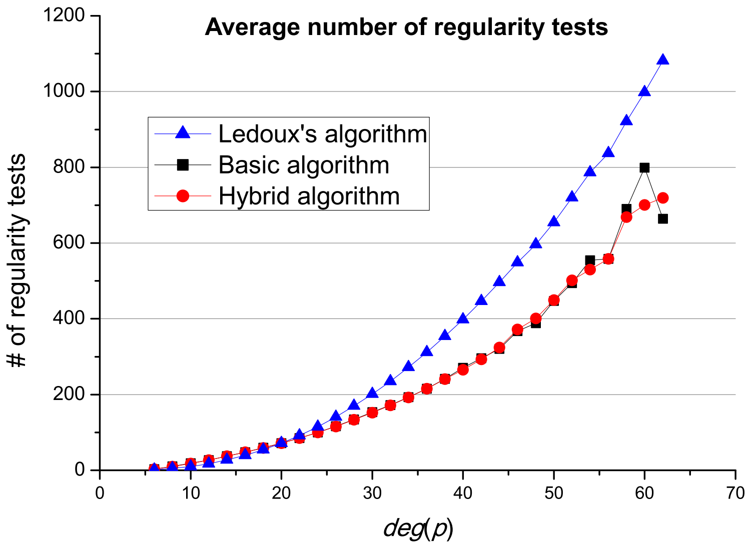


Figure 7.11: Average number of the regularity tests.

Chapter 8

Conclusion and Future Work

This thesis was focused on three-dimensional regular triangulations within the context of the dynamic variable data We described several algorithms for construction of RT and also algorithms allowing to delete a point in a triangulation. Further we discussed how to maintain kinetic and dynamic triangulations. Also we paid attention to the practical applications of regular triangulations, especially in biochemistry. We showed how regular triangulations can be used for the analysis of channels in protein molecules.

Further the results of author's research were presented. We proposed a novel method of a channel computation in a sequence of protein snapshots (in a dynamic protein) and we showed that this method is 4-8 times faster than the previous method. The next topic of the research was focused on a deletion in regular triangulations – we proposed a randomized algorithm that allows to delete a point in a regular or Delaunay triangulation by a sequence of flips.

Our implementation of the regular triangulation is used by several other researchers. Martin Maňák (a PhD student at the University of West Bohemia) employs it to speed up the computation of Euclidean Voronoi diagrams. Petr Brož (a PhD student at the University of West Bohemia) uses it for a path planning in a dynamic environment and we plan to write a joint article about this subject. Also Francois Anton (an associate professor at the Technical University of Denmark) and Tan Zhong Ming (a student at the Nanyang Technological University, Singapore) asked for our implementation of the regular triangulation.

The future work consists of two main parts. One part is focused on the applications of regular triangulation in the protein analysis, the other part is focused on the theoretical issues concerning the regular triangulation.

From our point of view, there are two main challenges in the protein analysis. The first task is an amino-acid replacement. A protein molecule consists of a chain of amino-acids. For some reasons, biochemists sometimes would like to replace a certain amino-acid in the protein by another. The biochemists know for each amino-acid a set of substitute amino-acids which can functionally replace the original amino-acid. But not each substitute amino-acid fits into each protein – it can collide with surrounding atoms. So, for a given original amino-acid and a given protein, our task is to select a substitute amino-acid from a set of candidates, which fits well into the protein. To solve this task, the geometric model of a protein described in Section 6.2.1 can be used.

The second task is the channel computation in a dynamic protein. We proposed the method which accelerated this process, but the time consumption is still an issue for a long sequences of protein snapshots. Therefore we would like to employ a dynamic triangulation for the channel computation to cut down the computing time.

In the theoretical part, we are going to continue with our work on the algorithms for point deletion. We believe that a performance of our proposed algorithm for deletion of a point by a sequence of flips (see Section 7.2) can be improved. Except the flipping strategy described in Section 7.2.1, we have tried several more flipping strategies. Some of them significantly accelerated the convergence of the algorithm, but, in a small percent of cases, the resulting triangulation contains a few non-regular tetrahedra after the deletion of a point. So we would like to design a better flipping strategy, which will still ensure that the resulting triangulation is regular (as well as the proposed strategy does) and which will yield a fast convergence.

Further we would like to explore the algorithm of sewing, mentioned in Section 4.3. By a deletion of a single point, this algorithm is probably slower than the other algorithms for deletion described in Section 4, but in contrast to them, it has a potential to allow a deletion of more points simultaneously. This can be often useful and we would like to use it in the amino-acid replacement described above.

As a next step, we would like to combine our algorithm for deletion with some techniques of a point movement to design an effective method of maintaining a dynamic regular triangulation. As already mentioned, we would like to optimize it for for a fast computation of channels in dynamic proteins. This will nicely close our research circle.

Bibliography

[1] Gerhard Albers, Joseph S. B. Mitchell, Leonidas J. Guibas, and Thomas Roos. Voronoi diagrams of moving points. *Internat. J. Comput. Geom. Appl*, 8:365–380, 1998.

[2] Nina Amenta, Sunghee Choi, and Ravi Krishna Kolluri. The power crust. In *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, New York, NY, USA, 2001. ACM.

[3] Franz Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1):78–96, 1987.

[4] Franz Aurenhammer. Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[5] Chandrajit L. Bajaj, Valerio Pascucci, Ariel Shamir, Robert J. Holt, and Arun N. Netravali. Dynamic maintenance and visualization of molecular surfaces. *Discrete Appl. Math.*, 127(1):23–51, 2003.

[6] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[7] Tilo Beyer, Gernot Schaller, Andreas Deutsch, and Michael Meyer-Hermann. Parallel dynamic and kinetic regular triangulation in three dimensions. *Computer Physics Communications*, 172(2):86 – 108, 2005.

[8] Adrian Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, 24(2):162–166, 1981.

[9] Petr Brož. Plánování cest pro virtuální realitu [master thesis, supervisior Ivana Kolingerová]. *University of West Bohemia, Faculty of Applied Sciences*, 2008.

[10] Moses Charikar and Sudipto Guha. Improved combinatorial algorithms for the facility location and k-median problems. In *FOCS '99: Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 378, Washington, DC, USA, 1999. IEEE Computer Society.

[11] Siu-Wing Cheng, Tamal K. Dey, Herbert Edelsbrunner, Michael A. Facello, and Shang-Hua Teng. Sliver exudation. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 1–13, New York, NY, USA, 1999. ACM.

[12] Paolo Cignoni, C. Montani, and Roberto Scopigno. Dewall: A fast divide and conquer Delaunay triangulation algorithm in E^d. *Computer-Aided Design*, 30(5):333–341, 1997.

[13] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational Geometry: Algorithms and applications.* Springer-Verlag Berlin Heidelberg, 1997.

[14] Olivier Devillers. Improved incremental randomized Delaunay triangulation. *CoRR*, 1999.

[15] Olivier Devillers. On deletion in Delaunay triangulations. In *SCG '99: Proceedings of the fifteenth annual symposium on Computational geometry*, pages 181–188, New York, NY, USA, 1999. ACM.

[16] Olivier Devillers, Sylvain Pion, and Monique Teillaud. Walking in a triangulation. *Internat. J. Found. Comput. Sci*, 13:106–114, 2001.

[17] Olivier Devillers and Monique Teillaud. Perturbations and vertex removal in a 3D Delaunay triangulation. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 313–319, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[18] Gustav Lejeune Dirichlet. über die reduction der positiven quadratischen formen mit drei unbestimmten ganzen zahlen. *Journal für die reine und angewandte Mathematik*, 40:209–227, 1850.

[19] Herbert Edelsbrunner, Michael Facello, and Jie Liang. On the definition and the construction of pockets in macromolecules. Technical report, Champaign, IL, USA, 1995.

[20] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9(1):66–104, 1990.

[21] Herbert Edelsbrunner and Nimish R. Shah. Incremental topological flipping works for regular triangulations. In *SCG '92: Proceedings of the eighth annual symposium on Computational geometry*, pages 43–52, New York, NY, USA, 1992. ACM.

[22] Michael A. Facello. Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. *Comput. Aided Geom. Des.*, 12(4):349–370, 1995.

[23] Jean-Albert Ferrez. Dynamic triangulations for efficient 3D simulation of granular materials. *École Polytechnique Fédérale de Lausanne*, 2001.

[24] Steven Fortune. A sweepline algorithm for Voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 313–322, New York, NY, USA, 1986. ACM.

[25] Carl Friedrich Gauss. Recursion der "untersuchungen über die eigenschaften der positiven ternären quadratische formen von ludwig august seeber, dr. der philosophie, ordentl. professor der universität in freiburg, 1831, 248 s. in 4.". *Journal für die reine und angewandte Mathematik*, 20:312–320, 1840.

[26] Leonidas Guibas and Daniel Russel. An empirical comparison of techniques for updating Delaunay triangulations. In *SCG '04: Proceedings of the twentieth annual symposium on Computational geometry*, pages 170–179, New York, NY, USA, 2004. ACM.

[27] Barry Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Comput. Aided Geom. Des.*, 8(2):123–142, 1991.

[28] Deok-Soo Kim, Cheol-Hyung Cho, Youngsong Cho, Chung In Won, and Donguk Kim. Pocket recognition on a protein using Euclidean Voronoi diagram of atoms. In *Computational Science and Its Applications ICCSA 2005*, pages 707–715, 2005.

[29] Deok-Soo Kim, Youngsong Cho, and Donguk Kim. Edge-tracing algorithm for Euclidean Voronoi diagram of 3D spheres. In *Canadian Conference on Computational Geometry*, pages 176–179, 2004.

[30] Deok-Soo Kim, Youngsong Cho, and Donguk Kim. Euclidean Voronoi diagram of 3D balls and its computation via tracing edges. *Computer-Aided Design*, 37(13):1412 – 1424, 2005.

[31] Charles L. Lawson. Properties of n-dimensional triangulations. *Comput. Aided Geom. Des.*, 3(4):231–246, 1987.

[32] Hugo Ledoux. The kinetic 3D Voronoi diagram: A tool for simulating environmental processes. In *Advances in 3D Geoinformation Systems*, chapter 20, pages 361–380. Springer-Verlag Berlin Heidelberg, 2008.

[33] Hugo Ledoux, Christopher M. Gold, and George Baciu. Flipping to robustly delete a vertex in a Delaunay tetrahedralization. In *ICCSA (1)*, pages 737–747, 2005.

[34] Pavel Maur and Ivana Kolingerová. The employment of regular triangulation for constrained Delaunay triangulation. In *Computational Science and Its Applications ICCSA 2004*, pages 198–206, 2004.

[35] Martin Maňák. Voronoi diagrams for spheres in E^3 [master thesis, supervisior Ivana Kolingerová]. *Charles University in Prague, Faculty of Mathematics and Physics*, 2008.

[36] Petr Medek, Petr Beneš, and Jiři. Sochor. Computation of tunnels in protein molecules using Delaunay triangulation. In *Journal of WSCG*, volume 15, pages 107–114, 2007.

[37] A. Meyerson. Online facility location. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 426, Washington, DC, USA, 2001. IEEE Computer Society.

[38] Ernst P. Mucke, Isaac Saias, and Binhai Zhu. Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations. In *In Proc. 12th Annu. ACM Sympos. Comput. Geom*, pages 274–283, 1996.

[39] Gernot Schaller and Michael Meyer-Hermann. Kinetic and dynamic Delaunay tetrahedralizations in three dimensions. *Computer Physics Communications*, 162:9–23, 2004.

[40] Jonathan Richard Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete & Computational Geometry*, 18(3):305–363, oct 1997.

[41] Jonathan Richard Shewchuk. Sweep algorithms for constructing higher-dimensional constrained Delaunay triangulations. In *SCG '00: Proceedings of the sixteenth annual symposium on Computational geometry*, pages 350–359, New York, NY, USA, 2000. ACM.

[42] Jonathan Richard Shewchuk. Updating and constructing constrained Delaunay and constrained regular triangulations by flips. In *SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry*, pages 181–190, New York, NY, USA, 2003. ACM.

[43] Jonathan Richard Shewchuk. Stabbing Delaunay tetrahedralizations. *Discrete Comput. Geom*, 32:343, 2004.

[44] Jiří Skála and Ivana Kolingerová. Clustering geometric data streams. In *SIGRAD 2007*, pages 17–23, 2007.

[45] Peter Su and Robert L. Scot Drysdale. A comparison of sequential Delaunay triangulation algorithms. In *Comput. Geom. Theory Appl*, pages 61–70. ACM, 1995.

[46] Marc Vigo, Núria Pla, and Josep Cotrina. Regular triangulations of dynamic sets of points. *Comput. Aided Geom. Des.*, 19(2):127–149, 2002.

[47] Georgy F. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques premier mémoire: sûr quelques propriétés des formes quadratiques positives parfaits. *Journal für die reine und angewandte Mathematik*, 133:97–178, 1907.

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

[49] Michal Zemek. Dělení prostoru pro rozsáhlá a měnící se data [master thesis, supervisior Ivana Kolingerová]. *University of West Bohemia, Faculty of Applied Sciences*, 2007.

[50] Michal Zemek and Ivana Kolingerová. Hybrid algorithm for deletion of a point in regular and delaunay triangulation. In *Spring Conference on Computer Graphics* [accepted, in print], 2009.

[51] Michal Zemek, Jiří Skála, Ivana Kolingerová, Petr Medek, and Jiří Sochor. Fast method for computation of channels in dynamic proteins. In *Vision, Modeling, and Visualization 2008*, pages 333–342, 2008.

Activities

Reviewed Publications

- Michal Zemek and Ivana Kolingerová. Hybrid algorithm for deletion of a point in regular and Delaunay triangulation. In *Spring Conference on Computer Graphics*, pages 149-156, Budmerice, Slovakia, 2009.
- Michal Zemek, Jiří Skála, Ivana Kolingerová, Petr Medek, and Jiří Sochor. Fast method for computation of channels in dynamic proteins. In *Vision, Modeling, and Visualization 2008*, pages 333-342, Konstanz, Germany, 2008.
- Michal Zemek. Dělení prostoru pro rozsáhlá a měnící se data. *Master thesis*, supervisior Ivana Kolingerová, University of West Bohemia, Faculty of Applied Sciences, 2007.

Non-Reviewed Publications

- Michal Zemek, Ivana Kolingerová, Petr Medek, and Jiří Sochor. Regular triangulations and tunnels in proteins. *Technical report*, University of West Bohemia, Faculty of Applied Sciences, 2007.

Related Talks

- Voronoiovy diagramy. Center of Computer Graphics and Data Visualization, University of West Bohemia, Czech Republic, May 2009.
- Tunnels in Static and Dynamic Protein Molecules. Center of Computer Graphics and Data Visualization, University of West Bohemia, Czech Republic, May 2008.
- Protein Molecules and Regular Triangulation. University of Maribor, Slovenia, November 2007.

Stays Abroad

- University of Maribor, Slovenia, November 2007, 1 week.

Participation in Projects

- the Ministry of Education of the Czech Republic, Project No. LC06008 (Center of Computer Graphics - National Network of Fundamental Research Centers).
- the Grant Agency of the Czech Republic, Project No. 201/07/0927 (Triangulated Models for Haptic and Virtual Reality).