University of West Bohemia
Faculty of Applied Sciences

Application of Computational Geometry to Modeling and Visualization of Proteins

Mgr. Martin Maňák

Doctoral thesis
submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in Computer Science and Engineering

Supervisor: Prof. Dr. Ing. Ivana Kolingerová
Department: Department of Computer Science and Engineering

Pilsen 2016

Západočeská univerzita v Plzni
Fakulta aplikovaných věd

Využití výpočetní geometrie pro modelování a vizualizaci proteinů

Mgr. Martin Maňák

Disertační práce
k získání akademického titulu doktor
v oboru Informatika a výpočetní technika

Školitel: Prof. Dr. Ing. Ivana Kolingerová
Katedra: Katedra informatiky a výpočetní techniky

Plzeň 2016

Prohlášení

Předkládám tímto k posouzení a obhajobě disertační práci zpracovanou na závěr doktorského studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni. Prohlašuji, že tuto práci jsem zpracoval samostatně s použitím odborné literatury a dostupných pramenů uvedených v seznamu, jenž je součástí této práce.

V Plzni dne 29. června 2016 Mgr. Martin Maňák

Contents

Abstract

Modeling and visualization of protein models help domain experts to better understand the function of these bio-molecules. The geometric model is just a collection of partially overlapping spheres representing individual atoms. The inspection of this model with a spherical collision-avoiding probe may reveal cavities among atoms. The evaluation of such cavities may show the spots of bio-chemical activity. Spatial-subdivision concepts from the field of computational geometry, in particular various kinds of Voronoi diagrams, are often used in the spatial analysis of these models. This thesis is focused on additively weighted Voronoi diagrams and their application to protein models because these diagrams provide complete information for the navigation of a collision-avoiding probe among protein atoms. An acceleration technique for a fast construction of these diagrams is presented. These diagrams may suffer by disconnected cases, which occur when the one-dimensional skeleton of Voronoi edges consists of several components. A newly proposed extension of the diagram construction algorithm guarantees the construction of all components by the algorithm. This extension improves the reliability of the construction algorithm and makes it applicable to general input data. A practical approach for interactive detection of cavities for probes of any size is presented. This approach combines the additively weighted Voronoi diagram for a fast detection of cavities with GPU ray casting to achieve fast rendering of their surfaces when the probe radius is changed. The implementation of these methods in the software tool CAVER Analyst make them available to the world-wide community of bio-chemists.

This dissertation thesis was supported by the following projects:

- GA201/09/0097 – Triangulated Models for Haptic and Virtual Reality
  Czech Science Foundation (GACR)
- GAP202/10/1435 – Analysis and Visualization of Protein Structures
  Czech Science Foundation (GACR)
- CZ.1.05/1.1.00/02.0090 – NTIS - New Technologies for the Information Society
  European Regional Development Fund (ERDF)
- LO1506 – PUNTIS
  Czech Ministry of Education, Youth and Sports (MEYS)
- SGS-2013-029 – Advanced Computing and Information Systems
  University of West Bohemia (UWB)
- SGS-2016-013 – Advanced Graphics and Computational Systems
  University of West Bohemia (UWB)

Abstrakt

Modelování a vizualizace modelů proteinů pomáhá expertům lépe porozumět funkci těchto biomolekul. Geometrický model je tvořen několika sférami reprezentujícími jednotlivé atomy. Zkoumání tohoto modelu pomocí sférické sondy, která se vyhýbá kolizím s atomy, může odhalit dutiny mezi atomy. Vyhodnocování takových dutin pak může ukázat místa bio-chemické aktivity. Koncepty dělení prostoru z oblasti výpočetní geometrie, zejména různé druhy Voronoi diagramů, jsou často používány k prostorové analýze těchto modelů. Tato disertační práce je zaměřena na aditivně vážené Voronoi diagramy a jejich aplikaci na modely proteinů, protože tyto diagramy poskytují kompletní informaci k navigaci nekolizní sondy mezi atomy proteinu. Je zde prezentována urychlovací technika pro rychlou konstrukci těchto diagramů. Tyto diagramy mohou být postiženy nespojitostmi, které nastávají, když jednodimenzionální kostra Voronoi hran je tvořena několika komponentami. Nově navržené rozšíření algoritmu konstrukce diagramu garantuje konstrukci všech komponent. Toto rozšíření zlepšuje důvěryhodnost konstrukčního algoritmu a dělá ho aplikovatelným na obecnější vstupní data. Dále je prezentrována praktická metoda pro interaktivní detekci a zobrazení dutin detekovaných jakoukoliv sondou. Tato metoda kombinuje aditivně vážený Voronoi diagram pro rychlou detekci dutin s GPU metodou vrhání paprsku pro rychlé stínování povrchu dutin vždy když je změněn poloměr sondy. Díky implementaci těchto metod v softwarovém nástroji CAVER Analyst jsou tyto metody dostupné celosvětové komunitě bio-chemiků.

Tato disertační práce byla podporována následujícími projekty:

- GA201/09/0097 – Triangularizované modely pro haptiku a virtuální realitu
  Grantová agentura České republiky (GAČR)
- GAP202/10/1435 – Analýza a vizualizace proteinových struktur
  Grantová agentura České republiky (GAČR)
- CZ.1.05/1.1.00/02.0090 – NTIS - Nové technologie pro informační společnost
  Evropský fond pro regionální rozvoj (ERDF)
- LO1506 – PUNTIS - Podpora udržitelnosti centra NTIS
  Ministerstvo školství, mládeže a tělovýchovy (MŠMT)
- SGS-2013-029 – Pokročilé výpočetní a informační systémy
  Západočeská univerzita v Plzni (ZČU)
- SGS-2016-013 – Pokročilé grafické a výpočetní systémy
  Západočeská univerzita v Plzni (ZČU)

Acknowledgement

Hereby I would like to thank my thesis advisor Ivana Kolingerová for a great support from the beginning to the end. The cooperation with the Masaryk university in Brno, Czech Republic, namely with Jiří Sochor, Barbora Kozlíková and their team, was also very important. Many thanks belong to my family. They always believed that I will finish the thesis some day.

Chapter 1

Introduction

Proteins are bio-molecular structures that play an essential role in living organisms. Some proteins are basic building blocks, other control biological processes, serve as receptors, transfer signals, etc. The function of a protein highly depends on its 3-dimensional structure. The ability to explore this complex structure is important for a better understanding of biological processes, for the study of the evolution cycle of viruses, for the development of new drugs or improving existing ones, for the engineering of new proteins that perform specific tasks, etc.

In bio-chemistry, methods are known to find out the sequence of amino acids that constitute a protein and to measure the positions of atoms in a protein. Processing this information on computers saves a huge amount of time and provides a much deeper insight into the function of a protein than experiments performed in laboratory conditions.

For the purposes of spatial analysis, a protein is commonly modeled as a set of partially overlapping spheres, which represent the atoms of the protein. The space outside atoms is usually examined by a spherical probe of some predefined radius. This simplification of a complex reality has been successfully used by scientists for decades. The tasks may include finding internal cavities, analysis of escape paths for the probe, or evaluation and visualization of the boundary between the space that can be reached by the probe and the space where the probe collides with atoms. This boundary is called the Connolly surface [17].

Concepts from the field of computational geometry can help with the analytic solution of these tasks. Various kinds of Voronoi diagrams have been used as a partitioning scheme that assigns a region of space to each atom [95]. Among them, the aw-Voronoi diagram (additively weighted) provides the maximum amount of information [50]. Each aw-Voronoi region contains all the points having the (signed) distance from the surface of the atom sphere less than or equal to the distance from any other atom sphere. Points in the boundary of these regions are equidistant to the surface of the corresponding atom spheres. Thanks to this property, the aw-Voronoi diagram is well suited for the analysis of atom spheres with respect to a spherical probe.

Aw-Voronoi diagrams are more complicated than ordinary diagrams because they have non-linear region boundaries. Algorithms for their construction are slower and more complex than those for ordinary diagrams. Furthermore, the union of Voronoi edges can consist of several components. Disconnected cases are problematic for current algorithms.

Nevertheless, using aw-Voronoi diagrams has some benefits. For instance, the detection of cavities that can accommodate a spherical probe of the given radius is a common task in biochemistry, but the probe radius is often limited for efficiency reasons. If the aw-Voronoi

diagram is used, there is no need for such a limit and cavities can be quickly detected for any probe. Another example of a benefit can be the detection of paths along which a smaller molecule can reach the active site of an enzyme and trigger some chemical reaction there.

Graphical visualization of results is also important. For instance, the cavities detected in a protein for some given probe radius can be visualized as the corresponding parts of the Connolly surface. However, changes of the probe radius will not be interactive if the surface is represented traditionally as a triangular mesh because the surface will have to be recomputed whenever the probe radius is changed. Modern approaches [68, 78] use sophisticated GPU programs instead, but only one side of the surface is rendered correctly.

1.1 Problem Definition

This thesis aims to improve the situation in the spatial analysis of protein models by using aw-Voronoi diagrams. The first objective is a fast and reliable computation of these diagrams because previous algorithms were not designed to handle all disconnected cases that may occur in this type of diagrams. The second objective is to devise new methods for interactive detection and visualization of cavities in protein models and the computation of cavity properties. Cavities are detected by a spherical probe. Variations of the probe radius are considered in this thesis to gain more information about cavities.

1.2 Summary of Contributions

The first contribution [84] is related to the computation of aw-Voronoi diagrams. A new filtering approach was developed to reduce the running time of a diagram construction algorithm. It is based on an older idea of other authors but uses more sophisticated dynamic filters, which can be searched through in a lazy way. This is achieved by using the Delaunay triangulation of atom centers instead of storing atoms in a regular grid. The proposed filtering approach was later improved to work also in the case when the input contains spheres of very different radii [116].

The second contribution [85] is an extension of a diagram construction algorithm to handle disconnected cases. This problem was unsolved at least for a decade. The proposed method is based on a hierarchy of aw-Voronoi components. The bottom of the hierarchy is easily discoverable, but inner nodes can be missing and these need to be computed. The proposed method systematically removes leaf nodes, detects missing nodes and computes their corresponding components. This contribution greatly improves the reliability and extends the usability of the diagram construction algorithm to general input data.

The third contribution [82] is interactive detection and visualization of cavities in proteins. Cavities are detected by processing a subset of the aw-Voronoi diagram, their Connolly surface is rendered via GPU ray casting. Both parts, i.e., the computation of cavities via aw-Voronoi diagrams and the fast rendering via GPU ray casting, have been introduced earlier. The novelty is in their combination. The previous method of cavities detection was not interactive because it was using a triangulation of the whole Connolly surface. Previous rendering methods based on GPU ray casting were not able to hide selected cavities, render both sides of the surface correctly, or change the probe radius to higher values. The proposed method overcomes all these shortcomings and computes advanced properties of cavities, which can be used by domain experts to evaluate cavities.

The fourth contribution [81] is a method for computing all possible cavities detectable in a protein model. Cavities are usually detected w.r.t. the radius of a spherical probe. The proposed method simulates continuous shrinking of the probe and detects the creation and merging of cavities. This information is then organized into a hierarchy and filtered according to user-defined criteria to get a more concise view on the structure of the cavities in a protein model.

All proposed methods have been implemented and presented on international conferences or in impacted journals. The methods related to the construction of aw-Voronoi diagrams are freely available via the open source library AwVoronoi [15]. The methods related to the analysis of proteins were integrated as plugins into the software tool CAVER Analyst [67], which has been developed on Masaryk University in Brno, Czech Republic.

1.3 Organization of the Thesis

The first half of this thesis explains the necessary background and summarizes the state of the art. Chapter 2 is an introduction to the structure of proteins and their geometric models. Chapter 3 provides the background from the field of computational geometry, which is necessary for spatial analysis of protein models. This background includes Voronoi diagrams, their dual triangulations and related alpha/beta complexes. Chapter 4 gives a short historical overview of using the computational geometry for spatial analysis of proteins, and focuses on the state-of-the-art techniques of the computation and visualization of Connolly surfaces and detection of cavities in protein models. Chapter 5 provides a comprehensive description of the edge-tracing algorithm for the construction of aw-Voronoi diagrams.

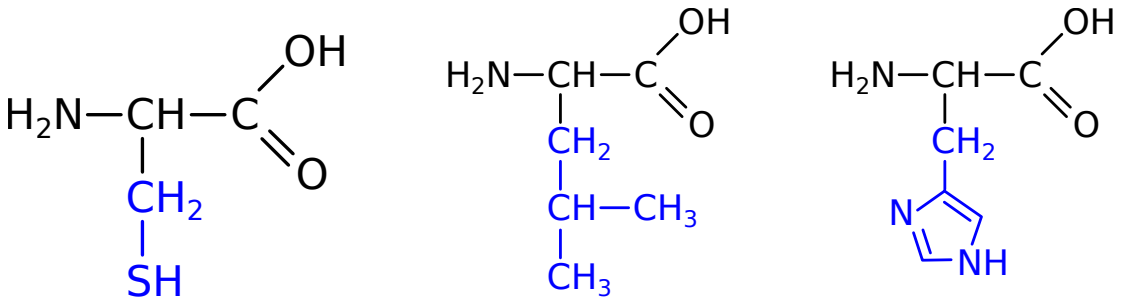
Contributions are in the second half. Chapter 6 describes an acceleration technique for a fast computation of aw-Voronoi diagrams. Chapter 7 describes an extension of the edge-tracing algorithm, which enables the algorithm to solve also difficult cases of disconnected Voronoi skeletons. Chapter 8 describes a method for interactive analysis of protein models, in particular a fast computation and rendering of Connolly surfaces for various probe sizes with an emphasis on the detection of empty cavities and the computation of their properties. Chapter 9 contains the description of the method for the computation of all cavities detectable by any probe in a protein model. Chapter 10 suggests possible directions for future work and Chapter 11 concludes this thesis.

Chapter 2

Protein Models

This chapter is a brief introduction to the structure of proteins and their models.

The building blocks of proteins are amino acids. There are 20 standard kinds of amino acids found in proteins, some of them are shown in Figure 2.1. Each amino acid consists of just a few atoms (H - hydrogen, C - carbon, N - nitrogen, S - sulfur, O - oxygen, etc.), held together by strong covalent bonds, which are depicted as line segments. Each amino acid has a common part that enables the amino acid to be joined into a chain, and a residual part specific to each kind. The way how amino acids form a chain is depicted in Figure 2.2. A protein consists of one or more such chains.



(a) Cysteine (b) Leucine (c) Histidine

Figure 2.1: Examples of amino acids with residual parts highlighted in blue

The primary structure of a protein is a sequence of amino acid residues in a chain. The order is encoded in DNA and chains are synthesized in the process of transcription and translation, performed by huge molecular structures, ribosomes. Some parts of the chain are then arranged into sheets, other parts into helices, held together by hydrogen bonds. This is the secondary structure of a protein and it is illustrated in Figure 2.3. These sheets and helices are fold into a stable three dimensional structure. This is a tertiary structure, which determines the function of a protein. The folding may be spontaneous or assisted by large proteins, chaperones.

For a study of the function of a protein, it is important to have some model, which would describe its tertiary structure and which would also include attractive and repulsive forces among atoms. The van der Waals model is often used for this. In this model, atoms are represented as spheres. If there is a strong attraction between atoms, e.g., via a covalent bond, then the spheres of these atoms overlap, otherwise the spheres should not overlap due to repulsive forces.

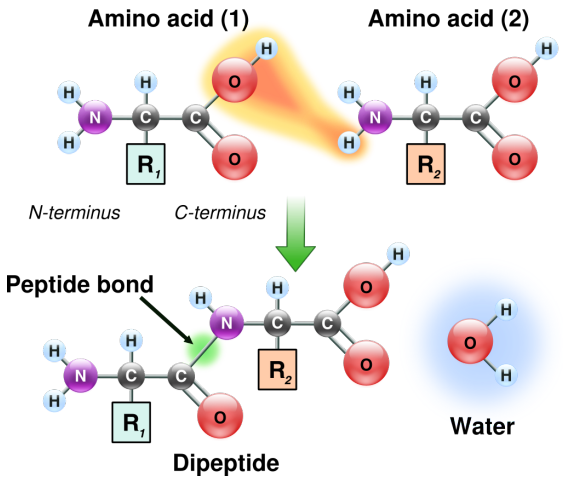


Figure 2.2: Amino acids join together to create a chain[1]

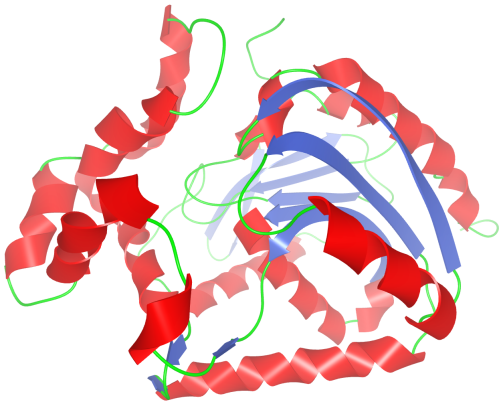


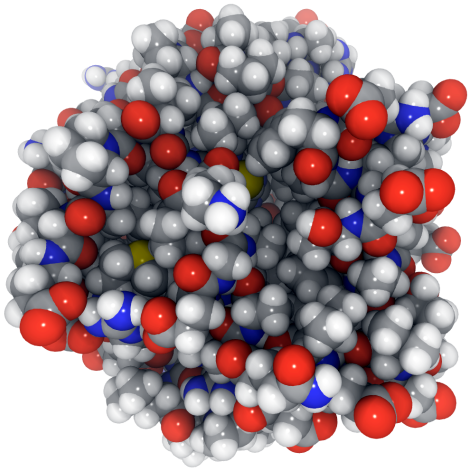
Figure 2.3: The secondary structure of a protein shows its motifs - helices in red, sheets in blue, and coils in green. Rendered by CAVER Analyst [67].

Van der Waals radii of elements were measured experimentally and are available as a set of constants [11]. Atomic coordinates can be obtained via NMR spectroscopy (nuclear magnetic resonance) or X-ray crystalography. Resolved coordinates together with addditional information are freely available in the Protein Data Bank [6] under a four-letter PDB ID. The unit of length is one ångstrom, $1\text{Å} = 10^{-10}\text{m}$.

An example of a small protein is depicted in Figure 2.4. Atom types are distinguished by colors. Hydrogen atoms are white, carbon gray, nitrogen blue, oxygen red and sulfur yellow. Spheres of different atom types also have different van der Waals radii. The protein in Figure 2.4 is a haloalkane dehalogenase enzyme, PDB ID: 1CQW [88]. Such enzymes are found in certain bacteria. They can be used, e.g., for the biodegradation of toxic compounds [97].

Proteins interact with small molecules. For instance, proteins are usually surrounded by water molecules. Hydrophobic parts of a protein are packed inside the protein, whereas hydrophilic parts are exposed to water. Some water molecules can also be trapped inside the protein to ensure its stability [114]. Another example are enzymes. They control chemical reactions by processing substrate molecules in the active sites of the enzymes. Interactions of these small molecules with

[1] Image taken from `https://en.wikipedia.org/wiki/Amino_acid`



(a) The van der Waals model. Atoms are represented as spheres.

(b) A simple visualization of the chain. Amino acid residues are excluded.

Figure 2.4: An example of a small protein. Rendered by CAVER Analyst [67].

proteins are usually studied on their van der Waals models. The small molecule is almost always approximated by a sphere of a fixed radius, the probe. The places where the probe does not collide with the van der Waals model are of particular interest. One of the ways how to visualize this information is by the boundary of the space occupied by all possible collision-avoiding probes of the given radius. This boundary is called the *Connolly surface* [16, 41, 99]. An example is depicted in Figure 2.5. This surface was originally introduced by Richards [99] as the surface left behind a probe, which rolls over the atoms. An alternative definition, which is also used in this thesis, was given by Greer and Bush [41]. Connolly [16] described this surface analytically and provided equations for the calculation of its area. This surface is sometimes called a solvent-excluded surface (SES) or just a molecular surface. Details about the terminology and arguments for using the term "Connolly surface" can be found in the survey of Kim et al. [64].

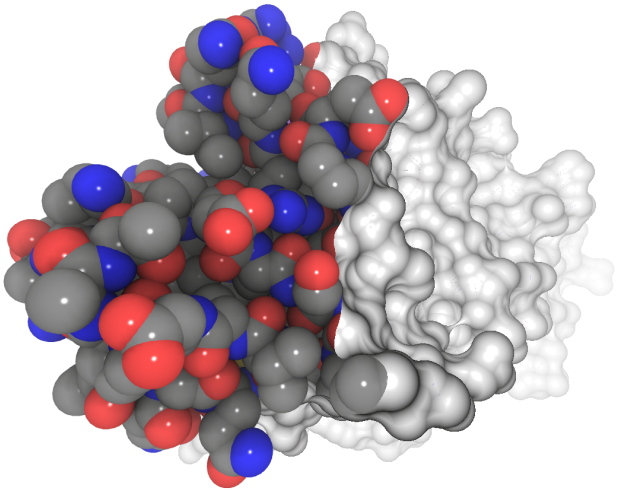


Figure 2.5: Connolly surface of a protein. Rendered by CAVER Analyst [67], using author's own plugin.

As it can be seen in Figure 2.6, the Connolly surface consists of parts of concave spherical triangles, saddle-shaped toroidal patches and convex spherical patches. Toroidal patches can self-intersect and there can be also intersections among spherical triangles. The parts behind these intersections do not belong to the Connolly surface and must be removed.

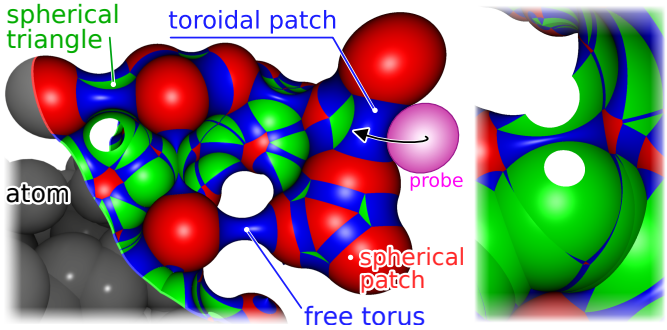
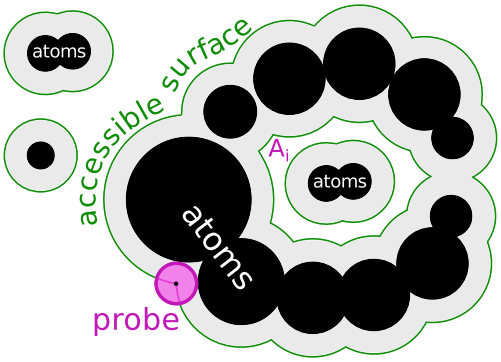


Figure 2.6: Connolly surface of a protein. The surface consists of parts of spherical triangles, toroidal and spherical patches. Rendered by CAVER Analyst [67], using author's own plugin

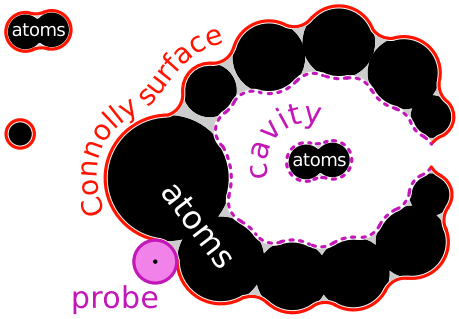
Inside a protein, there can be locations from which the spherical probe cannot escape without hitting some atoms. These locations are called cavities or voids. A *cavity* is a sub-space that can be occupied by a continuously moving probe of the given fixed size, avoiding collision with atoms. More formally, for a given set of atom spheres $\{s_1,\ldots,s_n\}$, where each s_i is given by its center $c_i \in \mathbb{R}^3$ and radius $r_i \in \mathbb{R}$, and for a given probe radius $r_p \in \mathbb{R}$, the connected components $A_1,\ldots,A_m$ of the set $A = \mathbb{R}^3 \setminus \bigcup_{i=1}^{n}\{x|x \in \mathbb{R}^3 \wedge ||x-c_i|| \leq r_i + r_p\}$ correspond one-to-one with *cavities*. Each A_i is the set of centers of all probes that belong to the cavity number i. The space of the i-th cavity is the set $\bigcup_{c\in A_i}\{x|x \in \mathbb{R}^3 \wedge ||x-c|| \leq r_p\}$, i.e., the union of all probes having the center in A_i and the probe radius equal to r_P.

The boundary of the set A is often called the *accessible surface* [71]. The space of a cavity can be visualized, e.g., by the part of its boundary that belongs to the Connolly surface, as it is depicted in Figure 2.7. Note that the surface does not have to be connected.

Empty space can also be analyzed for channels, which represent escape-paths for a flexible probe. Figure 2.8 illustrates cavities and channels on an example. It is again PDB ID: 1CQW, but



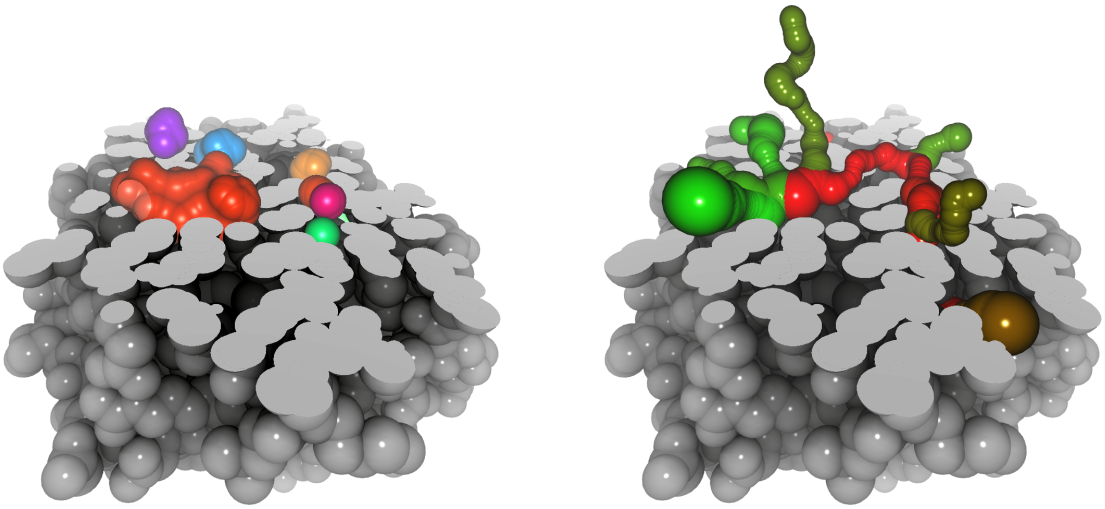
(a) The surface accessible to the probe center and a component A_i of the accessible space



(b) Connolly surface and the cavity corresponding to the probes with centers from A_i

Figure 2.7: The accessible and Connolly surface for a given probe radius and a cavity

without hydrogens. A part of the model has been cut off to see inside. Cavities are distinguished by colors. It is known that this protein has a cavity sufficiently large to accommodate a small substrate molecule in the active site, and a channel along which the substrate can get there.



(a) Connolly surface of cavities

(b) Channels visualized as filling spheres

Figure 2.8: Analysis of empty space performed by CAVER Analyst [67]. Cavities were computed by author's own plugin. Channels – Masaryk University in Brno, Czech Republic.

Chapter 3

Computational Geometry

The concepts discussed in this chapter are fundamental for the spatial analysis of proteins. Voronoi diagrams will be discussed first. They are very useful for the exploration of empty space among atom spheres and their understanding is essential for this thesis. After that, dual structures (triangulations) and related concepts of alpha/beta shapes will be discussed.

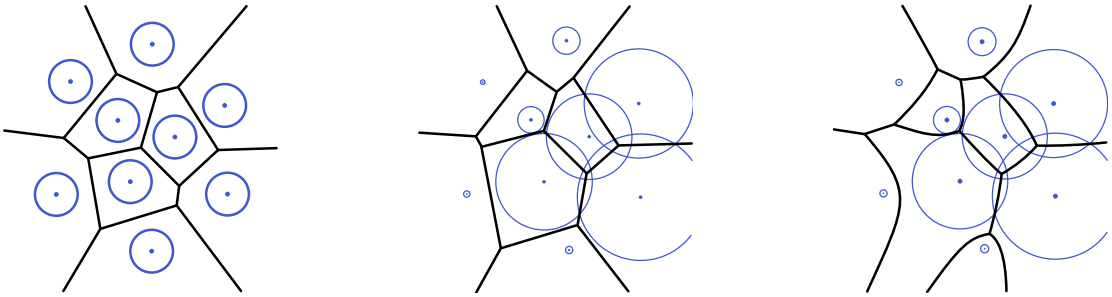
3.1 Voronoi Diagrams

Voronoi diagrams have a long history in many areas of science and engineering. Okabe's books give a comprehensive survey on this subject [90,91]. This section defines three common types of Voronoi diagrams and focuses mostly on the aw-Voronoi diagram because it is the most important concept for this thesis.

A Voronoi diagram of a set of sites $S = \{s_0, \ldots, s_{n-1}\}$ is the set of Voronoi regions $VD(S) = \{VR_0, \ldots, VR_{n-1}\}$, $VR_i = \{x \mid x \in \mathbb{R}^d \wedge \forall s_j \in S \;\; d(x,s_i) \leq d(x,s_j)\}$, where $d(x,s_i)$ is a function that measures the distance of points from the sites. The type of a Voronoi diagram depends on the definition of sites and the function d:

- If $d(x,s_i)$ is the Euclidean distance $d_E(x,s_i) = ||x - position(s_i)||$, then $VD(S)$ is the ordinary *Euclidean Voronoi diagram of points* [112].
- If all $s_i \in S$ have weights $w(s_i) \in \mathbb{R}$ and $d(x,s_i)$ is the power distance $d_p(x,s_i) = ||x - position(s_i)||^2 - w(s_i)$, then $VD(S)$ is the *power diagram* [3,43].
- If all $s_i \in S$ have weights $w(s_i) \in \mathbb{R}$ and $d(x,s_i)$ is the aw-distance (additively weighted) $d_{aw}(x,s_i) = ||x - position(s_i)|| - w(s_i)$, then $VD(S)$ is the *aw-Voronoi diagram*. The aw-distance of $x \in \mathbb{R}^3$ from the whole set S is $d_{aw}(x) = min_{s \in S}(d_{aw}(x,s))$.

Sites can be interpreted as spheres. In the case of ordinary Voronoi diagrams, a site s_i represents a sphere with the center $c_i = position(s_i)$ and a constant radius. In the case of power diagrams, a site s_i can be interpreted as a sphere with the center $c_i = position(s_i)$ and the radius $r_i = \sqrt{w(s_i)}$. In the case of aw-Voronoi diagrams, the radius of the sphere is exactly $r_i = w(s_i)$. Since aw-Voronoi diagrams do not change if a constant is added to the weights of all sites, we will implicitly assume that all sites have a positive weight and interpret them as spheres. These



(a) Ordinary Voronoi (b) Power diagram (c) Additively weighted

Figure 3.1: Different types of Voronoi diagrams

cases are illustrated in Figure 3.1. In this interpretation and with a suitable data structure, Voronoi diagrams provide information about spatial relations among the spheres.

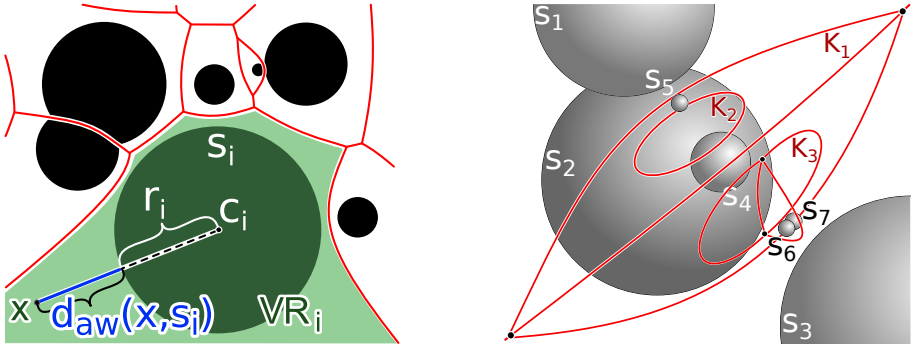
The boundary of Voronoi regions consist of lower-dimensional Voronoi *elements*. For any $k \in \{0,\dots,d\}$, a k-dimensional Voronoi element x is a part of the intersection of some $d+1-k$ Voronoi regions. In 3D, these elements are called 2-dimensional *faces*, 1-dimensional *edges* and 0-dimensional *vertices*. A Voronoi *skeleton* is the union of all vertices and edges.

In the case of ordinary Voronoi diagrams and power diagrams, Voronoi elements are linear, but in the case of aw-Voronoi digrams, these elements are quadrics [34, 113]. In particular, an aw-Voronoi face is a connected subset of a hyperboloid of two sheets, an aw-Voronoi edge is a connected subset of a conic curve, and an aw-Voronoi vertex is a point.

An aw-Voronoi skeleton can have several *components* (maximal connected non-empty sets of Voronoi edges and vertices), some of which can be simple ellipses, so called *loop edges*.

For a Voronoi element x, the set $S(x)$ will denote the $d+1-k$ sites of the Voronoi regions that constitute x. For any set X of Voronoi elements, $S(X) = \bigcup_{x\in X} S(x)$. If $X = K_i$ is a skeleton component, then $S(K_i)$ is the set of all sites that contribute to the elements of K_i.

Figure 3.2 illustrates the terms regarding aw-Voronoi diagrams in 2D and 3D.



(a) 2D analogy of a Voronoi region VR_i assigned to a site s_i

(b) Voronoi skeleton components K_1, K_2 and K_3. K_2 is a loop edge, $S(K_2) = \{s_1, s_4, s_5\}$ and $S(K_3) = \{s_3, s_4, s_6, s_7\}$

Figure 3.2: Aw-Voronoi diagrams in 2D and 3D

We will assume that the topology of Voronoi vertices and edges is stored in a suitable data structure, which also provides $S(K)$ for each component K, $S(e)$ for each edge e, $S(v)$ for each

vertex v, and also the coordinates of v.

An aw-Voronoi region $VR_i \in VD(S)$ is empty if and only if the sphere of the corresponding site s_i is fully contained in the interior of another $s_j \in S \setminus \{s_i\}$. The worst-case combinatorial complexity of a single region as well as the whole diagram in 3D is $\Theta(|S|^2)$ [3, 10, 113].

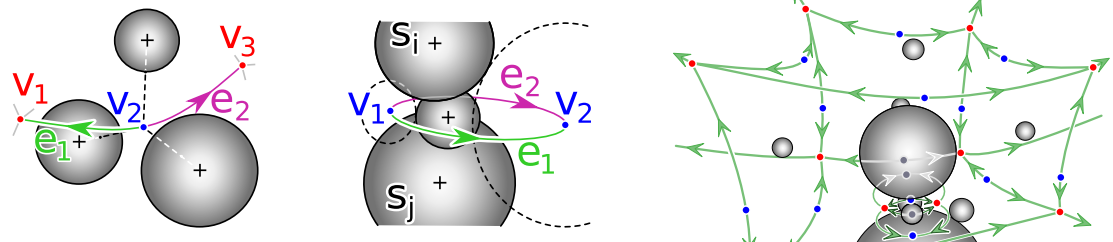
Certain configurations of sites might produce degenerate Voronoi elements. This happens if a point x has the same aw-distance from at least five sites, or the centers of at least four sites lie on the same plane, or at least four site-spheres are tangent to the same plane [34]. We will also consider degenerate the case when two site-spheres just touch and one of them contains the other, and the case when the curves of Voronoi edges just touch in a single point. All these cases are very specific and disappear if the degeneracy-causing sites are moved in random directions by an infinitesimally small distance. In this thesis, we will implicitly assume that sites in S do not cause any degeneracy in $VD(S)$.

Some terms are better explained via convex hulls. $CH(T)$ will denote the *convex hull of a set of spheres* $T \subseteq S$, i.e., the common intersection of all convex sets containing all spheres from the set T. We will say that a sphere $s_i \in T$ contributes to the $CH(T)$ if the surface of s_i touches the boundary of $CH(T)$. We will say that spheres of T are in a *convex position* if each $s_i \in T$ contributes to the $CH(T)$.

The shape of a Voronoi edge e depends on the configuration of the spheres in $S(e)$ [113]. Spheres in a convex position make the shape of a line or a hyperbola, e.g., $\{s_1, s_2, s_5\}$ in Figure 3.2(b). In a non-convex position, i.e., one of the spheres is fully contained in the interior of the convex hull of the others, the shape is a circle or an ellipse, e.g., $\{s_1, s_4, s_5\}$, $\{s_3, s_4, s_6\}$ and $\{s_3, s_4, s_7\}$ in Figure 3.2(b).

$VD(S)$ contains unbounded Voronoi regions with unbounded faces and edges. These regions are exactly the regions of the sites that contribute by their spheres to $CH(S)$ [34]. Such cases can be represented by additional *Voronoi elements at infinity* corresponding one-to-one with parts of the boundary of the $CH(S)$.

Sometimes it is useful to break the Voronoi skeleton into monotone pieces w.r.t. the aw-distance. The *enriched Voronoi diagram skeleton* [77, 105] is made of Voronoi vertices, extreme points, and the orientation of edges in the direction of increasing aw-distance. It is just a directed graph $\vec{G}$. See Figure 3.3 for an example.



(a) Hyperbolic Voronoi edge: Voronoi vertices v_1, v_3, extreme vertex v_2 (minimum).

(b) Elliptic Voronoi edge: Extreme vertices v_1 (minimum) and v_2 (maximum).

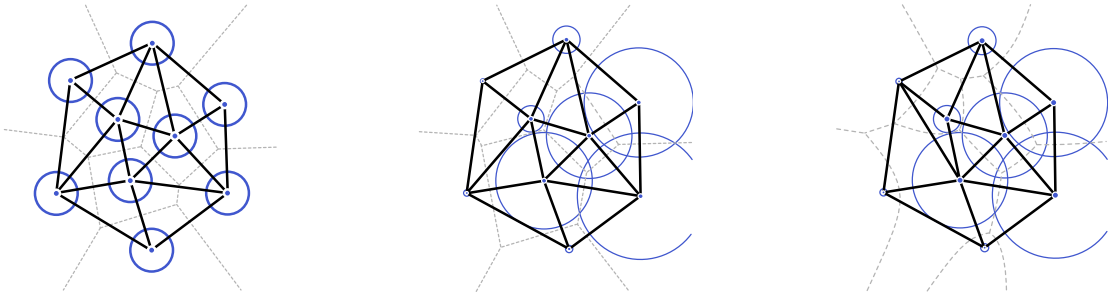
(c) A more complex example

Figure 3.3: Graph vertices correspond to Voronoi vertices and extreme points on Voronoi edges. Voronoi edges are split in extreme points and oriented in the direction of increasing aw-distance.

3.2 Duality

There is a duality between Voronoi diagrams and triangulations.

Each k-dimensional element x of a d-dimensional Voronoi diagram $VD(S)$ can be mapped to the convex hull of the $(d+1-k)$ points with positions from the set $S(x)$ of sites, i.e., to a $(d-k)$-simplex. In 2D, Voronoi vertices are mapped to triangles, Voronoi edges to straight line segments, and Voronoi regions to points. In 3D, vertices are mapped to tetrahedra, edges to triangles, faces to straight line segments and regions to points. The image of this mapping is called a d-dimensional dual triangulation. The dual triangulation of the ordinary Voronoi diagram is known as the *Delaunay triangulation* [21], the dual of the power diagram is the *regular triangulation*, and the dual of the aw-Voronoi diagram is known as the *quasi-triangulation* [60]. They are depicted in Figure 3.4.



(a) Delaunay triangulation is dual to Voronoi diagram

(b) Regular triangulation is dual to power diagram

(c) Quasi-triangulation is dual to aw-Voronoi diagram

Figure 3.4: Dual triangulations

Both Delaunay and regular triangulations satisfy the definition of a simplicial complex, i.e., a set of simplices, which contains all sub-simplices and the intersection of any two simplices is either $\emptyset$ or a shared sub-simplex. The mapping is one-to-one. This offers a very convenient way of storing the topology of the d-dimensional Voronoi diagram as a set of d-simplices. All remaining sub-simplices and coordinates can be easily computed.

Gavrilova [34, Chapter 3.2] defined the dual structure of an aw-Voronoi diagram and noticed some anomalies, but her definition did not cover all cases in 3D. Also Alinchenko et al. noticed many anomalies [1]. Kim et al. explored the dual structure into detail [58, 60], named it a quasi-triangulation and provided a comprehensive description of its anomalies [47, 57].

A quasi-triangulation is not necessarily a simplicial complex and the mapping can be many-to-one, e.g., in 3D, two Voronoi vertices can map to one tetrahedron, two or more Voronoi edges can map to the same triangle, and two or more Voronoi faces can map to the same straight line segment. Furthermore, the intersection of two simplices can contain two or more sub-simplices, and there can be lower-dimensional simplices that are not a sub-simplex of any higher-dimensional simplex, e.g., triangles that were mapped from loop edges. These cases are called *anomalies* [47, 60].

Anomalies might cause problems if a quasi-triangulation is used as an underlying data structure for storing the skeleton of the diagram. The ambiguity of mapping two Voronoi vertices to one tetrahedron can be handled by storing the coordinates of Voronoi vertices with tetrahedra and distinguishing tetrahedra by the coordinates, or alternatively by introducing an orientation to tetraheda. The ambiguity of mapping more Voronoi edges to one triangle can be handled by

explicitly storing the links between tetrahedra sharing a triangle.

3.3 Alpha/Beta Complexes and Shapes

Two concepts derived from dual triangulations are described in this section. They are useful for a spatial analysis of overlapping spheres. The concept of alpha shapes and complexes was originally introduced by Edelsbrunner et al. [29, 31, 32], and the concept of beta shapes and complexes by Kim et al. [59, 62]. It is possible to define alpha/beta shapes and complexes in several ways. The definition via Voronoi diagrams [30] is used in this thesis.

3.3.1 Alpha Complexes and Alpha Shapes

Let us consider a parameter $\alpha \in \mathbb{R}$ and a set $S = \{s_0, \ldots, s_{n-1}\}$ of sites. Each s_i, given by its $position(s_i) \in \mathbb{R}^d$ and $weight(s_i) \in \mathbb{R}$, represents a closed ball $B_i = \{x | x \in \mathbb{R}^d \wedge (x - position(s_i))^2 \leq weight(s_i) + \alpha\}$. Let $B'_i = B_i \cap VR_i$ be the intersection of the i-th ball with its Voronoi region in the power diagram of S. Note that the collection of all B'_i is a decomposition of the union of the balls. Each $T \subseteq S$ satisfying $\bigcap_{s_i \in T} B'_i \neq \emptyset$ is a simplex that belongs to a set called the *weighted alpha complex* of the set S. A *weighted alpha shape* is the boundary of the alpha complex.

A weighted alpha complex is a subset of the regular triangulation of S. As the parameter α grows, more and more simplices of the triangulation are included in the complex. However, the uniform growth of α does not imply the uniform growth of the balls, it holds only for ordinary (unweighted) alpha complexes. Only the subset for $\alpha = 0$ is important for the analysis of the union of the balls. An example of a weighted alpha complex in 2D is depicted in Figure 3.5. It consists of vertices, edges and triangular faces (red color). The weighted alpha shape is formed by just the vertices and edges except one edge, which is shared by two triangular faces.

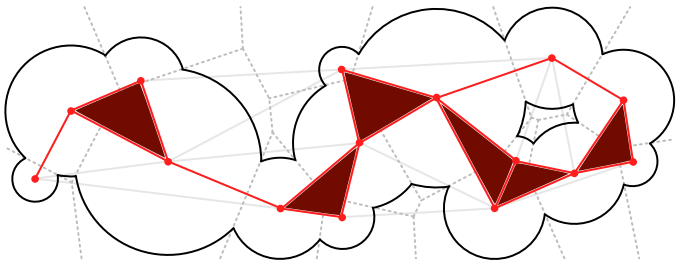


Figure 3.5: An example of a weighted alpha complex in $\mathbb{R}^2$.

3.3.2 Beta Complexes and Beta Shapes

Let us consider a parameter $\beta \in \mathbb{R}$ and a set $S = \{s_0, \ldots, s_{n-1}\}$ of sites. Each s_i, given by its $position(s_i) \in \mathbb{R}^d$ and $weight(s_i) \in \mathbb{R}$, represents a closed ball $B_i = \{x | x \in \mathbb{R}^d \wedge ||x - position(s_i)|| \leq weight(s_i) + \beta\}$. Let $B'_i = B_i \cap VR_i$ be the intersection of the i-th ball with its aw-Voronoi region. Each $T \subseteq S$ satisfying $\bigcap_{s_i \in T} B'_i \neq \emptyset$ is a simplex that belongs to a set called the *beta complex* of the set S. A *beta shape* is the boundary of the beta complex. Unlike alpha complexes, a beta complex is not necessarily a simplicial complex.

A beta complex is a subset of the quasi-triangulation of S. As β grows, more simplices are included in the complex as illustrated in Figure 3.6. The uniform growth of β implies the uniform growth of the balls (discs in 2D). Figure 3.6 shows the situation for a system of five sites in 2D and six values of β. The discs corresponding to the sites are gray and the union of these discs expanded by β is visualized by the red outline. The aw-Voronoi diagram is blue. The vertices, edges and triangular faces of the corresponding beta complex are shown in green and light red colors. Note that the beta complex in the last image (the largest β) is the whole quasi-triangulation. Beta shapes in Figure 3.6 are formed just by vertices and edges, except the two edges between triangular faces in the last image.

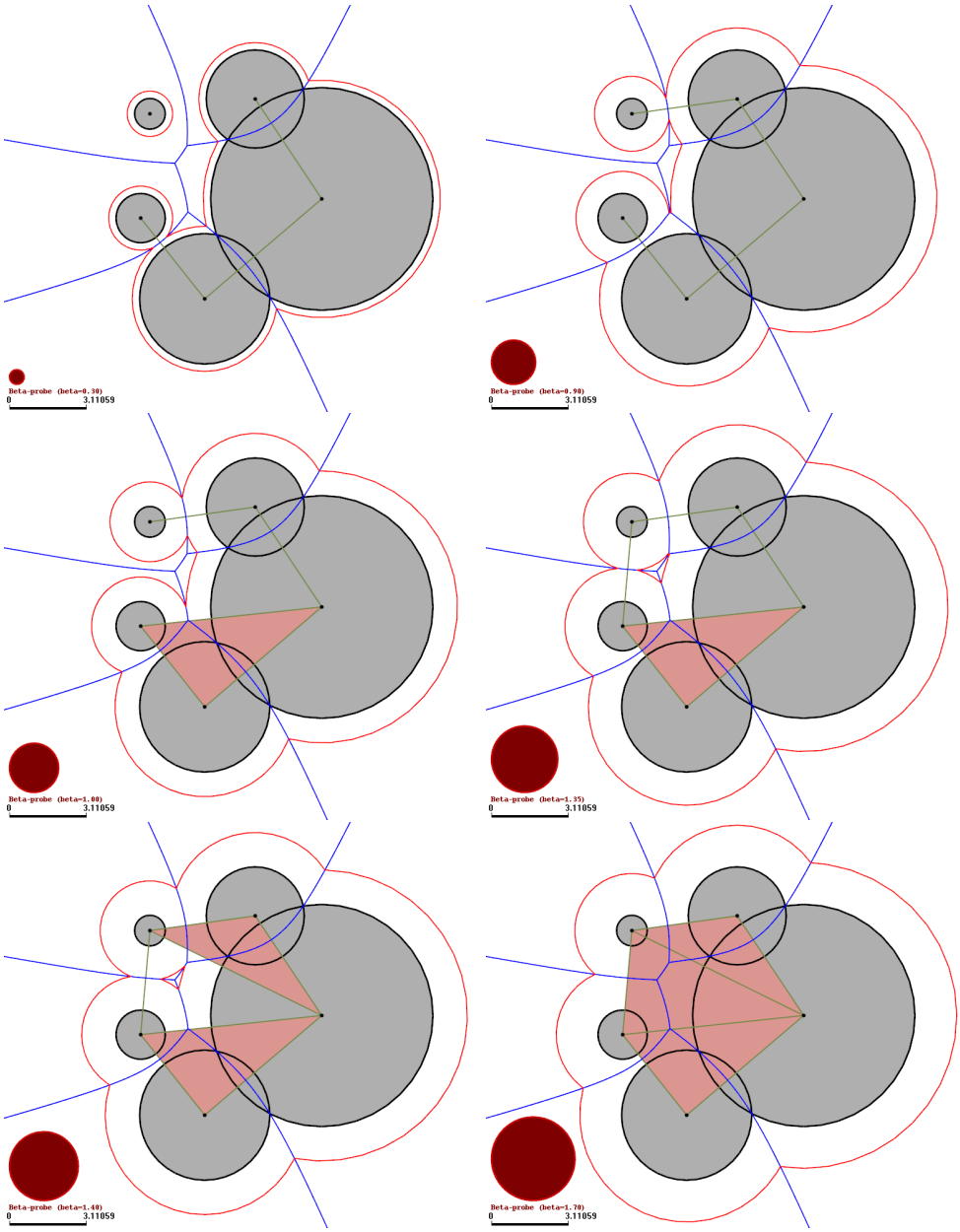


Figure 3.6: Beta complexes for increasing values of β (the radius of the red sphere). Computed by BetaConcept [65].

Chapter 4

Spatial Analysis of Proteins

The analysis of proteins may include the computation of the packing density of atoms, computations of the volume and surface area of the space occupied by atom spheres in the van der Waals model, the analysis of the space accessible by a small molecule that could react with the protein, etc. Voronoi diagrams have been used for such tasks since 1959, when Bernal suggested to study the properties of liquids in terms of the packing of irregular polyhedra [7]. A comprehensive survey can be found in the book of Okabe et al. [91] or in [38, 95].

4.1 Packing Density

Bernal and Finney introduced a standard Voronoi procedure for the analysis of the density in a random packing of spheres [8]. The packing density of a sphere is the ratio of the volume of the sphere in the corresponding Voronoi region to the volume of the region as illustrated in Figure 4.1. Richards applied the standard Voronoi procedure to determine the atomic volume and packing density of proteins [98]. However, the standard Voronoi procedure ignores radii of atoms. It would be better if larger atoms got more space than smaller atoms. Therefore, Richards introduced another "method B" which shifts boundaries of Voronoi regions to balance the volumes assigned to individual atoms. Unfortunately, this method has an unpleasant effect that some space can be left unassigned. To overcome this issue, Gellatly and Finey proposed a radical plane method [37], which is based on power diagrams. Under certain circumstances, atom centers in a power diagram can be located outside their regions. This is against a reasonable expectation that a locus of points near the atom center will be assigned to the atom. Gerstein, Tsai and Levitt proposed another tessellation, a multiplicatively weighted Voronoi diagram, to overcome this problem [39]. Regions of this tessellation no longer have linear boundaries but spherical. Goede, Preissner and Frömmel compared the previous approaches and suggested to use the aw-Voronoi diagram [40].

4.2 Volume

A power diagram built for a set of balls provides enough information about their intersections. Therefore, it can be used to compute the volume (and also the surface area) of their union via a decomposition into Voronoi regions [4, 12, 86, 111]. Power regions are convex polyhedra, so

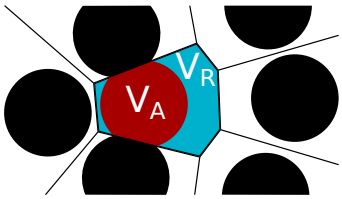


Figure 4.1: The packing density of the highlighted sphere is V_A/V_R, where V_A is the volume of the sphere inside its Voronoi region and V_R is the volume of the region.

the intersection of a region and a ball simplifies to the intersection of a simplex (tetrahedron) and a ball, which further reduces the problem to the intersection of a ball and at most 3 half-spaces [4]. Another method, proposed by Edelsbrunner and Fu [26, 28], computes the volume via an inclusion-exclusion principle as the sum of the volumes of individual balls, minus the volumes of the intersections of pairs, plus the volumes of the intersections of triplets, minus the volumes of the intersections of quadruplets. A regular triangulation provides these pairs, triplets, and quadruplets, via a weighted alpha complex [24, 25]. It is faster than processing all combinations. Also the beta complex (based on a quasi-triangulation, the dual structure of the aw-Voronoi diagram) can be used for the calculation of the volume. A decomposition scheme based on the beta complex has been proposed by Kim et al. [49].

4.3 Connolly Surface

Several methods for the computation of the Connolly surface are known. In general, such a method finds the analytic description of surface elements, solves their intersections and creates a model for visualization. Some methods keep the computation close to the surface, other are based on the exploration of volume.

4.3.1 Methods Based on Surface Exploration

These methods require a list of neighbors for each atom. Two atoms are *neighbors* if the distance between their centers is less or equal to the sum of their radii plus the probe diameter. These methods are fast for small probes and do not require much preprocessing, but their efficiency drops rapidly with increasing probe size.

Connolly's Algorithm

Neighboring atom pairs are candidates for parts of the surface where the probe creates toroidal patches and triplets of mutually neighboring atoms are candidates for spherical triangles. The algorithm processes all these pairs and triplets to get the corresponding parts of the surface [16]. Spherical triangles are created only in the positions, where the probe does not collide with other atoms. Three boundary arcs of each spherical triangle are accumulated at the corresponding tori and oriented. After that, arcs stored at each torus are sorted around the torus axis and grouped to create toroidal patches. The orientation of arcs help to distinguish real patches from gaps between patches. Boundary arcs in contact with atoms are created and accumulated at the corresponding atoms. After that, faces of spherical patches are constructed in two steps. First, arcs stored at

each atom are organized into loops. After that, loops are organized into groups. Each group corresponds to the boundary of a spherical patch. This grouping requires some non-trivial testing among all loop pairs of an atom.

Varshney's Algorithm

This algorithm, developed by Varshney et al. [109], constructs a feasible region for each atom. Feasible regions approximate Voronoi regions in a power diagram of expanded atom spheres (by the probe radius), but they are defined only on neighboring atoms. This restriction allows to compute feasible regions in parallel. The surface of the expanded sphere of each atom is then intersected with the volume of its corresponding feasible region. Vertices of this intersection correspond to spherical triangles, edges (circular arcs) correspond to toroidal patches, and faces correspond to spherical patches. Surface patches are then triangulated for visualization.

Sanner's Reduced Surface Algorithm

A reduced surface (RS), proposed by Sanner [104], is a simplification of the Connolly surface, where each spherical triangle is reduced to an ordinary triangle defined on the centers of the atoms touched by the probe (RS-face), each toroidal patch is reduced to an ordinary line segment (RS-edge), and each spherical patch is reduced to the center of the corresponding atom (RS-vertex). In fact, Sanner's reduced surfaces are equivalent to Edelsbrunner's alpha shapes.

The algorithm has four phases. First, the RS is computed. After that, an analytical representation of the Connolly surface is extracted from the parts of the RS. Next, intersecting parts of the Connolly surface are removed. At last, the Connolly surface is triangulated. The RS is computed by rolling the probe over atoms, starting from some initial position, where the probe defines an RS-face. The initial position can be computed, e.g., starting from the leftmost atom along the x-axis. RS-faces are incident to RS-edges. The probe is rotated around an RS-edge until it hits the sphere of some atom and hence the neighboring RS-face is discovered. Its RS-edges are scheduled for processing and the process is repeated until a closed surface is discovered. However, the whole RS can consist of more such parts and it can also contain free RS-edges and RS-vertices, which are not part of any RS-face/edge. Handling these cases is not trivial and it might seriously hurt the performance. A binary spatial division tree or a uniform grid data structure speeds up the retrieval of atom spheres in the neighborhood of any point [68, 104].

To improve both the rendering speed and visual quality, Krone et al. [68] combined Sanner's algorithm with GPU ray casting of Connolly surface elements. They also developed a GPU-parallel variant of the algorithm [69].

Contour-Buildup

This algorithm, developed by Totrov and Abagyan [108], finds the arcs of all spherical patches and then constructs remaining surface elements from this information. These arcs are found in the surface of the union of atoms with radii increased by the probe radius. A list of arcs at each exposed atom is updated as the neighborhood of the atom is searched for other atoms.

The contour-buildup algorithm was parallelized for CPU by Lindow et al. [78] and for GPU by Krone et al. [70]. The CPU variant runs in parallel over atoms whereas the GPU over arcs. Both variants use GPU ray casting for visualization.

4.3.2 Methods Based on Volume Exploration via aw-Voronoi Diagrams

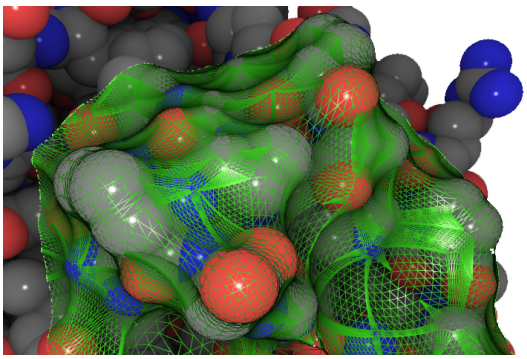
These methods provide more information, which is also useful for the detection and analysis of cavities. However, since they were developed mainly for molecular data, they do not solve problems that may occur in general input data. Interactivity of computation is another weak spot of these methods because these methods have complicated preprocessing and use triangle meshes for visualization.

The aw-Voronoi diagram helps to navigate probes of any size among atoms and detect collisions of the probe with atoms. Ryu et al. developed a method [101], where a probe is navigated through the network of Voronoi edges to find surface elements. Starting from the unbounded Voronoi edges that lead to infinity, the probe center runs through the network on collision-free paths. The edges, on which the probe first collides with atoms, are then used to get the triplets of atoms that generate spherical triangles. The ordinary Voronoi diagram of probe centers is used to accelerate finding intersections among spherical triangles, but it must be recomputed whenever the probe radius is changed. They have been using a NURBS model for visualization [101] but changed it for a triangular mesh later [100].

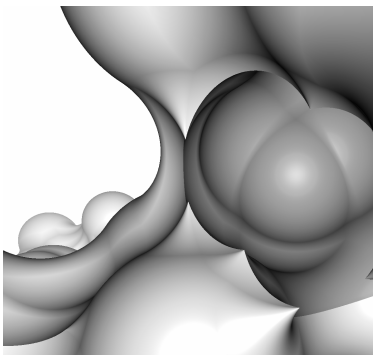
The previous approach evolved to the concept of beta-shapes based on a quasi-triangulation [58–60, 102]. Voronoi edges and vertices are represented by simplices and sorted by their *beta intervals*. These intervals determine the state of a simplex w.r.t. the beta complex. Details regarding beta intervals are well explained in [59] and they will be not discussed in this thesis. This sorting order allows to extract simplices for the construction of surface elements via orthogonal range search queries [61] in time $O(log(n)+k)$, where k is the number of extracted and n the number of all simplices. Unfortunately, a triangular mesh is used to represent the Connolly surface [100, 103]. The number of triangles is often too high to allow an interactive analysis.

4.3.3 Visualization of Connolly Surfaces

Early approaches have been using either dots or concentric curved segments for visual representation of Connolly surfaces [16, 17]. Traditional techniques use triangle meshes and current state-of-the-art techniques utilize GPU ray casting. These two are illustrated in Figure 4.2.



(a) Triangle mesh - many triangles, poor quality, slow rendering



(b) GPU ray casting - no triangles, high quality, fast rendering

Figure 4.2: Two different representations of the Connolly surface for rendering. Rendered by CAVER Analyst [67], using author's own plugin.

Triangle Meshes

Connolly developed a method for the triangulation of Connolly surfaces via a subdivision of the curved faces [18].

Sanner introduced another method [104]. Sanner's method first triangulates toroidal patches. The mesh parts corresponding to spherical triangles and patches are constructed using pre-triangulated sphere templates. The template is simply aligned with an atom and the triangles that do not belong to the Connolly surface are removed. Gaps are filled with new triangles, lying on the convex hull computed locally on the affected vertices of the atom sphere.

Ryu, Cho and Kim proposed another method [100], which extracts Connolly surface elements from the beta-shape of the molecule. These elements need to be triangulated. A subdivision scheme is used for spherical triangles and toroidal patches, and pre-triangulated templates for spherical patches. They also discussed self-intersections and proposed a solution, which ensures a watertight model. Their method was developed to support multiple levels of detail.

Triangle meshes have some drawbacks. Too many triangles are needed to achieve a high quality of visualization. Ryu et al. observed that the number of triangles in the mesh seems to be linear with respect to the number of atoms touched by the probe. However, for thousands of such atoms, the mesh can have millions (or even tens of millions) of triangles [100]. A high number of triangles implies high memory demands, low rendering performance, and long delays when the probe size is changed and the surface has to be recomputed. Nevertheless, a triangle mesh can be used for other things than visualization, e.g., to estimate the surface area of the Connolly surface or the volume of the space bounded by the Connolly surface. It is easier to solve these tasks on a triangle mesh.

GPU Ray Casting

Current state-of-the-art approaches [68–70, 78] use GPU ray casting to achieve both high quality and fast rendering. Almost all these approaches work in the following way. Each element of the Connolly surface is represented by a simple screen-aligned rendering primitive, e.g., a point sprite [68] or a tight-fitting rectangle [78]. During rendering, the analytical description of the element is passed to GPU via vertex attributes or a texture, and the primitive is moved to a position, where it fully occludes the corresponding Connolly surface element. GPU executes a fragment shader on each fragment generated by the rasterizer. The fragment shader computes the equation of the ray passing through the camera position and the fragment position. Next, the shader computes intersection points of the ray with the Connolly surface element. After that, the depth of the fragment is changed to lie on the Connolly surface or the fragment is discarded if the ray missed the element.

It is easy to compute the intersection of a ray and a sphere. See Figure 4.3. It leads to finding the roots of a quadratic polynomial. Computing the parameters of the point sprite for a sphere is not so trivial, but it has been well described, e.g., by Sigg et al. [106]. A great advantage of using point sprites is that many spheres can be rendered via a single OpenGL draw-call, using only one vertex per sphere.

The intersection of a ray and a torus is not so easy to compute because it leads to finding the roots of a quartic polynomial. Analytical solutions are known, but they are complicated and numerically unstable in 32-bit floating-point arithmetic. The intersection can also be computed numerically. Toledo and Levy [20] compared a few methods: Sturm, double derivative bisection,

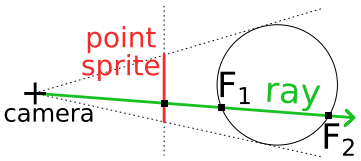


Figure 4.3: Ray casting a sphere - 2D analogy. F_1 and F_2 are the intersection points of the ray and the sphere. The ray was casted from the camera location and passes through a fragment generated on a point sprite.

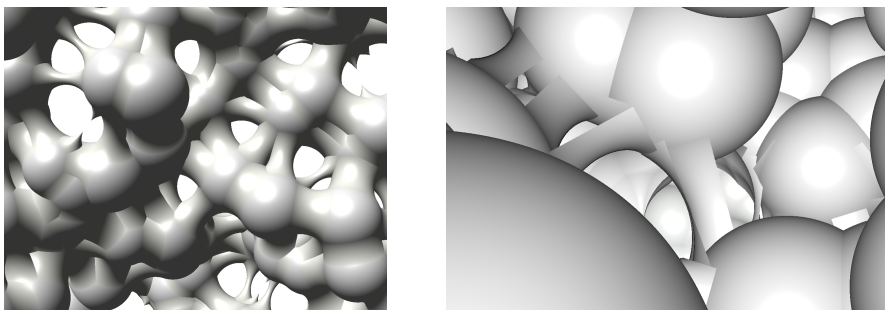
sphere tracing and Newton-Rhapson. They state that iterative methods are faster than analytical solutions because they compute only one root. Nevertheless, for the purpose of rendering Connolly surfaces, all intersections may be needed in some cases.

In the context of Connolly surfaces, Lindow et al. use a numerical sphere-tracing for the ray-torus intersection and tight-fitting rectangles [78], but this approach finds only the first intersection. A sphere-tracing is an iterative method. For a given point on the ray, the distance of the point from the torus is computed. This distance is the radius of a tangent sphere, so the point can safely travel this distance along the ray without hitting the torus. The point is moved by this distance along the ray, the distance is recomputed and this process is repeated until the distance or the number of iterations fall below some threshold.

Krone et al. use an analytic approach (stabilized Ferrari equations) and point sprites [70], but our experiments indicate a numerical instability for probes ≥ 10Å.

To achieve high rendering performance, both methods avoid clipping of spherical and toroidal patches. The whole spheres and full 360° toroidal patches are rendered instead. If the surface is opaque and viewed from the outside, then the rest is automatically handled by the depth buffer. Views behind the surface or clipping parts of the surface are impossible without seeing the parts that should not be seen. Figure 4.4 illustrates the problem.

This problem was solved later by Kauker et al. in the context of rendering transparent Connolly surfaces [45]. Their method is based on per-pixel linked lists. First, the whole scene is rendered into a shader storage buffer. The depth buffer is disabled during rendering. After that, fragments at the same screen position are sorted by depth, the scene is composed back-to-front, and boolean operations are performed to get rid of the parts that should not be seen.



(a) Krone's approach [70] (b) Lindow's approach [78]

Figure 4.4: Rendering performance is better if toroidal and spherical patches are not clipped, but the opposite side of the Connolly surface is incorrect.

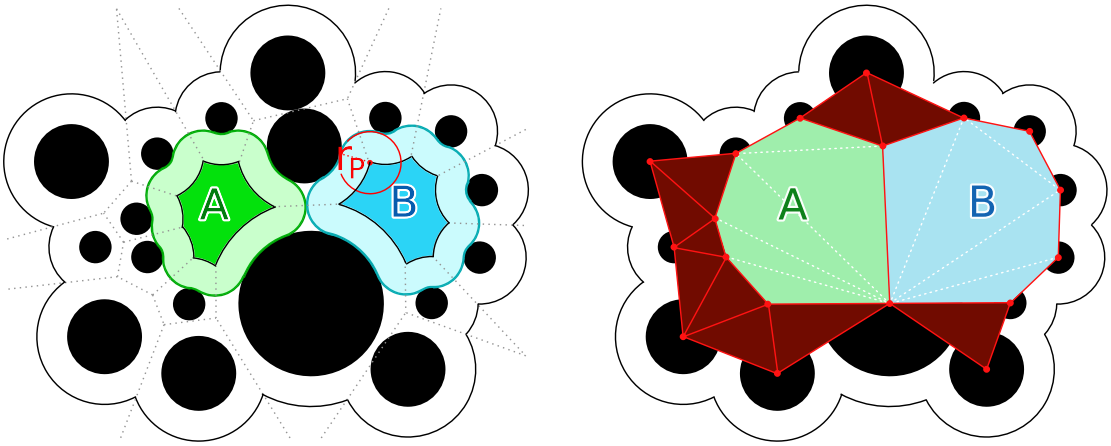
Parulek and Viola [94] described the Connolly surface as the isosurface of an implicit function. They are able to change the probe radius (up to 2Å) and render the visible part of the surface with GPU ray casting.

4.4 Detection of Cavities

Power diagrams and aw-Voronoi diagrams provide enough information for the detection of cavities in molecules. This section summarizes the approaches based on these diagrams and derived structures (dual triangulations, alpha/beta complexes).

4.4.1 Methods Based on Alpha Complexes

Edelsbrunner et al. presented how to use the concept of weighted alpha complexes for the detection of cavities [27, 73, 74]. An alpha complex describes overlaps among atom spheres expanded by the probe radius, but cavities are in the complementary space. A weighted alpha complex is a subset of a regular triangulation, remaining simplices of the triangulation, which are not in the alpha complex, describe the complementary space. These simplices are then grouped via shared triangles. Groups can be computed with the help of a union-find system [19, 33] and correspond one-to-one with cavities. This approaches is illustrated in Figure 4.5.



(a) Cavities A and B as connected components of the accessible space. Also their Connolly surfaces are shown. Dotted lines indicate the power diagram of the atom spheres enlarged by r_p.

(b) Cavities A and B as connected components of the complement of the alpha-complex in the regular triangulation ($\alpha = 0$). The alpha complex is dark/light red.

Figure 4.5: Different representations of cavities. The probe radius is r_P. Atom spheres are black and the outline indicates the accessible surface.

Since its introduction, the analysis based on weighted alpha complexes became very popular. It is the fundamental concept behind some online tools, e.g., CASTp [23]. It has also been used by Liang and Dill for the analysis of the packing of proteins, liquids and crystalline solids [72]. One of the presented characteristics was the number of cavities detected by probes of different sizes. A probe radius interval was sampled and the number of cavities was computed for each sample.

An interesting problem is the analysis of a system of balls with respect to probe radius variations. This has been solved for uniform balls. In this case, the underlying structure is the ordinary Voronoi diagram of ball centers and its dual structure is a simplicial complex. All simplices in the complex can be sorted into a sequence, where each prefix represents an alpha complex. The sequence of successive prefixes is called the family of alpha complexes and it is an equivalent description of the system of growing balls with equal radii. All topological features of this system can be detected by the incremental algorithm of Delfinado and Edelsbrunner [22]. The algorithm can be conceptually divided into three parts. In the first part, the sorted sequence of simplices is processed from the beginning to the end, building components of 0- and 1-dimensional simplices (points and line segments). In the second part, the processing order is reversed and components of 3-dimensional simplices (tetrahedra) are built. Two 3-dimensional simplices are connected via a 2-dimensional simplex. Topological changes are detected via changes in the number of components and via detection of cycles. The simplices on which a topological change occurred are marked. At last, the sequence of simplices is processed once more to count topological features according to the marked simplices. The union-find system [33] is used to manage connected components of simplices.

An interesting extension of [22] is the work of Edelsbrunner, Letscher and Zomorodian [30] regarding topological persistence and simplification. As the balls grow, topological features may appear and disappear, so the lifetime of a topological feature can be measured. This is used to distinguish important long-living features from the short-living ones. We should mention here that the method was designed for weighted alpha complexes, but if the balls have different radii, their growth is non-uniform.

There are also some disadvantages. If the probe radius is changed, then the whole regular triangulation must be recomputed. Furthermore, the triangulation does not provide enough information to solve certain tasks, e.g., finding the greatest filling sphere of a cavity.

4.4.2 Methods Based on Aw-Voronoi Diagrams and Beta Complexes

Luchnikov et al. used the general Voronoi S-network to analyze the distribution of empty space in liquid crystals [80] and to find cavities among spheres in a cylindrical container [79]. A Voronoi S-network is the skeleton of a general Voronoi diagram, where the Euclidean distance of points from the boundaries of general objects is used. If the objects are spheres, then the Voronoi S-network is equivalent to the aw-Voronoi diagram, although some additional definitions may be needed to handle the cases of overlapping spheres. The general Voronoi S-network was computed numerically [80], the analytical solution for spheres was developed later by Medvedev et al. [87]. Alinchenko et al. used the Voronoi S-network to analyze the morphology of cavities (voids) in a simulated hydrated lipid bilayer membrane [1]. They extract clusters of Voronoi edges and vertices along which the probe can move freely, avoiding collisions with atoms, and use them as skeletons of cavities. Besides these skeletons, cavities are approximated by sphero-cylinders to gain more information, e.g., their orientation and sphericity of cavities. The analysis was performed for probes of different sizes and revealed that the membrane is impenetrable even for the smallest molecules.

Since Kim et al. developed the edge tracing algorithm for the construction of aw-Voronoi diagrams [51–53], they use it to solve many problems in bio-chemistry. Kim and Sugihara developed a method for the detection of cavities and tunnels [56, 63]. The method trims the whole aw-Voronoi diagram by atom spheres and the remaining parts are used to describe topologi-

cal properties of voids and tunnels. However, this is not very efficient if the whole diagram is trimmed each time a probe size is changed. This approach was improved later to use the concept of beta shapes [66]. The simplicial boundary of all voids is efficiently extracted from a quasi-triangulation. The boundary is then classified into shells and simplices filling the shells are found. If the triangles of cavity shells are extracted by an advanced approach [61], then this method should be more efficient than the previous one. However, problematic cases corresponding to elliptic Voronoi edges and disconnected surfaces still remain to be addressed. Kim et al. also developed a method for the extraction of pockets on the surface of proteins via a quasi-triangulation [46]. Since Kim et al. developed the edge tracing algorithm for the construction of aw-Voronoi diagrams [51–53], they use it to solve many problems in bio-chemistry. Kim and Sugihara developed a method for the detection of cavities and tunnels [56, 63]. The method trims the whole aw-Voronoi diagram by atom spheres and the remaining parts are used to describe topological properties of voids and tunnels. However, this is not very efficient if the whole diagram is trimmed each time a probe size is changed. This approach was improved later to use the concept of beta shapes [66]. The simplicial boundary of all voids is efficiently extracted from a quasi-triangulation. The boundary is then classified into shells and simplices filling the shells are found. If the triangles of cavity shells are extracted by an advanced approach [61], then this method should be more efficient than the previous one. However, problematic cases corresponding to elliptic Voronoi edges and disconnected surfaces still remain to be addressed. Kim et al. also developed a method for the extraction of pockets on the surface of proteins via a quasi-triangulation [46].

Lindow et al. introduced the extraction of molecular paths from the aw-Voronoi diagram [77]. First, the Voronoi diagram is clipped by an approximation of the convex hull of the molecule. This is done by evaluating ambient occlusion in vertices and removing the ones where the occlusion exceeds some threshold. Next, the diagram is converted to a directed graph by splitting Voronoi edges in their critical points and passed through a cascade of filters to get a tiny subset representing significant molecular paths. The cascade removes vertices in the proximity of atoms, branches without critical points, cycles, redundant branches, etc.

Lindow's approach can be extended to trace the development of cavities in dynamic molecules [75, 76]. The cascade of filters is replaced with a single filter that removes the parts of the graph where the distance from atoms drops below the probe radius. Filling spheres of cavities are generated on Voronoi edges and the skin surface is computed on these spheres for visualization purposes.

Chapter 5

Construction of Voronoi Skeleton

An algorithm for the construction of Voronoi skeleton components is described in this chapter. The description goes into details because it is required by some contributions of this thesis. After a short historical overview, the basic idea of the algorithm will be explained and more details will be given in the following sections. The last section of this chapter is dedicated to an acceleration technique, which makes this algorithm practical for larger input sets.

Luchnikov et al. presented a numerical algorithm for the computation of a general Voronoi diagram with Euclidean distance from convex obstacles [80]. Edges and vertices of this diagram are called a Voronoi S-network. The algorithm uses a flexible spherical probe, keeps the probe in touch with three or more obstacles and moves it step by step among the input spheres. The center of the probe traces the S-network. The algorithm is general but too slow. It is based on an older idea of Tanemura, Ogawa and Ogita for ordinary Voronoi diagrams [107]. Medvedev et al. improved the S-network algorithm with analytical computation of Voronoi vertices for spherical obstacles (additively weighted Voronoi diagram) [87]. Kim et al. introduced a similar algorithm for analytical computation of the additively weighted Voronoi diagram via tracing Voronoi edges and named it the *edge tracing algorithm* [51,52,54]. Spatial filters and parallelism may be necessary to make the edge tracing algorithm applicable to larger models [13,77,84,93]. The algorithm described in this chapter is based on both the work of Kim et al. [52] and the work of Medvedev et al. [87].

5.1 Edge Tracing Algorithm

For a given set S of sites and any $s_i \in S$, Algorithm 1 discovers such a component K of the aw-Voronoi skeleton that satisfies $s_i \in S(K)$. First, an initial Voronoi vertex is discovered (this is discussed in Section 5.4) and added to a collection L, e.g., a stack or a queue. After that, the rest of K is found in the following way. Vertices are removed from L one after another. When a vertex v_i is removed from L, each one of its four slots for neighbors is processed as follows. Occupied slots are skipped. If a slot is empty, the end vertex v_j must be found (this is discussed in Section 5.5) and interconnected with v_i via a Voronoi edge (the corresponding slot at v_i is filled with a reference to v_j and vice versa). Newly found vertices are added to L for later processing. Some kind of a map is necessary to manage already found vertices, e.g., a hash-map or a tree-map. It is possible to use Voronoi vertex coordinates as a key in the map. Also the quadruple of sites on which the vertex was computed can be used, but this quadruple is not unique and some additional information is required to resolve ambiguity.

Algorithm 1: EdgeTracing(s_i, S)

Input: The set of sites S and an arbitrary site $s_i \in S$
Output: A component K (edges and vertices) of $VD(S)$ with $s_i \in S(K)$

```
1  el ← FindInitialVoronoiElement(s_i, S); // el: a vertex or a loop edge
2  if el is a loop edge then return {el} ;
3  L ← {el};  // L: a list of vertices for which to trace incident edges
4  K ← {el};
   // neighbors[.,.]: 4 neighbors for each Voronoi vertex. Initially empty.
5  while L ≠ ∅ do
6      v_i ← any member of L; L ← L \ {v_i}; // Remove any vertex v_i from L.
7      for n_i ← 0; n_i < 4; n_i ← n_i + 1 do   // n_i: the slot in v_i for a neighbor
8          if neighbors[v_i, n_i] == ∅ then
9              v_j ← FindEndVertex(v_i, n_i, S); // v_j: the neighbor of v_i via n_i
10             if v_j ∉ K then L ← L ∪ {v_j} ; // v_j has not been computed before.
11             else Get the data of previously computed v_j ∈ K ;
               // Create an edge v_i ↔ v_j:
12             n_j ← the slot in v_j for the neighbor v_i;
13             neighbors[v_i, n_i] ← v_j;
14             neighbors[v_j, n_j] ← v_i;
15             K ← K ∪ {v_j, edge(v_i, n_i, v_j, n_j)}; // Update the component K.
16         end
17     end
18 end
19 return K;
```

Unfortunately, the processing of all $s_i \in S$ by Algorithm 1 does not imply the discovery of the whole aw-Voronoi skeleton. This issue will be solved later by the contribution in Chapter 7.

5.2 Computation of Vertex Coordinates

The computation of aw-Voronoi vertex coordinates was described by Gavrilova [34, 35]. Given four sites $s_0, \ldots, s_3$, the unknown position $c \in \mathbb{R}^3$ of a Voronoi vertex candidate must satisfy (5.1) for some unknown $r \in \mathbb{R}$. It is nothing more than finding a sphere inscribed to four spheres $s_0, \ldots, s_3$, a special case of the 10^{th} problem of Apollonius in three dimensions.

$$d_{aw}(c, s_i) = ||c - c_i|| - r_i = r \;\; \forall i \in \{0, \ldots, 3\} \tag{5.1}$$

This can be squared to get (5.2).

$$(c - c_i)^2 = (r + r_i)^2 \text{ and } r + r_i \geq 0 \;\; \forall i \in \{0, \ldots, 3\} \tag{5.2}$$

Now we can choose any $k \in \{0, \ldots, 3\}$, subtract (5.3) from the rest to get rid of c^2 and r^2 and obtain (5.4) for each $i \in \{0, \ldots, 3\} \setminus \{k\}$.

$$(c - c_k)^2 = (r + r_k)^2 \tag{5.3}$$

$$c^2 - 2cc_i + c_i^2 - (c^2 - 2cc_k + c_k^2) = r^2 + 2rr_i + r_i^2 - (r^2 + 2rr_k + r_k^2) \tag{5.4}$$

Equation (5.4) simplifies to (5.5).

$$c(c_i - c_k) + r(r_i - r_k) = (c_i^2 - c_k^2 - r_i^2 + r_k^2)/2 \tag{5.5}$$

Equation (5.5) can be rewritten to (5.6).

$$c(c_i - c_k) + r(r_i - r_k) = ((c_i - c_k)^2 - (r_i - r_k)^2)/2 + c_k(c_i - c_k) - r_k(r_i - r_k) \tag{5.6}$$

A substitution $c'_i = c_i - c_k$, $r'_i = r_i - r_k$, $c' = c - c_k$, $r' = r + r_k$ then gives the following system of three linear equations and one quadratic equation in four variables:

$$c'c'_i + r'r'_i = (c'^2_i - r'^2_i)/2 \;\; \forall i \in \{0,\ldots,3\} \setminus \{k\} \tag{5.7}$$

$$c'^2 - r'^2 = 0 \text{ and } r' \geq -min(r'_i) \tag{5.8}$$

If k is chosen to satisfy $r_k = min(r_i)$, then the condition simplifies to $r' \geq 0$. The linear system can be solved w.r.t. a free variable $f' \in \{x', y', z', r'\}$, for which a solution exists, using Cramer's rule or Gaussian elimination, and the solution substitued into the quadratic equation. The quadratic equation is then solved for f' and the result is used to recover the values of remaining variables. At last, the inverse substitution is performed to get the final coordinates. Since a quadratic polynomial can have up to two real roots, there can be up to two Voronoi vertex candidates.

Precise computation has been addressed by Gavrilova [36], Anton, Mioc and Santos [2], and Nishida and Sugihara [89]. Olechnovič and Venclovas compute coordinates in a double precision floating-point type [93]. To reduce numerical errors, they use Kahan's summation algorithm [44] and a numerically safer algorithm for finding the roots of a quadratic polynomial [96].

5.3 Extreme Points

For a given set T of sites, the set $EP(T) = \{x \mid x \in \mathbb{R}^3 \wedge \forall s_i, s_j \in T \; d_{aw}(x, s_i) = d_{aw}(x, s_j)\}$ is the set of equidistant points of T, i.e., each $x \in EP(T)$ has the same aw-distance from all sites in T. $EP(T)$ contains a point of minimum aw-distance, $minEquidistantPoint(T)$. Its geometrical interpretation is the center of the minimal sphere inscribed to the spheres of T. If $EP(T)$ is an ellipse, it will also have a point of maximum aw-distance, $maxEquidistantPoint(T)$. These extreme points are useful for diagram construction and subsequent spatial analysis.

The computation of coordinates is trivial if $|T| \leq 2$. The case $|T| = 4$ was discussed in Section 5.2. The case $|T| = 3$ corresponds to finding extreme points on the geometrical domain of Voronoi edges. The set $EP(T)$ is given by three equations (5.1) for $T = \{s_0, s_1, s_2\}$ and an additional constraint that r must be minimal or maximal. This can be solved in at least two ways which are described below.

The first and easy way is based on an observation that extreme points must lie in the plane passing through the centers c_0, c_1 and c_2 of the site spheres. Cutting the spheres by the plane yields three circles. A fourth circle inscribed to these circles is computed. Its center is the extreme point. This approach requires a transformation to a 2D sub-space of the 3D space generated by the x-, y- and z-axis, solving two linear equations in three variables and one quadratic equation in a chosen free variable, and finally a transformation back to 3D.

The second way[1] has a more convenient transformation to 2D. The whole $EP(T)$ is simply embedded in 4D space generated by the x-, y-, z- and r-axis. It leads to a system of two linear

[1] Proposed by the author of this thesis.

equations in four variables (5.7) and one quadratic equation (5.8). To solve the linear system, two free variables must be chosen. One of them is r and the other is any $w \in \{x, y, z\}$ provided that the remaining 2x2 system is regular. Plugging the solution into (5.8) gives an implicit function (a quadric) in the 2D space generated by the w- and r-axis, which can be analyzed for extrema in the r-axis. The derivative of the implicit function can be found by the implicit function theorem or implicit differentiation.

It is useful to know the minima and maxima of the aw-distance on Voronoi edges. The minimum will be called the *bottleneck* of the edge. These points are among vertices or extreme points. Angular comparison [52] can be used to decide whether an extreme point lies on the edge as illustrated in Figure 5.1. In this thesis, the term bottleneck may refer to the point of minimal aw-distance on a Voronoi edges as well as to the value of the aw-distance in this point.

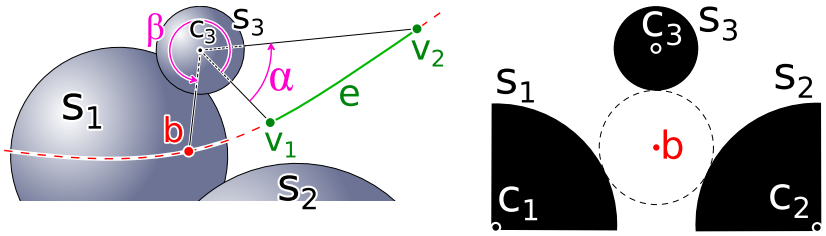


Figure 5.1: The point b is a false candidate to the bottleneck. The aw-distance $d_{aw}(s_3, b)$ is minimal, but $b \notin e$ since $\beta > \alpha$. The true bottleneck of e is v_1.

5.4 Finding Initial Voronoi Element

This section describes how to find some initial Voronoi element. The technique was originally proposed by Medvedev et al. [87] ("BLOCK1" in their algorithm) but since it works only for finite Voronoi vertices, we will extend the technique by adding a few tests to detect also elements at infinity and loop edges. To simplify the explanation, we will exclude empty Voronoi regions by an assumption that spheres of sites in S may intersect but not fully contain each other.

Algorithm 2 finds an initial Voronoi element as follows. For a given $s_i \in S$, it chooses the $s_j \in S \setminus \{s_i\}$ with minimal sphere inscribed between s_i and s_j, then it selects the $s_k \in S \setminus \{s_i, s_j\}$ with minimal sphere inscribed to s_i, s_j and s_k, and finally $s_l \in S \setminus \{s_i, s_j, s_k\}$ with minimal inscribed sphere. The configuration $\{s_i, s_j, s_k, s_l\}$ creates the initial Voronoi vertex. This strategy just selects a Voronoi face $VR_i \cap VR_j$, then an edge $VR_i \cap VR_j \cap VR_k$ and finally a vertex $VR_i \cap VR_j \cap VR_k \cap VR_l$. In addition to the original [87], Algorithm 2 handles special cases: If the number of element-defining sites is 1 (line 12), then the whole diagram has just one region. If the number of sites is 2 (line 13), then the diagram has two regions and the result is a loop edge at infinity. If the number of sites is 3 (line 14), then a decision is made by computing supporting planes for the triplet of spheres. If supporting planes exist (line 15), then the result is a vertex at infinity, otherwise it is a loop edge (line 16). At last, only if the number of sites is 4 (line 17), then the result is an ordinary Voronoi vertex. The time complexity of this technique is $O(|S|)$.

Algorithm 2: FindInitialVoronoiElement(s_i, S)

Input: The set of sites S and some $s_i \in S$
Output: A Voronoi element $el \in VD(S)$ satisfying $s_i \in S(el)$

$T \leftarrow \{s_i\}$; $p \leftarrow \emptyset$; // T: element-defining sites, p: vertex position.
for $\forall m \in (j,k,l)$. **do** // Three steps: j, k and l. s_m is a candidate site,
  $s_m \leftarrow \emptyset$; $r_m \leftarrow \infty$; // r_m: the radius of a sphere inscribed to $T \cup \{s_m\}$.
  foreach $s'_m \in S \setminus T$ **do**
    $p' \leftarrow minEquidistantPoint(T \cup \{s'_m\})$;
    if p' *exists* $\wedge\ d_{aw}(p', s'_m) < r_m$ **then**
      $r_m \leftarrow d_{aw}(p', s'_m)$; $s_m \leftarrow s'_m$; $p \leftarrow p'$;
    end
  end
  if $s_m \neq \emptyset$ **then** $T \leftarrow T \cup \{s_m\}$;
end
// Additionally to [87], detect elements at infinity and loop edges:
if $|T| == 1$ **then return** $\emptyset$;
if $|T| == 2$ **then return** $T \cup \{s_\infty\}$; // A loop edge at infinity.
if $|T| == 3$ **then** // A vertex at infinity or a loop edge.
  if T *has supporting planes* **then return** $T \cup \{s_\infty\}$ and a normal vector ;
  else return T ; // A loop edge.
else return T and the vertex position p ; // An ordinary Voronoi vertex.

5.5 Finding Other Voronoi Vertices

Each Voronoi vertex v_s is given by its position and a quadruple of sites $S(v_s)$. Four Voronoi edges are incident to v_s. The geometry of an edge e incident to v_s is given by a triplet $S(e) \subset S(v_s)$. The edge e leads to another vertex v_e, possibly not yet discovered, defined on a quadruple $S(v_e) = S(e) \cup \{s_m\}$ for some $s_m \in S \setminus S(e)$. The site s_m can be found by computing all Voronoi vertex candidates for all members of $S \setminus S(e)$, and choosing the candidate closest to v_s along e. Distances along e can be compared as the oriented angle[2] at the center of the smallest site of $S(e)$ from v_s to a candidate point $x \in EP(S(e))$. This technique is formalized in Algorithm 3 and has the time complexity $O(|S|)$.

5.6 Tracing Edges Faster

Time-complexity issues will arise if the edge tracing algorithm is combined with the ordinary end-vertex search strategy (Algorithm 1 and 3). For $|S| = n$ input sites, the ordinary strategy will take $O(n)$ time per edge, so the total time complexity will be $O(n^2)$ on usual data and $O(n^3)$ in the worst case. This can be improved with spatial filtering [13, 84, 93] and parallelism.

Cho et al. [13] introduced spatial filters to speed up the edge tracing algorithm. A filter is computed only once for the current edge being traced and only the sites with spheres hitting the filter are considered for the computation of Voronoi vertex candidates. A 2D analogy is depicted in Figure 5.2. The filter consists of a front feasible region R_F and a rear feasible region R_R,

[2]The comparison can be implemented without any evaluation of angles.

Algorithm 3: FindEndVertex($\{v_s, T\}$, n, S)

Input: The set of sites S, the position v_s of the start vertex together with its defining sites $T \subseteq S$, $|T| = 4$, and a number $n \in \{0, \ldots, 3\}$ to select a neighbor-slot in v_s.
Output: The end vertex for v_s in the direction of the edge given by the slot n

$s_r \leftarrow T[n]$; // A reference site from the slot n.
$U \leftarrow T \setminus \{s_r\}$; // Three remaining sites that define the edge.
$c_i \leftarrow$ the center of the site $\in U$ that has the minimal radius;
Initialize the orientation for angles $\measuredangle(v_s, c_i, .)$, from the knowledge of v_s, c_i, U and s_r;
$v_e \leftarrow \emptyset$; $s_m \leftarrow \emptyset$; // The initial end vertex candidate (position and site).
// Vertices at infinity are the first candidates to the end vertex:
$N \leftarrow$ normals of two supporting planes of U;
if $N \neq \emptyset$ **then**
    $s_m \leftarrow s_\infty$;
    // Choose among two vertices at infinity the one closer to v_s
    if $\measuredangle(v_s, c_i, c_i + N[0]) < \measuredangle(v_s, c_i, c_i + N[1])$ **then** $v_e \leftarrow c_i + N[0]$;
    else $v_e \leftarrow c_i + N[1]$;
end
// Remaining sites s'_m may yield closer candidates v'_e:
foreach $s'_m \in S \setminus U$ **do**
    foreach $v'_e \in EP(U \cup \{s'_m\}) \setminus \{v_s\}$ **do** // EP: Equidistant points; except v_s
        if $v_e == \emptyset \vee \measuredangle(v_s, c_i, v'_e) < \measuredangle(v_s, c_i, v_e)$ **then**
            $v_e \leftarrow v'_e$; $s_m \leftarrow s'_m$;
        end
    end
end
return $U \cup \{s_m\}$ and the vertex position v_e;

mutually separated by a plane Π. The ideal filter has a complicated shape, therefore, they enclose R_F into a sphere and use a uniform grid to find all sites hitting the enclosing sphere. The search may propagate over grid cells through R_R if necessary. A lazy strategy can be applied when the first candidate is discovered, but this idea was only marginaly mentioned in [13]. Also Medvedev et al. [87] use a spatial grid to find sites close to a point, but they did not explicitly describe any stop condition. Lindow et al. [77] use non-uniform spatial grid with a stop condition. They were also the first who reported a parallel version of the edge tracing algorithm. The usability of grid-based approaches higly depends on a proper choice of grid resolution and the uniformity of data. Olechnovic and Venclovas [93] improved the original filters of Cho et al. [13] by encapsulating R_F by a cylinder. Their method searches for sites hitting the filter lazily and may shrink the filter as the search progresses. In Figure 5.2 for instance if s_5 was found first, then R_F can be replaced by the sphere inscribed to s_1, s_2 and s_5, and hence s_3, s_5 and s_6 can be skipped. A hierarchy of bounding spheres built on all sites is used to find the sites hitting the filter approximation.

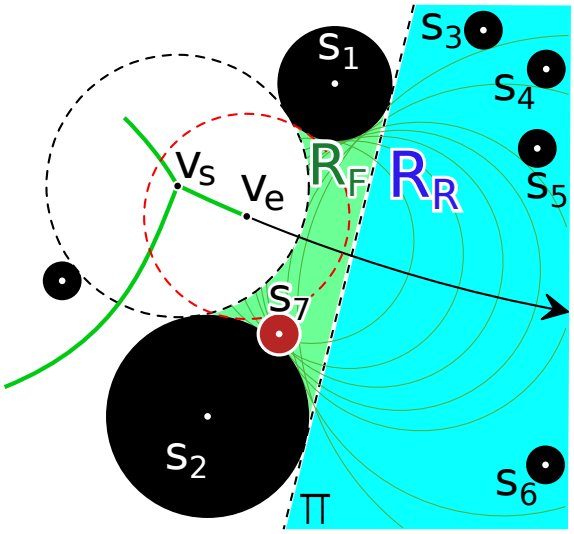


Figure 5.2: 2D analogy of a spatial filter $R_F \cup R_R$ for tracing an edge from a start vertex v_s. Voronoi vertex candidates are computed from s_1, s_2 and a site s_i hitting the filter. The end vertex v_e is computed for $s_i = s_7$.

Contributions

Chapter 6

Fast Discovery of Voronoi Vertices with Delaunay Triangulation

The expected time complexity of the basic variant of the edge tracing algorithm is at least quadratic w.r.t. the number of sites on the input. Running the algorithm on molecular models consisting of tens of thousands of atoms then takes thousands of seconds, which is not very practical. Cho et al. [13] introduced static spatial filters, which help to overcome this issue. However, their method uses a regular grid to search through the space delineated by the filters. If the number of grid cells is large, time complexity issues will arise.

A new acceleration method for the edge tracing algorithm is presented in this chapter [84]. The method uses filters very similar to the filters proposed by Cho et al. [13], which were discussed in Section 5.6. The proposed method uses Delaunay triangulation of the centers of sites to locate in a lazy way the sites hitting the filter. The filter and its approximation may shrink as new Voronoi vertex candidates are being discovered and hence less sites need to be processed.

This chapter is organized as follows. The first section describes a dynamic variant of Cho's static filters. Dynamic filters have complicated shapes. Therefore, their outer approximation is presented next. These approximations have simpler shapes (either a sphere or the union of two spheres), which can be searched through via Delaunay triangulation. The proposed method is described next. After that, the experiments and results of an implementation of the proposed method are presented and compared to another approach.

6.1 Feasible Region Relative to a Point

Let v_s be a Voronoi vertex and $S(v_s)$ the set of its corresponding sites. For an arbitrary reference site $s_r \in S(v_s)$, the set $S(v_s) \setminus \{s_r\}$ defines the geometrical domain of a Voronoi edge incident to v_s, which will be called a *trajectory*. Recall that the trajectory is a line, a hyperbola or an ellipse, with an orientation in the direction of a Voronoi edge incident to v_s. The vertex v_s is the start-vertex of the trajectory.

Let p be a point on the trajectory as illustrated in Figure 6.1. A *feasible region* relative to p is the space occupied by the union of all spheres externally tanget to $S(v_s) \setminus \{s_r\}$ with centers from v_s to p along the trajectory but without the space of the Voronoi sphere of v_s. Note that $\lim_{p\to\infty} R(p)$ gives exactly the same feasible region as originally proposed by Cho et al. [13], see Figure 5.2.

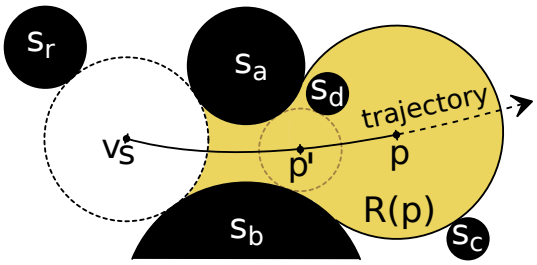


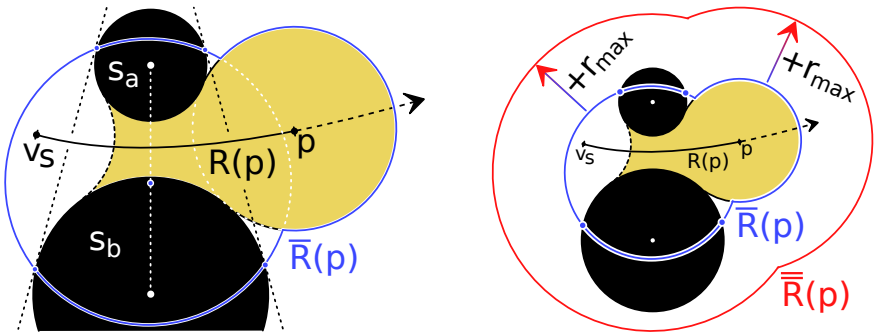
Figure 6.1: 2D analogy of a feasible region $R(p)$ relative to a point p. The trajectory starts at a Voronoi vertex v_s. $S(v_s) = \{s_a, s_b, s_r\}$ is the set of sites on which v_s is defined, s_r is a reference site, and $S(v_s) \setminus \{s_r\}$ defines the trajectory curve. The point p is an end-vertex candidate defined on $\{s_a, s_b, s_c\}$ but the real end-vertex is in p' defined on $\{s_a, s_b, s_d\}$ because no site sphere hits the interior of $R(p')$.

$R(p)$ is related to the end-vertex search. Suppose that p is a candidate to the end-vertex of the Voronoi edge. If the sphere of any site s_i hits the interior of $R(p)$, then the set $(S(v_s) \setminus \{s_r\}) \cup \{s_i\}$ generates an end-vertex candidate closer to v_s than p along the trajectory, otherwise p is the end-vertex of the Voronoi edge.

6.2 Outer Approximations of Feasible Regions

We need to be able to find site spheres intersecting $R(p)$, but the shape of $R(p)$ is quite complex. Therefore, we will enclose $R(p)$ into larger but simpler filters.

The first outer approximation $\overline{R}(p)$ of $R(p)$ is depicted in Figure 6.2(a). It is the union of a Voronoi sphere at p and another sphere enclosing the rest of $R(p)$. The second sphere is the same as the one suggested by Cho et al. [13] to enclose the front feasible region R_F. Its center lies in the plane passing through centers of sites in $S(v_s) \setminus \{s_r\}$ and its surface passes through the points where supporting planes touch $S(v_s) \setminus \{s_r\}$. The second approximation $\overline{\overline{R}}(p)$ of $R(p)$, depicted in Figure 6.2(b), is just $\overline{R}(p)$ enlarged by r_{max}, the maximal radius of all sites.



(a) The 1^{st} approximation $\overline{R}(p)$ of $R(p)$ (b) The 2^{nd} approximation $\overline{\overline{R}}(p)$ of $R(p)$

Figure 6.2: 2D analogy of the approximations of $R(p)$ for non-elliptic trajectories

Cho et al. [13] did not describe in detail how to compute the sphere for $\overline{R}(p)$. Let us show how it can be computed. Let us assume wlog that one of the spheres defining the trajectory is located in the origin and has zero radius. Let us denote remaining spheres as $s_1 = (c_1, r_1)$ and

$s_2 = (c_2, r_2)$. The normal vector n of a supporting plane (a plane touching both s_1 and s_2 from one side and passing through the origin) must satisfy (6.1).

$$nc_1 = -r_1,\ nc_2 = -r_2,\ n^2 = 1 \tag{6.1}$$

The unknown sphere $s = (c, r)$ passes through the origin and both points where the supporting plane touches s_1 and s_2, so c and r must satisfy (6.2), (6.3) and (6.4).

$$(c - (c_1 + nr_1))^2 = r^2 \tag{6.2}$$
$$(c - (c_2 + nr_2))^2 = r^2 \tag{6.3}$$
$$c^2 = r^2 \tag{6.4}$$

The center c also satisfies (6.5) for some scalar variables $u, v \in \mathbb{R}$ because it lies in the plane passing through c_1, c_2 and the origin.

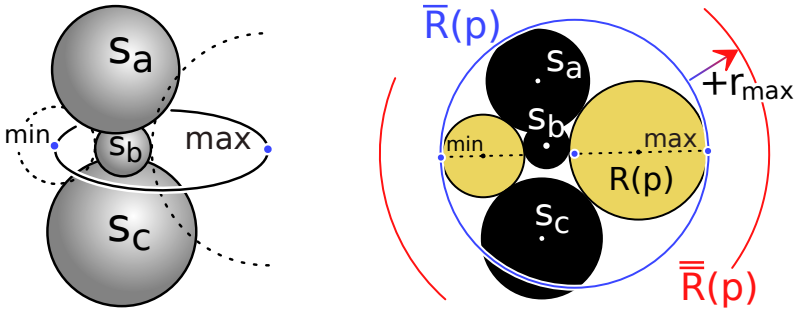
$$c = uc_1 + vc_2 \tag{6.5}$$

The equations above lead to (6.6) and (6.7), which can be solved for u and v and the solution substituted back to (6.5) and (6.4) to get c and r.

$$u(c_1c_1 - r_1r_1) + v(c_1c_2 - r_1r_2) = (c_1c_1 - r_1r_1)/2 \tag{6.6}$$
$$u(c_1c_2 - r_1r_2) + v(c_2c_2 - r_2r_2) = (c_2c_2 - r_2r_2)/2 \tag{6.7}$$

If the trajectory is an ellipse, then $R(p)$ is a subset of a Dupin cyclide. Then $\overline{R}(p)$ is the minimal bounding sphere of the whole Dupin cyclide and $\overline{\overline{R}}(p)$ is again $\overline{R}(p)$ enlarged by r_{max}. Figure 6.3 illustrates this on a simple example.

Approximations $R(p) \subset \overline{R}(p) \subseteq \overline{\overline{R}}(p)$ simplify testing whether the sphere of a site, e.g., s_d, intersects $R(d)$. The sphere of s_d can hit $R(p)$ only if the center of s_d lies in $\overline{\overline{R}}(p)$. Subsequently, the sphere of s_d can hit $R(p)$ only if the sphere of s_d hits $\overline{R}(p)$. If some of these conditions are violated, then s_d can be safely ignored, otherwise s_d still needs to be tested against $R(p)$ and $R(p)$ updated accordingly. This is accomplished by computing candidates to Voronoi vertices, i.e., spheres externally tangent to $(S(v_s) \setminus \{s_r\}) \cup \{s_d\}$. If some of the candidates, e.g., p', is closer to v_s than the current best candidate p but $p' \neq v_s$, then s_d hits $R(p)$ and the current best candidate p will be updated $p \leftarrow p'$.



(a) Minimum and maximum on an elliptic trajectory

(b) Approximations $\overline{R}(p)$ and $\overline{\overline{R}}(p)$

Figure 6.3: The approximations of $R(p)$ for elliptic trajectories

6.3 Delaunay Triangulation for Efficient Filtering

Finding sites captured by $\overline{\overline{R}}(p)$ is nothing else than finding which points (site centers) hit a sphere, the union of two spheres, or the union of a sphere and a half-space. This section describes how to solve it via Delaunay triangulation.

Lemma 1. *Let V be a set of points in $\mathbb{R}^d$ and $G = (V, E)$ be the graph of their Delaunay vertices and edges (0- and 1-sub-simplices of the Delaunay triangulation of V). For any open ball/half-space Q, the induced sub-graph $G[V \cap Q]$ is connected.*

Proof. Instead of a Delaunay triangulation, consider the dual Voronoi diagram of points. Finding a path between u and v in $G[V \cap Q]$ means finding a sequence $\{p_1, \ldots, p_n\} \subseteq V \cap Q$ satisfying $p_1 = u$, $p_n = v$, and $\forall\{p_i, p_{i+1}\}$ the corresponding Voronoi regions $VR(p_i)$ and $VR(p_{i+1})$ share a $(d-1)$-dimensional facet. Such a path can be constructed from points along two line segments: $L_{u,c}$ from u to c and $L_{c,v}$ from c to v, where c is the center of Q. The situation is depicted in Figure 6.4. Consider any point $x \in L_{u,c}$. Let $y \in \partial Q$ be the point of intersection of the boundary sphere ∂Q and the half-line from c to u. This y is the closest point of ∂Q from x. Apparently $||u - x|| < ||y - x||$ and $\forall z \in (V \setminus Q)\ ||y - x|| < ||z - x||$, hence $x \notin VR(z)$, therefore, $x \in VR(w)$ for some $w \in V \cap Q$. The second segment $L_{c,v}$ is done analogically. The sequence of sites that constitute the path is given by the Voronoi regions visited along $L_{u,c} \cup L_{c,v}$.

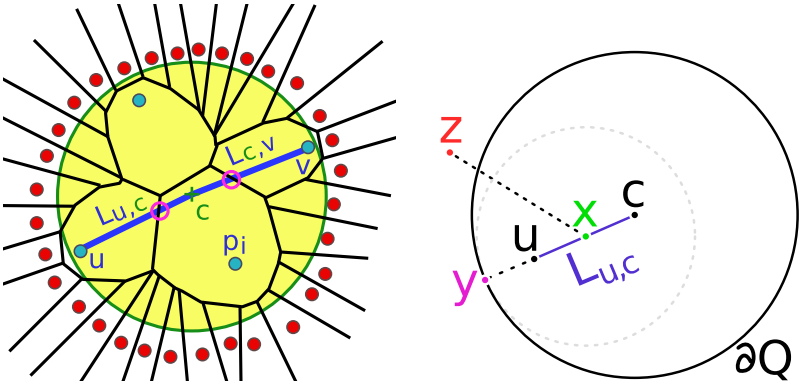


Figure 6.4: A path $u \to p_i \to v$ in a spherical filter Q

□

Corollary 1. *Let $G = (V, E)$ be given as in Lemma 1. For any open balls/half-spaces Q_1, Q_2, the graph $G[V \cap Q_1] \cup G[V \cap Q_2)]$ is connected if $V \cap Q_1 \cap Q_2 \neq \emptyset$.*

Since for each filter $\overline{\overline{R}}(p)$ the sites of $S(v_s) \setminus \{s_r\}$ are captured by both components of $\overline{\overline{R}}(p)$, Lemma 1 and Corollary 1 guarantee that remaining sites can be discovered by searching through the Delaunay vertices and edges that lie in $\overline{\overline{R}}(p)$. This is illustrated on a 2D example in Figure 6.5.

If Delaunay triangulation of site centers is available, then we can use spatial filtering for efficient searching of Voronoi vertices in the edge tracing algorithm. This approach is summarized in Algorithm 4, which replaces lines 12-19 in Algorithm 3, i.e., the part where a trajectory starting in a given Voronoi vertex v_s is traced to find the end-vertex of a Voronoi edge.

In Algorithm 4, Delaunay vertices and edges will be searched through in the space bounded by approximations of feasible regions. First, the analytical description of $\overline{R}(v_e)$ and $\overline{\overline{R}}(v_e)$ is

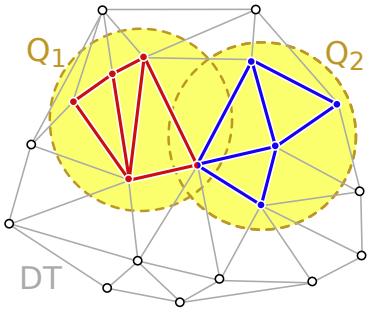


Figure 6.5: Delaunay triangulation is used to find points in open balls Q_1 and Q_2 via a graph search on induced sub-graphs

computed with respect to the initial end-vertex candidate v_e as discussed in Section 6.2. It is either a sphere, the union of two spheres, or the union of a sphere and a half-space. Next, immediate neighbors of trajectory-defining sites are added to a queue L. Each site s'_m extracted during the search is tested whether its center c'_m hits $\overline{\overline{R}}(v_e)$. If it does not, then s'_m cannot produce a better candidate than v_e and there is no need to schedule neighbors on this site. If c'_m hits $\overline{\overline{R}}(v_e)$ then unprocessed neighbors of s'_m must be added to L. Furthermore if the sphere of s'_m hits $\overline{R}(v_e)$, then up to two end-vertex candidates are computed from the quadruple $U \cup \{s'_m\}$. Each candidate v'_e is tested whether it is closer to v_s along the trajectory than the current best candidate v_e. Closer candidates are used to update v_e, s_m, $\overline{R}(v_e)$ and $\overline{\overline{R}}(v_e)$. The shrinking of $\overline{\overline{R}}(v_e)$ may reduce the number of sites added to L in upcoming iterations.

The time complexity of Algorithm 4 is dominated by $O(m+n)$ of the breadth-first search for m visited Delaunay edges and n visited Delaunay vertices. We often expect $m,n \in O(1)$, but the worst-case is worse: n can be the number of all sites if spatial filters are inefficient, and $m \in O(n^2)$ is the combinatorial complexity of 3D Delaunay triangulation.

The work of Algorithm 4 is illustrated in Figure 6.6. The initial end-vertex candidate is a vertex at infinity ($v_e = v_\infty$), so the initial filter $\overline{\overline{R}}(v_e)$ is the union of a sphere and a half-plane (a half-space in 3D) as illustrated in Figure 6.6(a). The Delaunay triangulation of site centers is depicted in Figure 6.6(b). Sites are labeled by the extraction order in a BFS queue. Neighbors 1-9 of trajectory-defining sites are scheduled for BFS. Extracted sites 1-4 are skipped because their centers do not hit $\overline{\overline{R}}(v_e)$. The center of the site 5 hits $\overline{\overline{R}}(v_e)$. Therefore, its unprocessed neighbors 10-12 are scheduled for BFS. Since the sphere of the site 5 does not hit $\overline{R}(v_e)$, this site cannot produce any better candidate. The site 6 is extracted and its unprocessed neighbors 13 and 14 are scheduled because the center of the site 6 hits $\overline{\overline{R}}(v_e)$. Since the sphere of the site 6 hits $\overline{R}(v_e)$, the site 6 is used to compute a new candidate v'_e. This v'_e is closer to v_s than the previous candidate v_∞, therefore, v_e is updated to v'_e, s_m is updated to the site 6 and filters are updated. The new situation with updated filters is depicted in Figure 6.6(c). The site 7 is extracted and skipped because its center does not hit $\overline{\overline{R}}(v_e)$. The situation at the site 8 is the same as it was at the site 5, so the site 15 is scheduled for BFS. The situation at the site 9 is the same as it was at the site 6, so a new candidate v''_e is computed. This v''_e is closer to v_s than the current best candidate v_e, so v_e is updated to v''_e, s_m is updated to the site 9, filters are recomputed and also $\overline{\overline{R}}(v_e)$ shrinks a bit. The new situation with updated filters is depicted in Figure 6.6(d). Note that the center of the site 14 no longer hits $\overline{\overline{R}}(v_e)$. At last, sites 10-15 are extracted, skipped, the queue becomes empty and BFS finishes. The v''_e generated by $s_m = 9$ is the Voronoi end-vertex.

Algorithm 4: TraceTrajectory(U, v_s, v_e, s_m, c_i). Filtered processing of remaining sites for finding the end-vertex along a Voronoi edge trajectory

Input: The set U of three sites defining the trajectory, the start-vertex v_s of the trajectory, the initial end-vertex candidate v_e and its site s_m, the center c_i of the smallest site $s_i \in U$ for angular distance comparison along the trajectory. The graph of Delaunay vertices and edges.

Output: The next Voronoi vertex on the trajectory (the end-vertex)

```
1  Initialize R̄(v_e) and R̿(v_e); // See Section 6.2
2  P ← ∅; // P: processed sites
3  for u ∈ U do P ← P ∪ Neighbors(u) ;
4  L ← P \ U; // L: BFS queue
5  while L ≠ ∅ do
6      s'_m ← the last member of L; L ← L \ {s'_m};
7      if c'_m ∈ R̿(v_e) then // s'_m hits the 2nd outer approx. of R(v_e)
8          L ← L ∪ Neighbors(s'_m) \ P;
9          P ← P ∪ Neighbors(s'_m);
10         if s'_m ∩ R̄(v_e) ≠ ∅ then // s'_m hits the 1st approx. of R(v_e)
11             for v'_e ∈ EP(U ∪ {s'_m}) \ {v_s} do // a new candidate v'_e
12                 if ∡(v_s, c_i, v'_e) < ∡(v_s, c_i, v_e) then // v'_e is closer to v_s
13                     v_e ← v'_e; s_m ← s'_m;
14                     Update R̄(v_e) and R̿(v_e);
15                 end
16             end
17         end
18     end
19 end
20 return U ∪ {s_m} and the vertex position v_e;
```

6.4 Experiments and Results

This section shows the running time of the proposed method on molecular models taken from the Protein Data Bank [6]. Each model provides the coordinates and types of atoms. The Bondi's set of atomic radii was used to map radii to atoms [11]. For our purposes, a molecular model is nothing else than a collection of spheres. In this kind of data, spheres may intersect but not fully contain each other, and all spheres have similar radii ranging from 1.2Å to 1.8Å.

Experiments were performed on a 64-bit Fedora Linux 22, running on Intel® Core$^{(TM)}$ i7-4930K CPU @ 3.40 GHz × 12, 64 GB RAM. This CPU offers 12 CPUs on its 6 cores.

The proposed method was implemented in Java. The implementation is freely available as an OpenSource computational library AwVoronoi [15]. The core of the method is the edge tracing algorithm [52, 87] and it supports the following strategies for finding end vertices on Voronoi edges.

BruteForce implements Algorithm 3, which uses almost all input sites for the computation of end-vertex candidates on each Voronoi edge

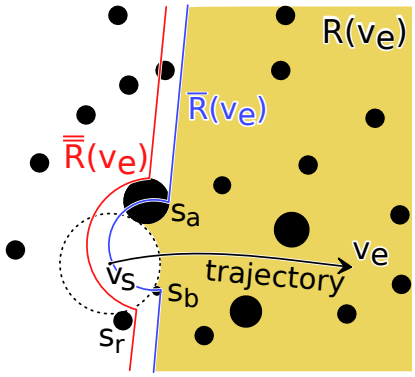
BruteForce + Filtering additionally, spatial filters are used to detect sites on which no valid candidate can be computed. This saves a huge amount of floating-point operations but it does not reduce the number of steps.

BruteForce + Filtering + Parallelism additionally, end-vertex candidates on each Voronoi edge are computed in 12 CPU threads in parallel. All threads work on the same edge.

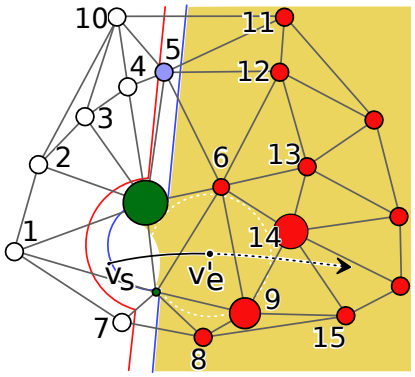
Delaunay + Filtering implements Algorithm 4, which uses the Delaunay triangulation of sphere centers to quickly search through spatial filters. This can reduce the number of steps because many sites that do not hit a filter can be skipped without testing.

Delaunay + Filtering + Parallelism additionally, up to 12 edges are traced in parallel. Threads are synchronized when the topology of the Voronoi skeleton is being updated and when the collection of edges scheduled for tracing is being updated. This strategy may trace some edges twice. This happens when an edge is being traced by one thread in one direction and, in the same time, by another thread in the opposite direction.

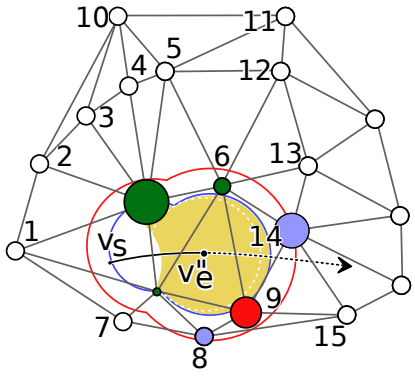
Figure 6.7 illustrates the dependency of the computation time on the input size for different



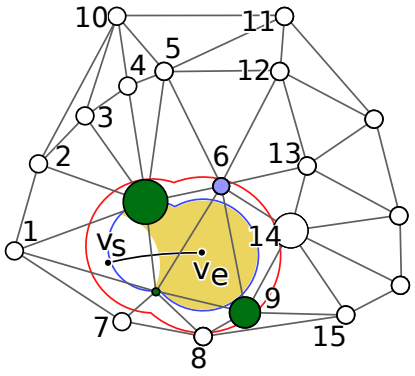
(a) Initial filters for a trajectory given by $U = \{s_a, s_b\}$ and starting at the Voronoi vertex v_s. The initial end-vertex candidate is $v_e = v_\infty$.



(b) Sites are labeled by the extraction order in a BFS queue. A new Voronoi vertex candidate v'_e is computed at the site 6.



(c) Filters were updated. A new Voronoi vertex candidate v''_e is computed at the site 9. The candidate v''_e is closer to v_s than v'_e.



(d) Filters were updated. The BFS finishes after extracting sites 10-15 and the candidate $v_e = v''_e$ becomes the Voronoi end-vertex.

Figure 6.6: Delaunay triangulation of site centers is used to find sites for the computation of candidates to a Voronoi vertex. 2D analogy.

strategies on thousands of molecular structures. The horizontal axis represents the input size in the number of atoms. Proteins with 90 000 atoms can be considered very large, but also larger structures exist, e.g., viruses or RNA. The vertical axis is the Voronoi diagram computation time in seconds. The computation time does not include the time spent on parsing the input data from a file or writing the output to a file, however, the preprocessing time of the Delaunay strategy, i.e., the computation of the Delaunay triangulation, is included. Figure 6.7 indicates the quadratic time complexity of the BruteForce strategy. This is not a surprise because the combinatorial complexity of a Voronoi diagram is expected to be $O(n)$ for a protein with n atoms, and the BruteForce strategy spends $O(n)$ time on each Voronoi edge. The utilization of paralellism and filtering reduces the running time by a multiplicative constant.

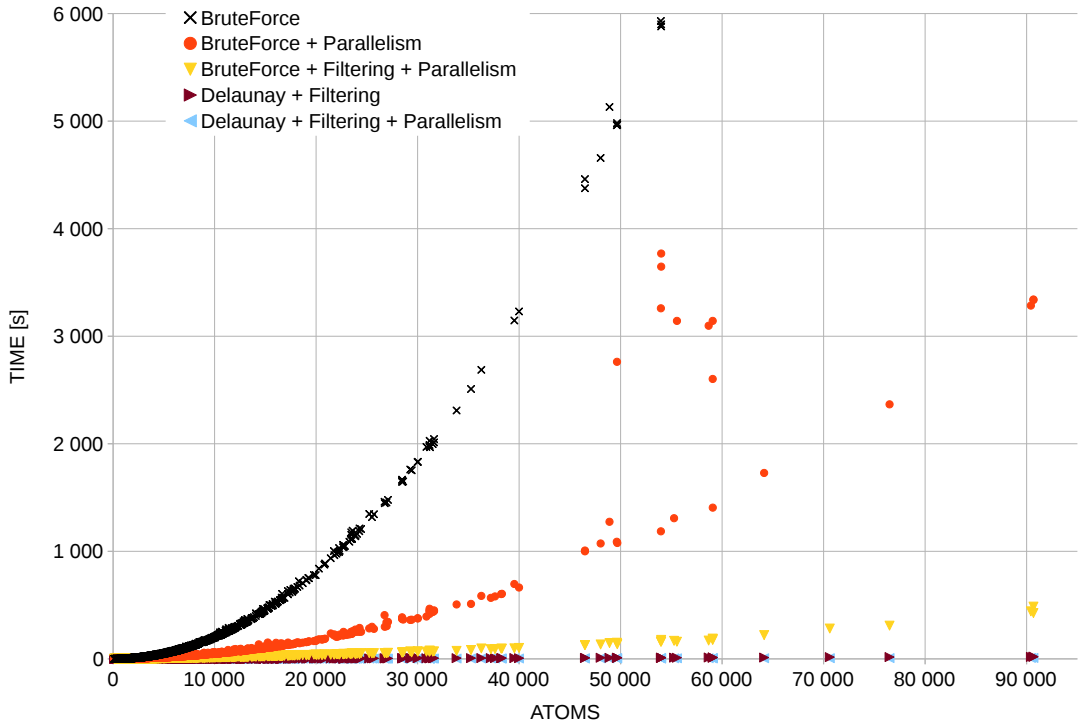


Figure 6.7: Voronoi diagram computation time for different strategies

Figure 6.8 focuses on Delaunay strategies. They seem to be always better than any BruteForce variant, but this highly depends on the efficiency of filters. The efficiency of filters will decrease rapidly for increasing difference between the radius of the smallest and the largest atom. If the difference was comparable with the size of the whole protein, then the filters would not help at all. Fortunatelly, such cases are impossible in protein models. Delaunay strategies require preprocessing, which is not negligible as indicated in Figure 6.9.

Figure 6.10 compares the performance of AwVoronoi [15] and QTFier [110]. AwVoronoi is written in Java. It reads PDB data from XML files, computes the Delaunay triangulation of atom centers, computes a quasi-triangulation via the edge tracing Algorithm 1 using the proposed optimization based on dynamic filters and Delaunay triangulation, and writes the resulting quasi-triangulation to an XML file. QTFier is written in C/C++. It reads plain text PDB files, computes a quasi-triangulation via the edge tracing algorithm [52,60] using an optimization based on static filters [13] and a uniform grid, and writes the resulting quasi-triangulation to a plain text file [55].

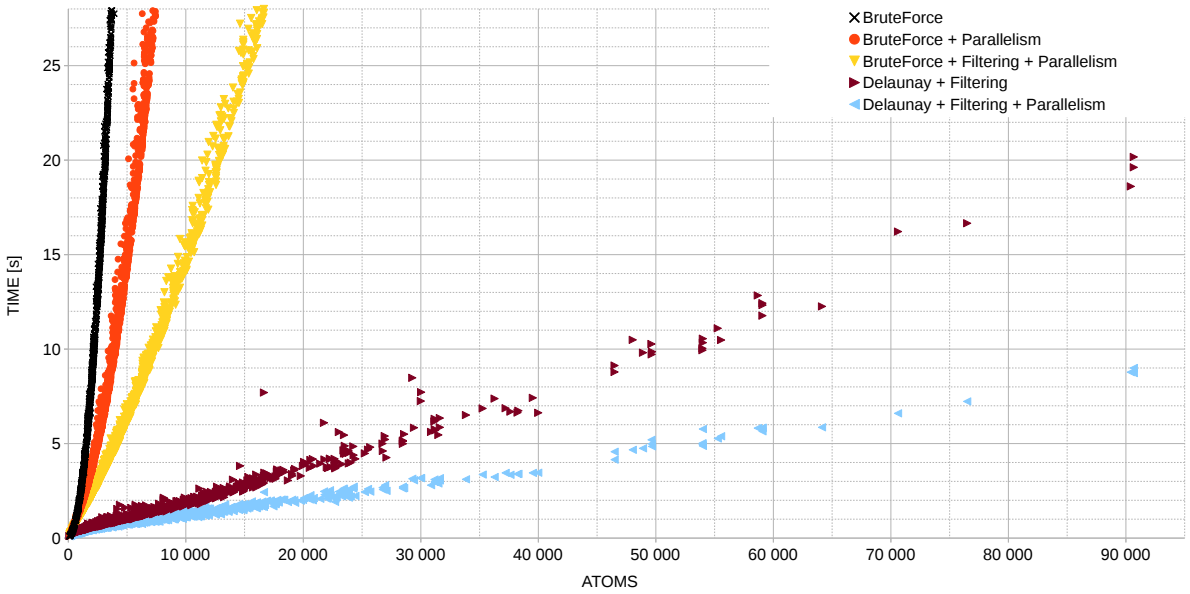


Figure 6.8: Voronoi diagram computation time, focus on Delaunay strategies

Thousands of time samples were collected for both applications. Each sample is the time interval in seconds from application start to its end for one PDB model. It includes any overhead caused by reading the input data, writing the output data and preprocessing (Delaunay triangulation in the case of AwVoronoi) for the sake of a fair comparison between AwVoronoi and QTFier because QTFier does not provide any performance information. This overhead should be bigger for AwVoronoi because AwVoronoi works with XML files instead of plain text files. For instance, QTFier had to read 2 GB of plain text data but AwVoronoi 24.1 GB XML data. All data were stored on a fast SSD to minimize the influence of I/O operations. Results indicate that AwVoronoi is faster.

6.5 Chapter Conclusion

The most time-consuming part of the edge tracing algorithm is finding the Voronoi vertices along edge trajectories. A brute force approach must test almost all sites per trajectory. Using parallelism and filtering improves the performance by a multiplicative constant. A better method was presented in this chapter. The proposed method employs dynamic spatial filters to limit the number of tested sites. Sites hitting the filter are searched for via the graph of Delaunay triangulation vertices and edges built on site centers. The filter may shrink in size as the search progresses. The proposed method has been tested on thousands of molecular models. Experiments indicate a significant performance improvement over other methods.

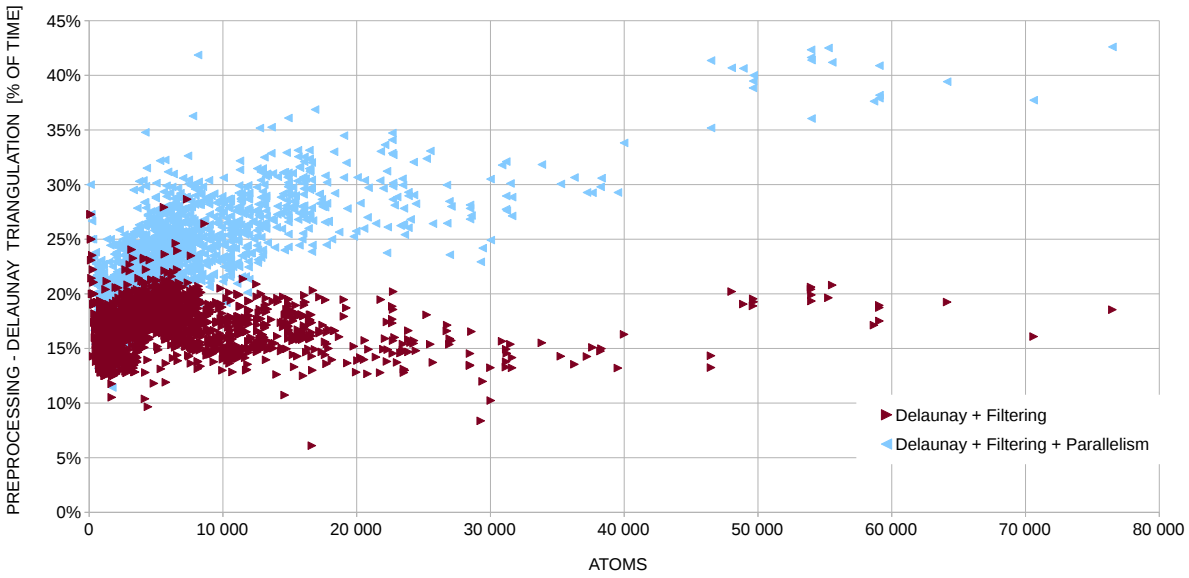


Figure 6.9: The percentage of preprocessing time in Delaunay strategies. The computation of Delaunay triangulation does not use parallelism.

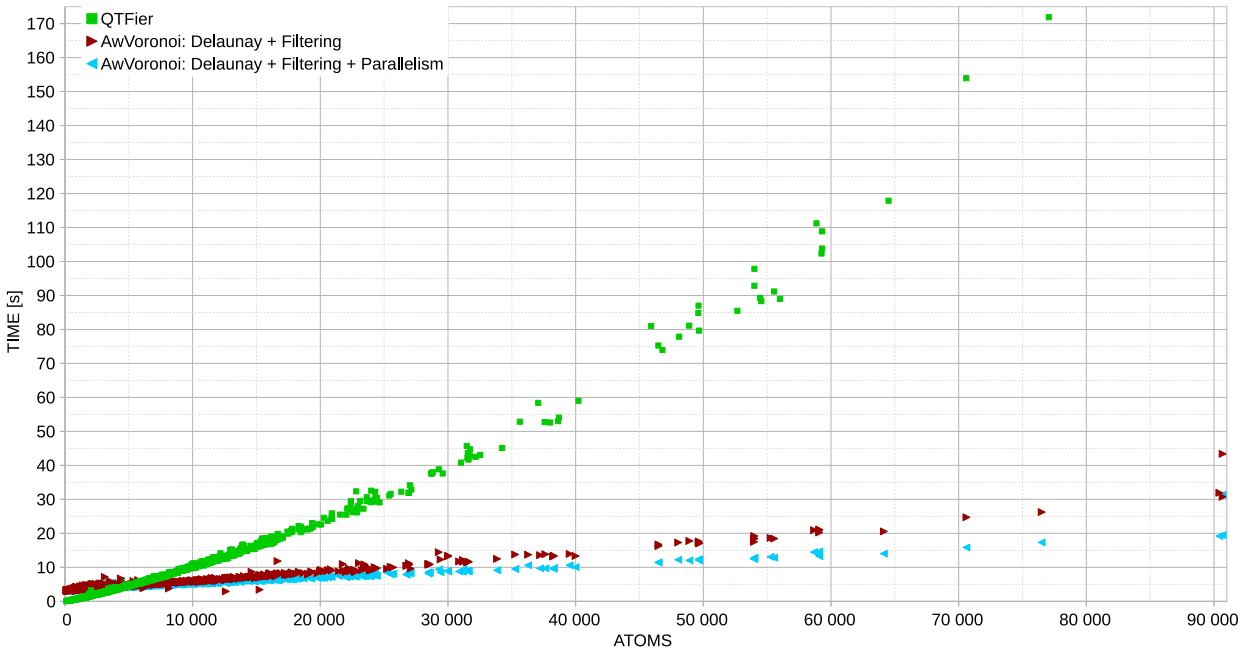


Figure 6.10: Comparison of AwVoronoi [15] using the strategy of dynamic filters and Delaunay triangulation against QTFier [110] using the strategy of static filters and a uniform grid. The computation time includes I/O overhead.

Chapter 7

Extension of Edge Tracing to Disconnected Voronoi Skeletons

The edge tracing algorithm first locates an initial Voronoi vertex and then traces Voronoi edges until the whole component is discovered. However, the diagram can have more components. Some authors suggest to search for components until each site contributes to some component [60, 87, 93], but this strategy is insufficient. Even if the occurrence of such cases may be rare in some data [48], a good algorithm should be able to handle them, too. Manak and Kolingerova were the first who noticed this problem and proposed a solution [85]. It is an extension of the edge tracing algorithm, which ensures that the algorithm will find all components.

The proposed method is inspired by an earlier idea that components can be arranged into a hierarchy [60]. The novel observation is that the bottom of the hierarchy is easily discoverable and at least one leaf node can be identified with a little effort. The method then tears leaves of the hierarchy and discovers missing components when they appear at the bottom of the hierarchy.

This chapter is organized as follows. For the sake of simplicity, we will assume that the conditions listed in the first section of this chapter hold. Advanced properties of aw-Voronoi diagrams are described next, in particular the pair of sites important for each component and the hierarchical arrangement of components. These properties are fundamental for the description of the proposed method. After that, experiments and results of the proposed method are presented and the method is compared to others. The next sections give a clue how to remove the initial assumptions.

7.1 Initial Assumptions

To simplify the explanation of the proposed method, we will avoid negative weights of sites by Assumption 1, we will also avoid complications with empty Voronoi regions by Assumption 2 and with partially unbounded Voronoi regions by Assumption 3.

Assumption 1. *Each site $s_i \in S$ has $weight(s_i) > 0$, so s_i is a sphere with the center $c_i = position(s_i)$ and the radius $r_i = weight(s_i)$.*

Assumption 2. *The spheres of sites may intersect but not fully contain each other.*

Assumption 3. *Only two spheres contribute to $CH(S)$.*

Note that Assumption 1 is without loss of generality because $VD(S)$ does not change if a constant is added to the weights of all sites. Assumption 3 is also without loss of generality if it is interpreted as an embedding: For a given non-empty set S_A of spheres, one can create a set $S_B = S_A \cup \{s_b\}$ satisfying Assumption 3, where s_b is a sufficiently large and distant sphere and the skeleton of $VD(S_A)$ including elements at infinity is topologically equivalent to the skeleton of $VD(S_B)$ without elements at infinity. Actually, $VD(S_B)$ will have one loop edge at infinity and two faces at infinity glued by the edge.

7.2 Advanced Properties

In general, aw-Voronoi faces can have interior holes. The boundary of such a hole is a topological circle made of one or more elliptic edges. They belong to some component of the whole Voronoi skeleton. Each component creates a hole in some face. The face lies in the intersection of two regions. The two sites of these regions give a means for arranging components into a hierarchy, which will help us to devise an extension of the edge tracing algorithm to discover the whole Voronoi skeleton. We will first characterize the pair more precisely, next we will describe how to find it for a given component, and then we will describe the hierarchy.

We will also need the following term. Given a set of components $\{K_1, \ldots, K_n\}$, a site s is *isolated* if it does not contribute to the definition of any of the components, i.e., if $s \notin \bigcup_{i=1}^{n} S(K_i)$.

7.2.1 Big Brothers

Each Voronoi skeleton component K_i creates a hole in some face $f \subseteq VR_a \cap VR_b$, where VR_a and VR_b are Voronoi regions. Sites s_a, s_b corresponding to these regions will be called *big brothers* of the component K_i. The pair will be denoted $BB(K_i) = \{s_a, s_b\}$. See Figure 7.1 for an example. Let us have one more component K_j that contributes to the boundary of f. Even if $\{s_a, s_b\} \subseteq S(K_i) \cap S(K_j)$ because both K_i and K_j contribute to the boundary of f, the pair $\{s_a, s_b\}$ plays the role of big brothers in the context of K_i and not in the context of K_j.

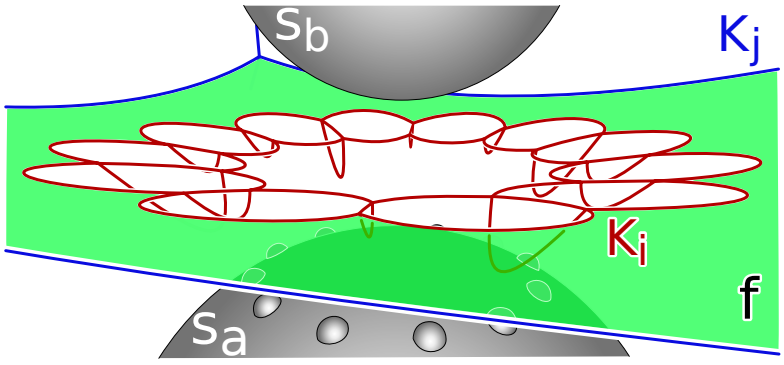
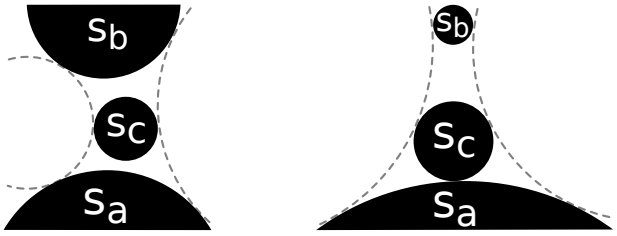


Figure 7.1: Component K_i creates a hole in a face $f \subseteq VR_a \cap VR_b$. $BB(K_i) = \{s_a, s_b\}$ is the pair of big brothers of K_i

Big brothers have originally been introduced by Kim et al. [51–53, 60] to name a configuration where small spheres are located between big spheres as illustrated in Figure 7.2(a), however, note that only one sphere of the pair is big whereas the other can be small as illustrated in Figure 7.2(b). The pair has also been named a capsule-generating pair or a gate pair [58]. In the rest

of this section we will introduce new characteristics of big brothers and describe how to find this pair for a given skeleton component.



(a) Both s_a and s_b are big and s_c is small

(b) Only s_a is big, s_b is small

Figure 7.2: Big brothers s_a and s_b

Lemma 2. $|S(K_i) \cap S(K_j)| \leq 2$ *for all mutually different components* K_i, K_j *of a Voronoi skeleton.*

Proof. Let $f \subseteq VR_a \cap VR_b$ be a face with a hole and s_a, s_b be the sites of VR_a, VR_b. The boundary of the hole belongs to some component K_i. A *shield* of K_i is the union of straight line segments drawn from the centers c_a, c_b of s_a, s_b to all points on hole boundary. The shield is a topological sphere with poles c_a, c_b, and hole boundary on the equator. No other component can penetrate the shield because it would break the definition of VR_a or VR_b along the penetrated line segment of the shield. The whole K_i except the equator must lie inside the shield. Also centers of $S(K_i)$ must lie inside, otherwise we could intersect the shield with a straight line segment drawn from a center c_o lying outside the shield to a point lying on an edge of K_i inside the shield. If this edge is from VR_o, then we will get a contradiction with the definition of VR_a or VR_b in the intersection point. Shields of two components K_i and $K_j \neq K_i$ can be either separated from each other or one shield can be enclosed by the other. The enclosed shield still works as a protection against the other component. In both cases, only the poles can be shared, so $|S(K_i) \cap S(K_j)| \leq 2$. □

Lemma 3. *For each Voronoi skeleton component* K_i, *the pair* $BB(K_i)$ *satisfies* $CH(BB(K_i)) = CH(S(K_i))$.

Proof. Let $f \subseteq VR_a \cap VR_b$ be a face with a hole caused by K_i. Let $BB(K_i) = \{s_a, s_b\}$ be the pair of sites corresponding to VR_a and VR_b. The geometry of each edge e in the boundary of the hole is defined by $\{s_a, s_b, s_c\}$, $s_c \in S(K_i)$. The geometry must be an ellipse, otherwise e would extend to infinity or hit the boundary of f that belongs to another component. Therefore, the sphere s_c is in the interior $CH(BB(K_i))$ [113, Lemma 2]. To prove it for other sites, consider a simpler diagram $VD(S(K_i))$. Its skeleton is just K_i. The surface of $CH(S(K_i))$ cannot contain any triangle because it would indicate the existence of an edge of K_i leading to infinity. The surface of $CH(S(K_i))$ cannot contain more than one full conical (or cylindrical) patch because it would indicate the existence of another Voronoi face into which K_i creates a hole. A component $K_j \neq K_i$ contributing to the boundary of both faces can be made and $|S(K_i) \cap S(K_j)| \geq 3$ is a contradiction. □

For a given component K, the pair $BB(K) = \{s_a, s_b\}$ can be found among members of $S(K)$ in linear time. We already know that only two spheres contribute to $CH(S(K))$, we just need to find them in $S(K)$. First a sphere $s_a \in S(K)$ is found, which has its radius maximal among all members of $S(K)$. If two spheres have the same maximal radius, then any of them can be

chosen as s_a. After that $s_b \in S(K)$ is found as the one that maximizes $||c_a - c_b|| + r_a + r_b$, i.e., the *diameter* of $CH(\{s_a, s_b\})$. The maximal value is the diameter of $CH(BB(K))$. See Figure 7.3 for an illustration.

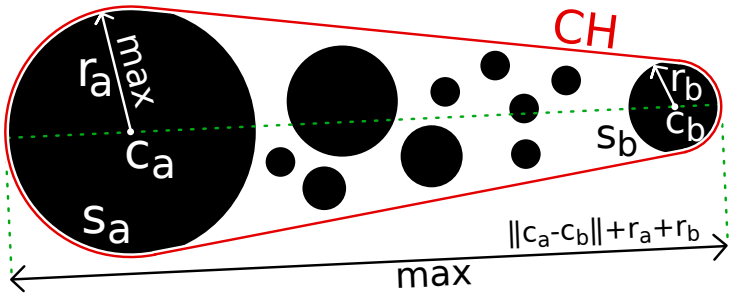


Figure 7.3: Finding big brothers s_a and s_b among sites of a component

Note that the chosen s_a necessarily contributes to $CH(S(K))$, otherwise s_a would not fit into $CH(S(K))$ defined by a pair of smaller spheres and we would get a contradiction with convex hull properties. Also note if s_b did not maximize the diameter, then $CH(S(K))$ would not fit into $CH(\{s_a, s_b\})$, which would also be a contradiction with convex hull properties.

The time complexity of finding $BB(K)$ in $S(K)$ is $O(|S(K)|)$ because it is just a maximization of a simple value over $|S(K)|$ sites followed by a maximization of a simple function over $|S(K)| - 1$ sites.

7.2.2 Hierarchy of Components

Let K_P and K_C be two components. If $BB(K_C) \subset S(K_P) \wedge BB(K_C) \neq BB(K_P)$, then K_C is a *child node* attached below the *parent node* K_P. For instance K_i is a child of K_j in Figure 7.1 and both K_2 and K_3 are children of K_1 in Figure 3.2(b). The *hierarchy of Voronoi skeleton components* is a graph of nodes and links, where each node represents a Voronoi skeleton component and children are linked to their parents.

The hierarchy has originally been introduced in [58, 60]. Our definition is different in description but should give the same (or similar) structure.

Each node K_C has the diameter of $CH(S(K_C))$ strictly less than the diameter of $CH(S(K_P))$ of its parent K_P because all members of $S(K_C)$ are enclosed by $CH(S(K_P))$. Equality of diameters is impossible, otherwise $BB(K_C) = BB(K_P)$, which is a contradiction. A direct consequence is that the hierarchy is acyclic.

The hierarchy graph is a tree or a forest. Leaf nodes are at the bottom of the hierarchy and all roots share the same pair of big brothers.

Each set M of nodes satisfying $\bigcup_{K_i \in M} S(K_i) = S$ has a nice property that the bottom of the hierarchy, i.e., the set of all leaf nodes, is a subset of M. It is a consequence of the fact that leaf nodes may contribute only with their big brothers to the sites that define other components. If a leaf $K_l \notin M$, then $S(K_l) \setminus BB(K_l)$ are isolated, which is a contradiction with the definition of M. A direct consequence is that $K_{min} \in M$, which has the minimal diameter of $CH(BB(K_{min}))$ among all members of M, is a leaf. If there was a child K_C of K_{min}, then the diameter of $CH(S(K_C))$ would be strictly less than the diameter of $CH(S(K_{min}))$, which is a contradiction.

If we consider $S' = S \setminus (S(K_{min}) \setminus BB(K_{min}))$, then the Voronoi skeleton of S' will be the same

as the Voronoi skeleton of S but without K_{min}. This way we can *hide* the component K_{min}. It is nothing more than filling the hole caused by K_{min} in the face between $BB(K_{min})$.

7.3 Proposed Method

This section describes an extension of the edge tracing algorithm to find all Voronoi skeleton components. The extension is formalized in Algorithm 5.

In Algorithm 5, each site has its own counter that keeps the number of components the site currently contributes to. Initially, all sites are isolated, so all counters are zero. Algorithm 5 has the following two parts.

First, some initial set of components is discovered. It can be any set provided that each site of S contributes to at least one component (the set M in Section 9.4). This is achieved by applying `EdgeTracing(`s_i`, S)` on each $s_i \in S$ that is still isolated. This first part has already been described in [87]. Nevertheless, some components can still be missing.

In the second part of Algorithm 5, all missing components are discovered. Components discovered so far are put into a minimum-priority queue. The priority of a component K is exactly the diameter of $CH(BB(K))$, i.e., the distance between the centers of $BB(K)$ plus their radii. Algorithm 5 then sequentially removes components from the queue in the order of ascending priorities. Each removed component K_{min} is then hidden as follows. First, all sites $S(K_{min})$ except $BB(K_{min})$ are excluded from further computation. After that, the counter on remaining affected sites (members of $BB(K_{min})$) is decremented. The counter can drop to zero on some $s_i \in BB(K_{min})$. If this happens, then s_i just becomes isolated and the edge tracing algorithm is started on s_i to discover a missing component K satisfying $s_i \in S(K)$. This K is then added to the result, counters are updated (incremented at each member of $S(K)$) and K is put into the queue.

Now we are going to discuss the worst case time complexity of Algorithm 5. Let $n = |S|$ be the number of input sites, m the number of components in $VD(S)$, and $T(K_i)$ the time of finding K_i of $VD(S)$ by `EdgeTracing()`. Initialization (lines 1-4) takes $O(n)$. The first part of Algorithm 5 (lines 5-13) takes $O(n+\sum_{i=1}^{m}(T(K_i)+|S(K_i)|))$. Initialization of the priority queue in the second part (line 14) takes $O(m)$. Each of the m components will pass through the queue exactly once. The while-loop takes at least $O(\sum_{i=1}^{m}(log(m)+|S(K_i)|))$ (without the part below the branch at line 21), where $O(log(m))$ takes `Dequeue()` and $O(|S(K_i)|)$ takes updating a bit map of hidden sites, including the realization of $S \setminus Hidden$. There can be $O(m)$ missing components. The contribution to the total time complexity of the while-loop for missing components (lines 21-28) is $O(\sum_{i=1}^{m}(T(K_i)+|S(K_i)|+log(m)))$. Big brothers are computed once per component, so the contribution to the total time complexity is $\sum_{i=1}^{m}|S(K_i)|$. The total time complexity is the sum of the parts above, i.e., $O(n+\sum_{i=1}^{m}(T(K_i)+|S(K_i)|+log(m)))$. This can be simplified because $n \leq \sum_{i=1}^{m}|S(K_i)|$, $T(K_i) \in \Omega(|K_i|)$ and $|S(K_i)| \leq 3|K_i|$, so the total time complexity is $O(m\ log(m)+\sum_{i=1}^{m}T(K_i))$. Note that the overhead to all `EdgeTracing()` calls is only $O(m\ log(m))$. Also $m < n$ and often $m \in O(1)$. This m can be further reduced to the actual number of missing components, but it would require explicit construction of hierarchy sub-trees.

Algorithm 5: FindAllComponents(S)

Input: The set of sites S
Output: The set of all components in the Voronoi skeleton of $VD(S)$

```
Components ← ∅;
foreach s_i ∈ S do
    Count(s_i) ← 0;
end
// 1. Cover all sites by components as described in [87]
foreach s_i ∈ S do
    if Count(s_i) == 0 then
        K ← EdgeTracing(s_i, S);
        Components ← Components ∪ {K};
        foreach s_j ∈ S(K) do
            Count(s_j) ← Count(s_j) + 1;
        end
    end
end
// 2. Find missing components (novel)
Q ← priority queue made of Components, items with minimal diameter of CH(BB(K)) come first;
Hidden ← ∅;
while Q not empty do
    K_min ← Dequeue(Q);
    Hidden ← Hidden ∪ (S(K_min) \ BB(K_min));
    foreach s_i ∈ BB(K_min) do
        Count(s_i) ← Count(s_i) − 1;
        if Count(s_i) == 0 then
            K ← EdgeTracing(s_i, S \ Hidden);
            Components ← Components ∪ {K};
            foreach s_j ∈ S(K) do
                Count(s_j) ← Count(s_j) + 1;
            end
            Enqueue(K, Q);
        end
    end
end
return Components;
```

7.4 Experiments and Results

This section contains some examples of applying Algorithm 5 and a comparison to other approaches.

In Figure 7.4, there are three components K_1, K_2 and K_3, with $S(K_1) = \{s_1, s_2, s_3, s_4\}$, $BB(K_1) = \{s_1, s_2\}$, $S(K_2) = \{s_1, s_3, s_5, s_6\}$, $BB(K_2) = \{s_1, s_3\}$, $S(K_3) = \{s_2, s_4, s_7, s_8\}$, $BB(K_3) = \{s_2, s_4\}$. In the hierarchy, both K_2 and K_3 are children of K_1. The component K_2 has the minimal set

diameter of the convex hull of big brothers. Right after the first part of Algorithm 5, only components K_2 and K_3 are discovered and counters of all sites are 1. Note that $\forall s_i \in S$ the method $EdgeTracing(s_i, S)$ is not able to discover any initial vertex of K_1. Sites s_5, s_6, s_7 and s_8 prevent the method from choosing a Voronoi face context for finding an initial vertex of K_1 (see Chapter 5). This is because s_5 is the nearest neighbor of s_1, s_6 of s_3, s_7 of s_4, and s_8 of s_2. The second part of Algorithm 5 will solve this problem as follows. First, K_2 is removed from the queue and hidden, i.e., s_5 and s_6 are excluded from further computation. Counters are decremented on $BB(K_2) = \{s_1, s_3\}$. When the counter of s_1 (or s_3) drops to 0, the $EdgeTracing()$ method will be called on s_1 (or s_3). Now without K_2, s_5 and s_6, the method will discover K_1. After that, counters of all sites $S(K_1)$ are incremented and K_1 is put into the queue. Note that both counters of s_2 and s_4 will be 2. Next, K_3 is removed from the queue, s_7 and s_8 are excluded from further computation and the counters of $BB(K_3) = \{s_2, s_4\}$ drop to 1. After that, also K_1 is removed from the queue, s_3 and s_4 are excluded from further computation, counters of $BB(K_1) = \{s_1, s_2\}$ drop to zero. Now only $BB(K_1)$ remain, the rest is hidden and Algorithm 5 finishes.

Figure 7.4 shows that even if all sites contribute to some component, e.g., $S = S(K_2) \cup S(K_3)$, some components (K_1) can still be missing. Thinking the opposite is a mistake in [60, 87, 93].

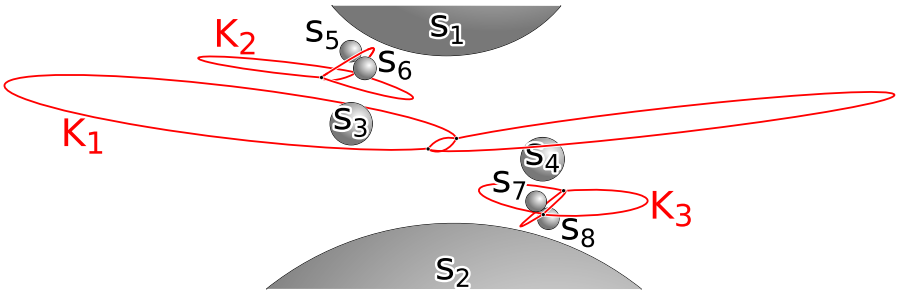


Figure 7.4: The edge tracing algorithm cannot discover the large component K_1 until one of the smaller components K_2 or K_3 is hidden.

The five sites in Figure 7.5 generate three components K_0, K_1, and K_2, made only of loop edges. Note that $S(K_0) = \{s_0, s_1, s_4\}$, $BB(K_0) = \{s_0, s_4\}$, $S(K_1) = \{s_1, s_2, s_4\}$, $BB(K_1) = \{s_1, s_4\}$, $S(K_2) = \{s_2, s_3, s_4\}$ and $BB(K_2) = \{s_2, s_4\}$. In the hierarchy, K_2 is a child of K_1, and K_1 is a child of K_0. If Algorithm 5 processes sites in the order of their indices, only K_0 and K_2 will be discovered in the first part. In the second part, K_2 will be removed from the queue, s_3 hidden and s_2 will become isolated. Then the edge tracing will start on s_2, discover K_1 and put it into the queue. Next, K_1 will be removed from the queue, s_2 hidden, but this time no site will be isolated. Finally, K_0 will be removed, s_1 hidden, s_0 and s_4 will become isolated, but no more components will be discovered because only s_0 and s_4 remain.

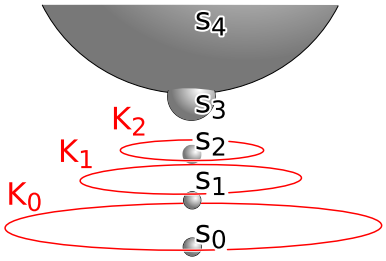
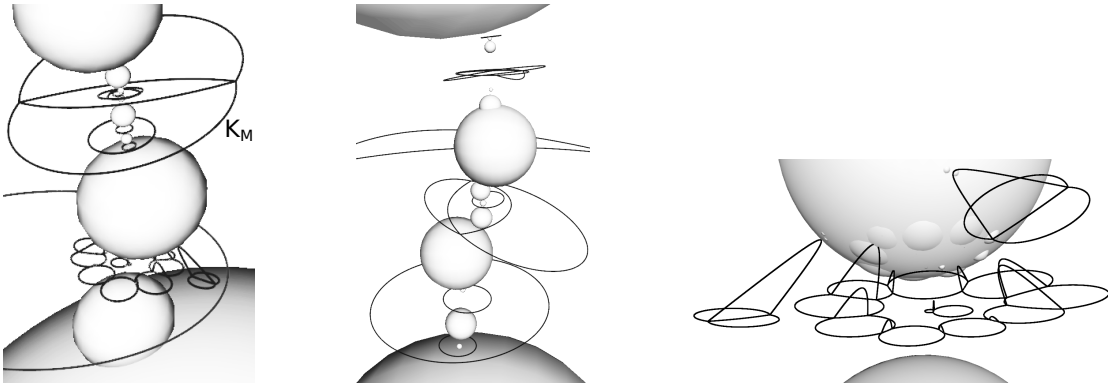


Figure 7.5: Three loop edges.

Figure 7.5 also contradicts a conjecture that a direct child can be found via the greatest isolated site [60]. In our case if we had only K_0, then $S^I = \{s_2, s_3\}$ would be the set of isolated sites and s_3 the greatest one, but $s_3 \notin S(K_1)$.

The proposed method has also been tested on problematic configurations. One of them can be seen in Figure 7.6. There are 31 spheres, the Voronoi skeleton has 14 components, 7 of them are loop edges. The hierarchy has 8 leaves, 4 of them have depth 4 and the rest 3. This configuration was designed to produce many components organized into a non-trivial hierarchy. Some of them are not discoverable by conventional methods, but the solution presented in this chaper discovers them all. Thanks to the low number of spheres, the results can be easily verified.



(a) Overview (b) The upper part (c) The lower part

Figure 7.6: A more complex example

The proposed method is implemented in a computational library awVoronoi [15]. The implementation is compared with QTFier [110], which probably implements the solution proposed in [60], and to Voronota [93]. On the configuration of spheres shown in Figure 7.4, Voronota successfully discovered Voronoi vertices of K_2 and K_3 but failed to discover any vertex of K_1. QTFier was unsuccessful. If both s_5 and s_6 were moved to achieve $BB(K_2) = \{s_2, s_3\}$, then both QTFier and Voronota successfully discover all components. On the configuration shown in Figure 7.5, Voronota did not discover any component because it finds only Voronoi vertices, but QTFier discovered all components. On the configuration in Figure 7.6, Voronota successfully discovered Voronoi vertices of all components except the one marked as K_M and except all seven loop edges, and QTFier was unsuccessful. AwVoronoi processed all examples correctly. On the other hand, Voronota has the computation of vertex coordinates more robust than AwVoronoi.

7.5 Full Inclusions Among Spheres

If full inclusions are allowed among spheres in S, then the edge tracing algorithm needs to be modified to ignore included spheres, otherwise it will not work correctly. It is a simple modification of the strategy from Chapter 5 for finding an initial Voronoi element. The first sphere $s_i \in S$ is given as an argument and the second $s_j \in S \setminus \{s_i\}$ must minimize $||c_i - c_j|| - r_i - r_j$. The minimum is found in time $O(|S|)$. If the minimum is $\leq -2r_i$, then $s_i \subseteq s_j$. All included spheres will be recognized at line 7 of Algorithm 5 as the spheres s_i for which `EdgeTracing`(s_i, S) did not discover any component. In the worst case, S contains $O(S)$ included spheres and the total time complexity will grow to $O(|S|^2)$. Additional data structures might help to improve performance in certain cases, but our method was not intended for efficient solving of a high number of inclusions.

7.6 Non-trivial Convex Hulls

If more than two spheres contribute to $CH(S)$, then some Voronoi edges or faces extend to infinity. An additional site s_∞ should be introduced to represent infinity. If an edge e extends to infinity, then a new vertex at infinity should be created for the edge. The vertex will be defined on sites $S(e) \cup \{s_\infty\}$ and its position will be given as the normal vector of one of the supporting planes of $S(e)$. Voronoi vertices at infinity should be connected by Voronoi edges at infinity. Also loop edges at infinity are possible. Elements at infinity have a one-to-one correspondence with parts of $CH(S)$ [9, 10]. The edge tracing algorithm also needs to be modified.

All Voronoi vertices and edges, including elements at infinity, create an extended Voronoi skeleton. Big brothers and the priority used in Algorithm 5 are no longer valid for all components of the extended skeleton.

If a skeleton component K has $s_\infty \in S(K)$, then we will define $BB(K) = \{s_\infty, s_a\}$, where $s_a \in S(K) \setminus \{s_\infty\}$ is the sphere with maximal radius r_a among all members of $S(K) \setminus \{s_\infty\}$ that contribute to $CH(S(K) \setminus \{s_\infty\})$. If more spheres are found, then ambiguity must be removed by a secondary decision based on another attribute, e.g., by choosing the sphere with maximal center coordinates. Priorities must also be redefined. In the priority queue, all components K_j with $s_\infty \in S(K_j)$ must be sorted behind all ordinary components K_i with $s_\infty \notin S(K_i)$. Among all K_j with $s_\infty \in S(K_j)$, the radius r_a of the site $s_a \in BB(K_j) \setminus \{s_\infty\}$ is the priority of K_j.

7.7 Chapter Conclusion

An extension of the edge tracing algorithm was presented. The extension guides the edge tracing to discover all components of the Voronoi skeleton. The method adds only $O(m\ log(m))$ time complexity, where m is the number of Voronoi skeleton components. The proposed method extends the applicability of the edge tracing algorithm to general non-degenerated input data.

Chapter 8

Interactive Detection and Visualization of Cavities for Various Probes

A method is presented in this chapter, which discovers all cavities for the given probe size and renders their Connolly surface. These cavities are analyzed for advanced properties and filtered, so it is possible to hide cavities that do not satisfy certain criteria. The proposed method allows to change the probe size interactively. The novelty is in the combination of previous methods together with a bit of own invention to overcome their weak spots. The full article by Manak, Jirkovsky and Kolingerova can be found in [82].

Recall that Connolly surface consists of pieces of concave spherical triangles, saddle-shaped toroidal patches and convex spherical patches. Several problems need to be solved during the detection of cavities and the computation of Connolly surface. The first problem is finding the triplets of sites on which spherical triangles are created, the pairs on which toroidal patches are created and the sites that contribute to spherical patches. The second problem is the classification, which parts constitute which cavity. This is not trivial because the boundary of a cavity can be disconnected. At last, all these parts must be quickly rendered.

The aw-Voronoi diagram can be used to solve some of these problems. Each Voronoi edge corresponds to one or two spherical triangles of the Connolly surface for some probe size. So all what is needed is an efficient way of extracting the right information for the given probe size. Furthermore, the aw-Voronoi diagram allows to explore the whole space, which simplifies the classification of surface parts into cavities as well as the computation of their advanced properties. The principle is illustrated on a 2D analogy in Figure 8.1.

The method presented in this chapter combines the basic idea of navigating the probe center along Voronoi edges [101] with a fast rendering of Connolly surfaces via ray casting based on the ideas of Krone [69] and Lindow [78]. In addition to [101], the proposed method also detects cavities and disconnected cases are also discussed. This is very efficient if Voronoi vertices are sorted first. A similar effect could be achieved with beta shapes, however, the formulation in terms of the aw-Voronoi skeleton is more intuitive because it has a direct geometric interpretation and hence understanding the theory behind beta shapes is not required. In addition to [69, 78], the proposed method fully clips all surface parts during rendering, so it is possible to observe the surface from both sides, hide certain cavities, etc.

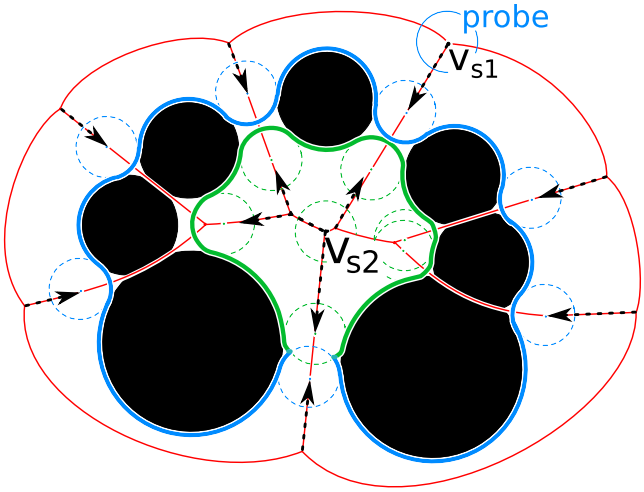


Figure 8.1: The probe center is navigated through the Voronoi network. Collisions with atoms are detected on Voronoi edges and the probe is not navigated behind such edges. The outer Connolly surface is found when the probe starts at v_{s1}, a cavity is found when the probe starts at v_{s2}.

8.1 Method Overview

The method consists of several parts, which are shown in Figure 8.2 and will be discussed in the following sections. Structures independent of the probe radius are computed in preprocessing. Next, surface elements are found via a probe navigation. After that, intersections among spherical triangles are located and stored as clip planes. At last, surface elements are uploaded to GPU and rendered. As long as the probe size stays the same, data in GPU memory can be reused.

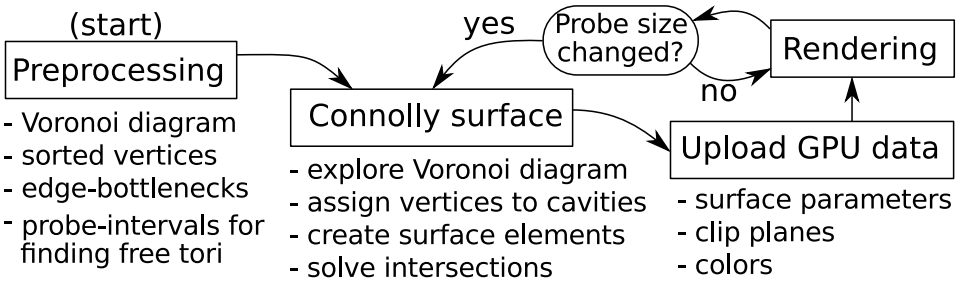


Figure 8.2: Method overview

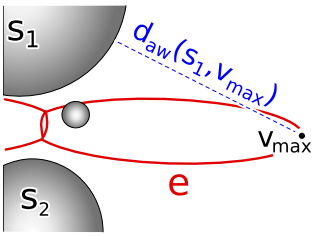
We will assume that a data structure is available, which provides the list of Voronoi vertices and edges, their incidence relationships, references to the sites that define the geometry of each vertex and edge, the position and the radius of each vertex and the bottleneck radius of each edge. The *vertex radius* is the aw-distance from its position to any of its defining sites. The *bottleneck radius* is the minimum aw-distance on the edge w.r.t. any of the defining sites and its purpose is to decide whether the probe can pass through the edge: If the bottleneck radius is less than the probe radius, then the probe cannot pass through.

8.2 Preprocessing

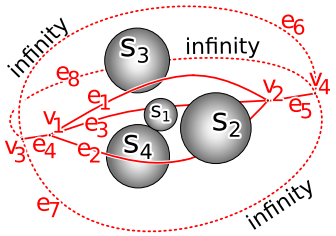
Structures computed in preprocessing will make the computation of the Connolly surface efficient as they do not have to be recomputed for different probe sizes. The following steps are performed in preprocessing.

1. Computation of Voronoi vertices and edges
2. Sorting of vertices in descending order by their radii
3. Computation of bottlenecks of all edges
4. Computation of probe-intervals for finding free tori

Voronoi edges and vertices are computed by the edge tracing algorithm, which was discussed in Chapter 5. Implementations of this algorithm are freely available [15,110]. The diagram needs to be slightly modified as illustrated in Figure 8.3 because the proposed method needs to start a graph search algorithm from any point of a local supremum of the aw-distance. These points are on edges but not necessarily in vertices. Our method always starts from a vertex, therefore, if an edge has the geometry of an ellipse, an artificial vertex (of degree two) is put on the edge into the point of maximal aw-distance, see Figure 8.3(a). Another modification deals with unbounded edges and faces. If an edge/face is unbounded, it will be closed by an artificial vertex/edge at infinity, see Figure 8.3(b). This will allow the method to continue beyond the elements that would otherwise be unbounded.



(a) An artificial vertex v_{max} in the point of max. aw-distance from edge-defining atoms



(b) Artificial vertices v_3, v_4 and edges e_6, e_7, e_8 at infinity

Figure 8.3: Artificial elements added to the diagram to simplify the surface construction

After the diagram has been computed, vertices are sorted by their radii. This order will improve the running time of the algorithm because it will process only the vertices having the radii greater than or equal to the specified probe radius and leave remaining vertices intact as indicated in Figure 8.4.

Next, the bottleneck radius of each Voronoi edge is found. A candidate to the bottleneck is the point with minimal aw-distance on the edge curve. This candidate is the bottleneck if it lies on the Voronoi edge, otherwise the boundary vertex with minimal radius is the bottleneck point. This subject was discussed in Section 5.3.

Finally, the preprocessing for the identification of free tori is performed. It can be interpreted as a simplification of beta-intervals from the theory of beta-shapes [59]. In this step, atom pairs that are able to create free tori are collected and a single interval is stored at each pair, giving the minimal and maximal radius of a probe to create a free torus. Details are given below.

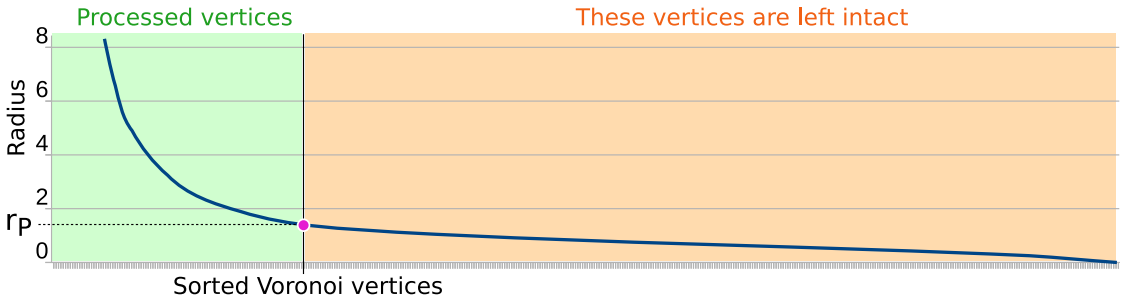
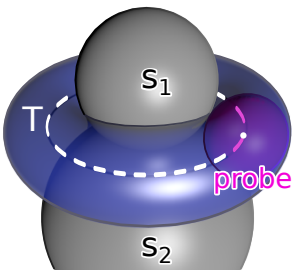


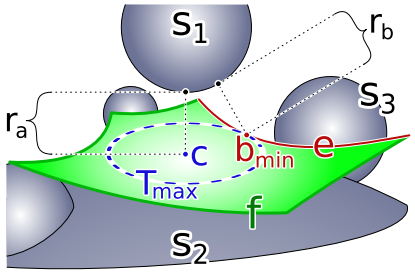
Figure 8.4: If Voronoi vertices are sorted, only the vertices with radii at least as big as the probe radius r_P can be processed and the rest can be left intact. PDB ID: 1CQW.

A torus is created by a probe rolling around a pair of atoms as shown in Figure 8.5(a). It is a free torus if the probe does not hit another atom, or equivalently, if the *main circle* T (the circular trace of the probe center) lies completely on a Voronoi face. In Figure 8.5(b), a Voronoi face f is given by atoms s_1 and s_2. Each circle $T \subset f$, concentric with but not exceeding T_{max} (the main circle of the maximal possible free torus), is the main circle of a free torus. All other tori would hit the atom s_3. The probe-interval for free tori is $(r_a, r_b]$, where $r_a = d_{aw}(s_1, c)$, $r_b = d_{aw}(s_1, b_{min})$, c is the point with minimum aw-distance from both atoms, i.e., the center of their minimal inscribed sphere, and b_{min} is the minimal bottleneck in the boundary of f and all other faces with geometry given by the pair $\{s_1, s_2\}$. Note that T_{max} passes through b_{min}. In edge connected diagrams, the probe-interval is valid only if c lies on a Voronoi face, i.e., if and only if $d_{aw}(s_1, c) \leq d_{aw}(s_i, c)$ for each Voronoi neighbor s_i of s_1.

Figure 8.6 illustrates a situation with two intervals. The proposed method still does not handle these situations. A clue how to compute these intervals can be found in Chapter 9.

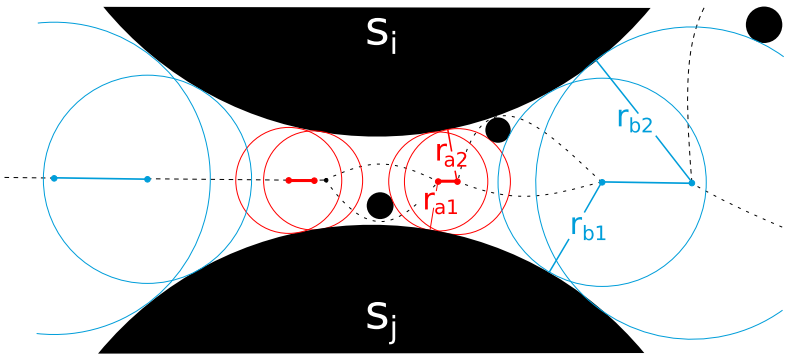


(a) A torus and its main circle T for a pair of atoms s_1, s_2.

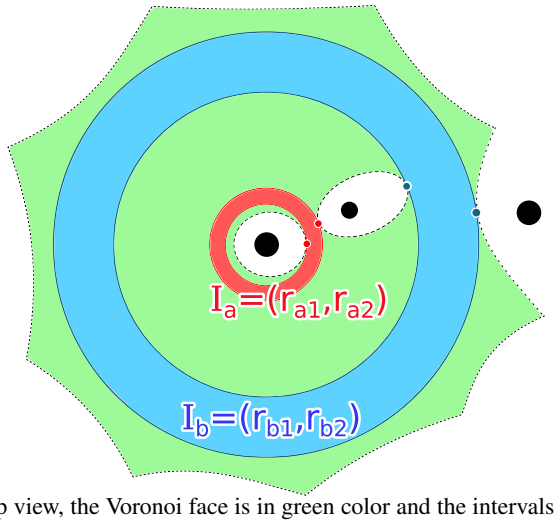


(b) Each circle $T \subset f$, concentric with but not exceeding T_{max}, is the main circle of some free torus.

Figure 8.5: The probe-interval $(r_a, r_b]$ for free tori on a pair s_1, s_2. The point $b_{min} \in e$ is the minimal bottleneck of all edges on the boundary of all faces given by s_1, s_2.



(a) Side view with inscribed spheres of minimum and maximum size. The radii of these spheres give two intervals: $I_a = (r_{a1}, r_{a2})$ and $I_b = (r_{b1}, r_{b2})$.



(b) Top view, the Voronoi face is in green color and the intervals I_a and I_b are highlighted as concentric rings.

Figure 8.6: The case of two intervals for atoms s_i and s_j. Atoms are in black color, dashed lines represent the Voronoi diagram. Given a probe radius r_p, a free torus can be created on atoms s_i and s_j if and only if $r_p \in I_a \cup I_b$

8.3 Finding Connolly Surface Elements

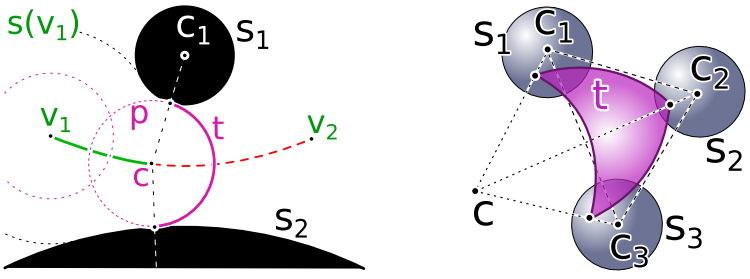
This section describes how to find Connolly surface elements for the given probe radius. First spherical triangles, then toroidal patches, free tori and finally spherical patches. All these parts will be divided into groups corresponding to cavities.

Spherical Triangles

The center of each spherical triangle lies on a Voronoi edge. All spherical triangles will be discovered during a sequence of graph searches on Voronoi vertices and edges. All vertices that belong to cavities will be visited and divided into groups. However, some groups may still need to be merged later (via toroidal patches) to have a one-to-one correspondence of groups and cavities.

Finding spherical triangles is formalized in Algorithm 6. Voronoi vertices are already sorted by their radii from big to small and stored in an array, which is available from the preprocessing stage. The array is processed sequentially until a vertex radius less than the probe radius is detected. Processed vertices are marked. If an unprocessed vertex is detected at the current index i, a graph search is started from that vertex. The search goes only on Voronoi edges having the bottleneck radius at least as big as the probe radius, marks visited vertices as processed and assigns them to the group of the starting vertex (index i). When the search is in some vertex j and detects an incident edge with the bottleneck radius less than the probe radius, the probe cannot pass through and a spherical triangle is created. The triangle stores a reference to the corresponding Voronoi vertex and its incident Voronoi edge. This information is sufficient to compute its geometry, i.e., the center of the probe sphere on which the triangle lies and its three corners.

Figure 8.7 illustrates the creation of a spherical triangle. The unknown probe center c is a solution to a similar system of equations as in the case of Voronoi vertex coordinates in Section 5.2, but now with only three spheres as the input instead of four and with r set to the given probe radius, i.e., $r = ||c - c_i|| - r_i$ for all $i \in \{1,2,3\}$ in Figure 8.7(b). It leads to a simple system of two linear equations in three variables (the coordinates of c) and one quadratic equation in a chosen free variable. There can be two solutions. The one which is closer to the Voronoi vertex along the Voronoi edge is the correct one.



(a) 2D analogy, t is a circular arc (b) A spherical triangle t

Figure 8.7: A surface element t is created from a Voronoi edge. The probe p cannot move from the vertex v_1 to v_2 because it would hit the atoms s_1 and s_2. The furthest point that can be reached is c, where p touches both atoms and creates t.

A data structure for spherical triangles must provide a way to identify the vertex and its

Algorithm 6: FindSphericalTriangles(probeRadius, vertexRadius[.], vertexDegree[.], vertexNeighbor[., .], bottleneck[., .])

Input: Voronoi vertices are sorted by the radii of their spheres from big to small. For each vertex, the array vertexRadius provides the vertex radius, the array vertexDegree provides its degree, the array vertexNeighbor provides its neighboring vertices, and the array bottleneck provides the bottleneck radii at incident edges.

Output: All spherical triangles and their groups. Each triangle contains a reference to the vertex and edge where it was discovered. Groups are recorded at vertices.

```
1  ST ← ∅; M ← ∅; // ST: spherical triangles; M: marked vertices
2  for i ← 0; i < the count of all vertices; i ← i + 1 do
3      if vertexRadius[i] < probeRadius then
4          break;
5      if i ∉ M then
6          M ← M ∪ {i}; Q ← {i}; // Mark the vertex i, put it to the queue Q.
7          while Q ≠ ∅ do
8              j ← any member of Q; Q ← Q \ {j}; // Remove a vertex j from Q.
9              group[j] ← i;
10             for k ← 0; k < vertexDegree[j]; k ← k + 1 do // k: neighbor-slot
11                 l ← vertexNeighbor[j, k];
12                 if bottleneck[j, k] < probeRadius then // Probe cannot reach l via k
13                     ST ← ST ∪ {new SphericalTriangle(j,k)};
14                 else if l ∉ M then // The probe can reach unprocessed neighbor l.
15                     M ← M ∪ {l}; Q ← Q ∪ {l};
16                 end
17             end
18         end
19     end
20 end
21 return ST and group; // All spherical triangles and their groups.
```

incident edge. The group, into which the triangle belongs, is inherited from the group assigned to the Voronoi vertex by Algorithm 6. It is also wise to store probe centers at spherical triangles because these centers will be needed for the construction of toroidal patches. The data structure should also reference three incident toroidal patches, but these patches will be constructed later.

The proposed method searches for spherical triangles similarly as [101]. The difference is that the proposed method requires sorted Voronoi vertices, it deals with cavities and it was designed to handle also elliptic edges. The proposed detection of cavities is simpler than the methods relying on beta shapes [56, 61, 66].

Toroidal Patches

After all spherical triangles are found, their sides need to be joined via toroidal patches. Each spherical triangle t_i touches three atoms, so each t_i gives three pairs of atoms on which a toroidal patch must be created. In Figure 8.8 for instance, the pairs given by t_i are $\{s_1, s_2\}$, $\{s_1, s_3\}$ and $\{s_2, s_3\}$. Each pair can be shared by two or more spherical triangles. In Figure 8.8 for instance,

the pair $\{s_1, s_2\}$ is shared by triangles t_i and t_j. To create all toroidal patches for a pair of atoms, all spherical triangles sharing the pair are stored in a list and sorted by the rotation angle[1] of their corresponding probe centers around the torus axis. The axis is given by the centers of the atom pair. All these probe centers lie on a circle. After that, a toroidal patch is created for each consecutive pair of triangles in the sorted list.

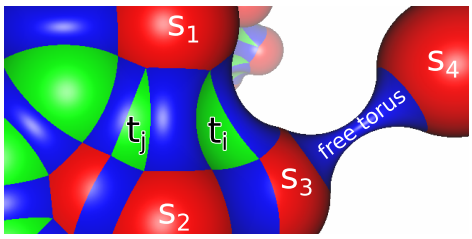


Figure 8.8: Spherical triangles t_i and t_j are joined via a toroidal patch. The patch belongs to the pair s_1, s_2. A free torus is created at the pair s_3, s_4.

Toroidal patches of an atom pair correspond one to one with arcs of the torus main circle T lying on Voronoi faces defined by the atom pair. This is illustrated in Figure 8.9, where c_{ti} denotes the probe center of the spherical triangle t_i. Arcs (c_{t1}, c_{t2}), (c_{t3}, c_{t4}), (c_{t5}, c_{t6}) and (c_{t7}, c_{t8}) correspond to toroidal patches. Note that it is necessary to distinguish correct arcs from complementary arcs (gray dashed arcs in Figure 8.9). The center of the atom defining the triangle t_1, except the two atoms defining the torus axis, can be used for this: If the atom center is sorted around the torus axis and it does not fall between c_{t1} and c_{t2}, then all these pairs are correct, otherwise they are complementary arcs.

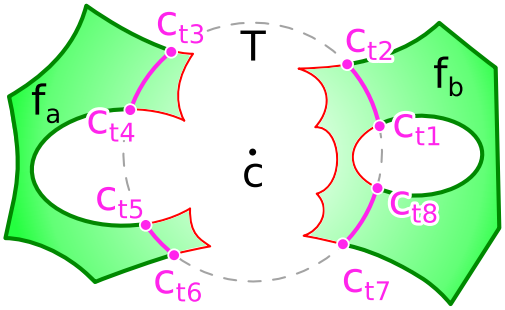


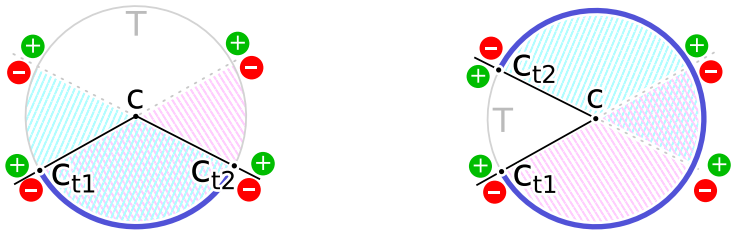
Figure 8.9: Toroidal patches correspond to arcs given by the intersection of the torus main circle T with Voronoi faces f_a and f_b. Spherical triangles $t_1 \ldots t_8$ are sorted by centers $c_{t1} \ldots c_{t8}$ of their spheres and joined via toroidal patches.

The toroidal patch data structure stores pointers to both atoms defining the torus and to both spherical triangles. The groups of these triangles need to be merged if they are different, e.g., the group of t_3 corresponding to c_{t3} in Figure 8.9 with the group of t_4 corresponding to c_{t4}. The group of the toroidal patch is inherited from any of these two triangles.

A toroidal patch is cut from a full 2π torus by a capped cone and two clip planes. The axis of the cone coincides with the torus axis. Caps are perpendicular to the axis. Each clip plane contains the torus axis and the center of a spherical triangle, and is oriented to contain the spherical triangle in the positive half-space as illustrated in Figure 8.10. If the arc is $\leq \pi$, then

[1] This sorting can be implemented without any evaluation of angles.

the patch lies in the intersection of negative half-spaces, otherwise in the union. A boolean flag is sufficient to distinguish between cases.



(a) Intersection for arcs $\leq \pi$ (b) Union for arcs $> \pi$

Figure 8.10: A toroidal patch, represented as an arc of the torus main circle T, has two clip planes. Each contains the torus axis c and the center c_{ti} of a spherical triangle t_i.

Free Tori

There can also be the so-called free tori. They are not incident to any spherical triangle. A free torus is given by an atom pair defining the geometry of a Voronoi face. The set of candidates for these pairs can be obtained from the quadruplets of atoms that define Voronoi vertices of each group, giving $\binom{4}{2}$ candidates per vertex. A candidate creates a free torus if the probe radius lies in the probe-interval mapped to the pair during preprocessing. The group of the free torus is inherited from Voronoi vertices.

Disconnected Cases

In disconnected cases, we propose to add some bridges as shown in Figure 9.6. Their geometry is irrelevant, they should only be used for merging groups. Each component of edge connectivity makes a hole in some face f. Let the face f be given by some atoms s_i, s_j. The point c in Figure 9.6 marks the point of minimum aw-distance between s_i and s_j satisfying $d_{aw}(s_i,c) = d_{aw}(s_j,c)$. The boundary of the hole consists of elliptic edges and contains virtual vertices that represent extreme points (aw-distance maxima on elliptic edges). Let v be the virtual vertex maximizing $d_{aw}(s_i,v)$ and T_v the circle satisfying $\forall x \in T_v \; d_{aw}(s_i,x) = d_{aw}(s_j,x) = d_{aw}(s_i,v)$. If $T_v \subset f$, a new bridge e will be created to connect v to the "closest" bottleneck point $b \in f$. The bottleneck point b is chosen among bottlenecks of all edges in the boundary of f as the one minimizing $d_{aw}(s_i,b) \geq d_{aw}(s_i,v)$.

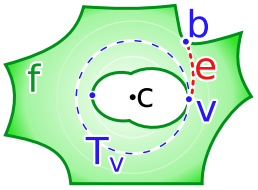


Figure 8.11: Bridge e for merging groups.

Spherical Patches

Spherical patches lie on the surface of atom spheres and their boundary, consisting of circular arcs, is formed by incident toroidal elements (toroidal patches and free tori). However, one atom can contribute to patches of different cavities and there can also be isolated atoms, which are not incident to any toroidal element. Fortunately, we already have Voronoi vertices (including virtual vertices) and toroidal elements in groups corresponding to cavities. For each group, we collect atoms that define vertices in that group. Isolated atoms are also included. Then we create a new spherical patch for each collected atom, assign it to the current group and construct its boundary. The boundary consists of circles and arcs formed by incident toroidal elements. Since each toroidal element references two atoms to which it is incident and the group to which it belongs, it is easy to assign toroidal elements from the current group to the corresponding spherical patches. A free torus creates a circle on the atom. This circle can be described by a plane intersecting the atom. Similarly, each toroidal patch creates a circular arc, which can be described by two planes. Each spherical patch references the atom sphere, the group, and the list of all these planes.

Finding Intersections among Spherical Triangles

A spherical triangle can be intersected by others. Intersecting parts must be removed for correct rendering. Previous approaches use either the Voronoi diagram of probe centers [101], a regular grid [78] or spatial hashing [70]. Our method uses hashing as well.

First, an optimization step is performed to reduce the number of triangles which need to be processed. A spherical triangle may be intersected by others only if its sphere intersects the plane passing through the centers of the three defining atoms. This property has already been described by Sanner [104].

Next, spherical triangles are mapped to a grid. Each triangle lies on a sphere of the probe radius r. The coordinates of the center of each sphere are scaled by $0.5/r$, rounded to integers and used as a key for storing the triangle in a hash-map.

Finally, intersection candidates are retrieved from the hash-map for each spherical triangle. The center of the sphere of the triangle is converted to grid coordinates and 3x3x3 cells are queried for candidates. Only those candidates which really intersect the sphere of the triangle are used for the construction of clip planes. Each intersecting candidate generates one clip plane. Some clip planes may be redundant. In our implementation, this issue has been left unsolved. A possible solution is to construct a Voronoi region (similarly as in the case of Varshney's feasible regions [109]) and further clip its faces by the triangular frustum of the spherical triangle. The faces of the resulting polytope will define all necessary clip planes.

Time Complexity

The time complexity of finding all Connolly surface elements by the proposed method, except solving the intersections among spherical triangles, is $O(V_P + T\ log(T))$, where V_P is the number of visited vertices, i.e., those with the radius at least as big as the probe radius, and T is the number of discovered spherical triangles. Note that $T \leq 4V_P$.

Spherical triangles are found in $O(V_P)$ because the algorithm visits each vertex in $O(1)$, checks its 4 bottlenecks and generates up to 4 triangles per vertex. To create toroidal patches in $O(T\ log(T))$, spherical triangles are duplicated 3 times, grouped by their 3 sides and sorted in

groups by the angle of rotation. Exactly $3T/2$ toroidal patches are then generated. Free tori are created in $O(V_P)$ if a hash-map is used for atom pairs because each vertex generates 6 candidates to a free torus and each candidate is tested against one interval. The construction of spherical patches takes $O(V_P)$.

Solving intersections among spherical triangles will add a time complexity from $O(T_I)$ to $O(T_I^2)$, where $T_I < T$ is the number of triangles that passed through the optimization test. The added time complexity depends on the input data. A reasonable expectation for protein models is $O(T_I)$ for small probes, i.e., $O(1)$ intersections per triangle, and $O(T_I^2)$ for very large probes, i.e., when each probe intersects almost all other probes. Fortunately, T_I decreases drastically with increasing probe radius.

8.4 Rendering

After the analytic description of boundary elements is computed, these elements can be rendered. We use GPU ray casting for efficient rendering of these elements, similarly as [69, 78], but we clip all surface elements.

Ray casting shaders work on the fragments generated by rendering bounding boxes of spherical triangles, and by rendering point sprites that cover toroidal and spherical patches. Only a single box is stored in GPU memory. This box is then duplicated by geometry instancing provided by OpenGL.

Clip planes are stored in a texture buffer, other per-instance data are stored in the following vertex attributes.

- *Spherical triangle*: The sphere, three corner points and their colors, the start index of clip planes and their count
- *Spherical patch*: The sphere, its color, the start index and the start index of clip planes and their count
- *Toroidal patch*: Two spheres and their colors, two clip planes and a boolean flag (see Section 8.3).

For each fragment, a ray from the camera is computed and transformed to the object space. Intersections with the object must be found. The fragment will be either discarded (no intersection, clipping) or its depth will be changed.

The intersection with a sphere leads to a quadratic polynomial and can be solved analytically [106].

The intersection with a torus leads to a quartic polynomial [20]. If the root interval is split into sub-intervals, each containing up to one root, roots can be approximated numerically. Interval split points can be found also numerically as the roots of the first derivative of the polynomial, which in turn can be found analytically as the roots of the second derivative. We approximate roots by a few iterations of the Newton's method in the root interval, i.e., $x_{n+1} = clamp(x_n - f(x_n)/f'(x_n), a, b)$ for an interval $[a,b]$.

Surface elements are clipped in fragment shaders when the intersection point of the ray with the surface is computed. The rest of this section is dedicated to this subject.

In the case of toroidal patches, the ray is first intersected with the bounding cone to get the initial root interval [78]. This removes the outer part of the torus that should not be visible. Each intersection is then tested against the bounding sphere of torus spindle to remove a possible self-intersecting part [69], and then against two clip planes to cut off the inner part that should not be visible.

In the case of spherical triangles, a fragment must be discarded if the corresponding surface point is outside the triangular frustum given by the probe center and three corners of the triangle. The fragment must also be discarded if the point is inside any of the clipping half-spaces. This can be implemented as a for-loop over these half-spaces, testing the dot product of the point with a half-space against zero.

In the case of spherical patches, a patch is given by the atom sphere and a list of circular arcs. Each arc is the intersection of a plane with the sphere, excluding a half-space, such as the arc e in Figure 8.12. Normal vectors at fragments are transformed to points in the same coordinate space as the arcs. The fragment is discarded if this point is outside the patch boundary. The test is based on counting boundary intersections with an arc drawn from the tested point to a reference point that is certainly outside as shown in Figure 8.12. The point is outside if and only if the number of intersections is even. So the fragment shader loops over all boundary arcs and counts the intersection points, which are computed as the common intersection of two planes and a sphere, excluding solutions inside a half-space, e.g., $(P \cap OF \cap S) \setminus H$ in Figure 8.12. The reference point O is chosen from the normals of the planes that define the boundary arcs because these planes separate the patch from incident tori. This fact can also be used to quickly discard many fragments outside the spherical patch, e.g., all fragments in a half-space of P.

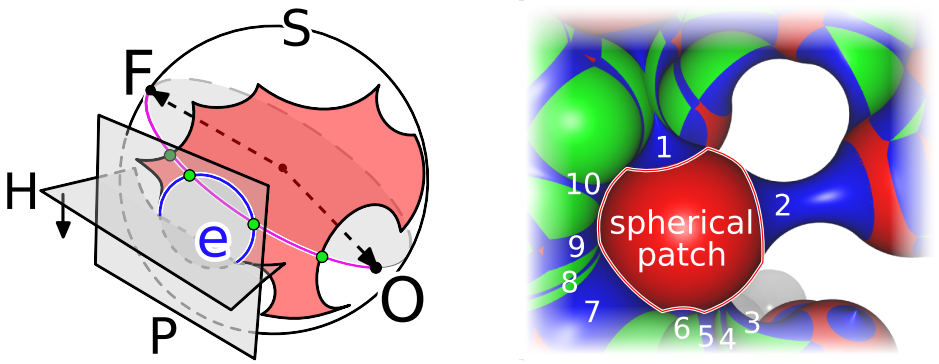
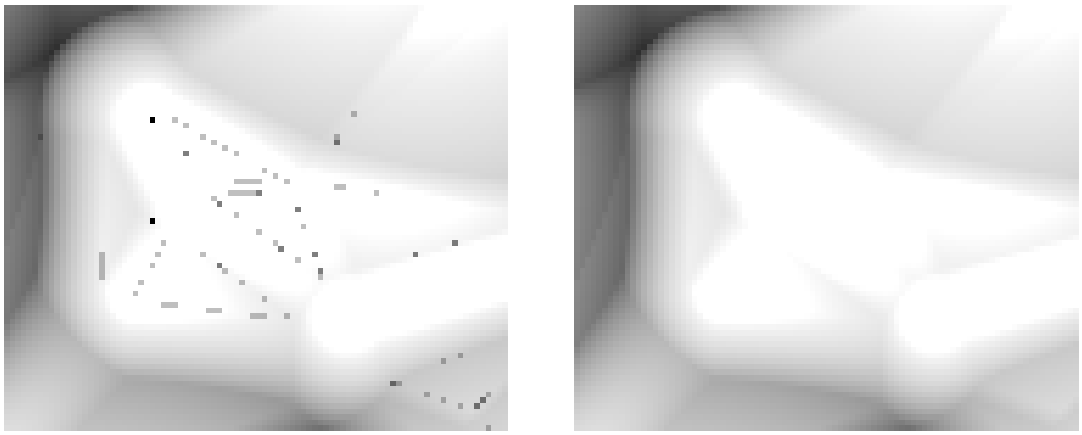


Figure 8.12: Testing the position of a point F w.r.t. a patch on a sphere S. F is outside because the arc from O to F has an even number of intersections (4) with the boundary. O is a reference point chosen to be outside, $e = (S \cap P) \setminus H$ is a boundary arc, where P is a plane and H is a half-space.

True clipping of surface elements leaves pixel-level artifacts on their boundaries. They are caused by a limited precision of computation, when a false decision is made to discard a fragment. Hardware-accelerated multi-sampling makes things even worse, so it should be disabled. Artifacts can be removed as follows. First, the scene is rendered to a custom frame-buffer using a color texture and a depth texture. After that, the default frame-buffer is switched back and a screen-filling rectangle is rendered by a post-processing shader. The shader copies the color and depth of each texel to the screen and fills pixel holes: If a fragment has at least 7 neighbors out of 8 closer to the camera, then the fragment will get a new depth and color - the average over these neighbors. Figure 8.13 shows the result.



(a) Before post-processing (b) After post-processing

Figure 8.13: Pixel artifacts are removed in post-processing

8.5 Cavity Properties

The proposed method also finds cavities. The Voronoi diagram provides much information for the computation of their various properties, especially when we have Voronoi vertices assigned to individual cavities. These properties are discussed in the following subsections.

8.5.1 Greatest Filling Sphere

The greatest filling sphere of a cavity can be located among the Voronoi vertices, which were assigned to the cavity during the Connolly surface computation.

The center of the greatest filling sphere is stable w.r.t. the probe radius, i.e., the center stays the same even if the probe radius is changed a little bit. This observation is useful in the visualization of cavities. If the color of a cavity is based on the center of the greatest filling sphere and changes of the probe radius are performed, the color of the cavity will also be stable in the visualization, see Figure 8.14. Since different cavities have different colors, the probe accessibility is visually apparent (the probe cannot get from one cavity to another). The development of space occupied by cavities and their merging can be observed as well. In our implementation, a hash value of the center of the greatest filling sphere is used as an index into the hue range of the HSB color model.

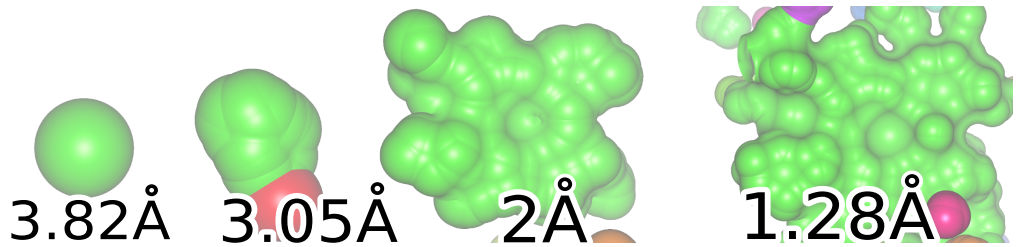


Figure 8.14: The development of a cavity w.r.t. the probe radius. The cavity has the same green color based on the center of the greatest filling sphere. PDB ID: 1AKD. Rendered by CAVER Analyst [67], using author's own plugin.

Figure 8.15 illustrates another utilization - the computation of tunnels. CAVER Analyst [67] allows to analyze and visualize tunnels leading from a point of interest to the outer surface. The computation is driven by many criteria. The center of the greatest filling sphere of a cavity is a good point of interest.

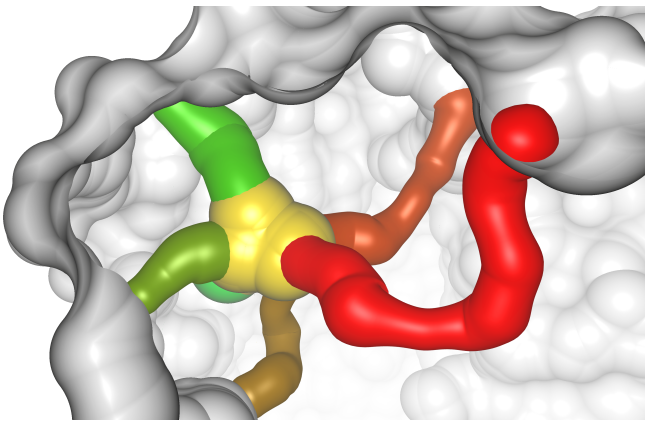


Figure 8.15: The center of the greatest filling sphere of the yellow cavity was used as a starting point for tunnels. Rendered by CAVER Analyst [67], using author's own plugin.

8.5.2 Approximate Volume

For volume computation, a cavity is filled with spheres. These are collected from the Voronoi vertices that were assigned to the cavity and from the spherical triangles of the Connolly surface. Random samples are generated in the bounding box and tested whether they hit a sphere. The volume of the bounding box multiplied by the hit probability approximates the volume of the cavity. The result may be underestimated a bit because it is not possible to fill the whole space occupied by a cavity with a finite number of spheres.

The total number of samples used to approximate the volume impacts both quality and performance. We use a multiple of the bounding box volume but limited by a constant. It can be increased manually on selected cavities.

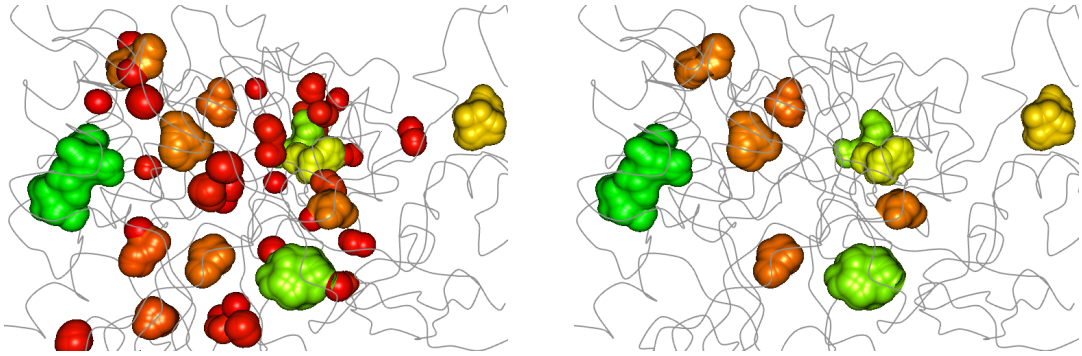
Cavities can be colored and filtered by the volume as shown in Figure 8.16. The coloring scheme is a linear interpolation from red (minimal volume) to green (maximal).

Lindow et al. use a similar method [76]. Their method creates more spheres on Voronoi edges. Voxels of the bounding box with centers in a filling sphere are counted for the volume.

8.5.3 Maximal Escaping Probe

Probe accessibility is visually apparent when cavities are distinguished by colors. The probe cannot escape from a cavity, but a smaller probe could. The maximal escaping probe is a useful characteristics. It is the parameter for which the cavity opens up and becomes accessible from the outer surface.

The maximal escaping probe radius is computed for each Voronoi vertex in preprocessing. It is a simulation of probe shrinking implemented as a search in the Voronoi graph. The search starts at edges leading to infinity and it is driven by a priority queue to choose edges with the



(a) Cavities, probe 1.4Å (b) $50Å^3 < \text{volume} < 230Å^3$

Figure 8.16: Coloring and filtering of cavities by their approximate volume. PDB ID: 1W1P. Rendered by CAVER Analyst [67], using author's plugin.

maximal bottleneck. At each newly visited vertex, the probe radius is limited by the bottleneck and recorded. During runtime, the maximal escaping probe radius of a cavity is obtained from the value stored at any Voronoi vertex of the cavity.

In Figure 8.17, the surface is clipped by a plane, atoms are hidden and cavities are colored by the center of the greatest filling sphere. Probe accessibility can be observed by changing the probe size. The big orange cavity is the active site. It is inaccessible from the outside for the probe 1.5Å. The probe escapes from the cavity at 1.4772Å.

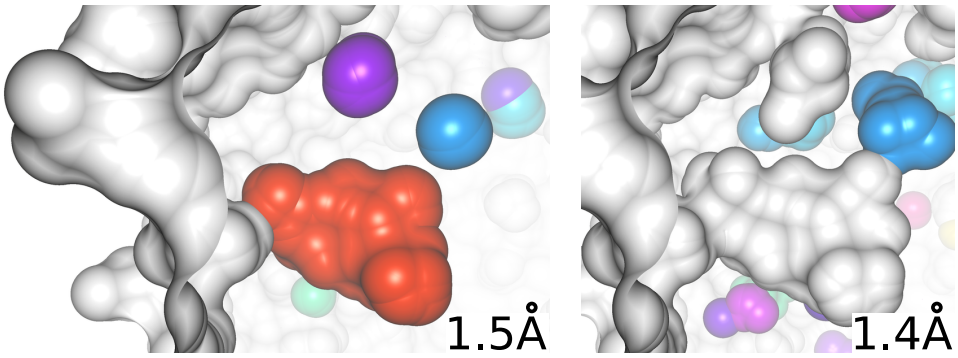


Figure 8.17: The outer Connolly surface and cavities (distinguished by colors) for different probe radii. PDB ID: 1CQW. Rendered by CAVER Analyst [67], using author's own plugin.

8.5.4 Locked Probes

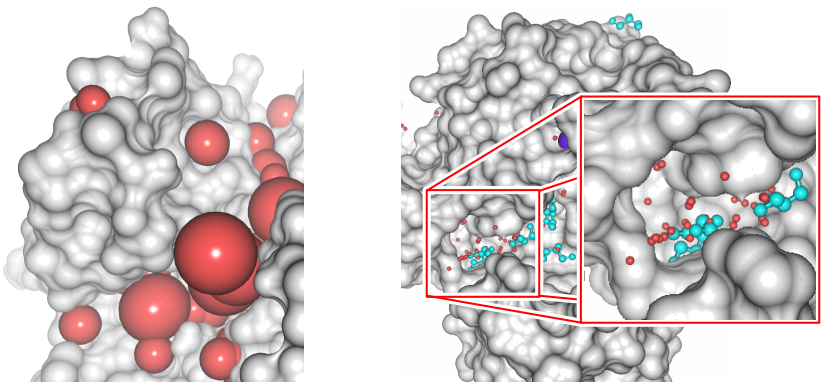
Certain configurations of atoms can hold a probe in such a way that it cannot move. Such *locked probes* may give a clue for the location of active sites.

Locked probes are equal to the greatest filling spheres of all cavities in Connolly surfaces defined by all possible probe radii. Locked probes can be easily identified as the ordinary Voronoi vertices having bottleneck radii of all incident edges less than the vertex radius, i.e., the vertices where the aw-distance function has a local maximum [77].

All locked probes are computed in preprocessing and sorted by the radius. During runtime, only the ones with the radius at least as big as the current probe radius defining the Connolly

surface are used. Since maximal escaping probes are known, it can be decided whether a probe is locked in a cavity or on the outer surface just by comparing the radius of the maximal escaping probe at the corresponding Voronoi vertex with the probe radius defining the Connolly surface.

Figure 8.18 shows locked probes in an enzyme (chitinase B) with ligands (glycerol, sulfate ions, and some inhibitors). The centers of the locked probes are displayed together with ligands. Many of them coincide with ligands.



(a) Rendered as spheres (b) Centers (red) and ligands (blue)

Figure 8.18: Locked probes. PDB ID: 1W1P. Rendered by CAVER Analyst [67], using author's own plugin.

The centers of locked probes have been introduced previously by Lindow et al. [77] as the points of aw-distance maxima on Voronoi edges, but they use it for other purposes of molecular paths extraction. They neither sort the probes for fast filtering nor they classify them w.r.t. the outer surface.

8.6 Results and Discussion

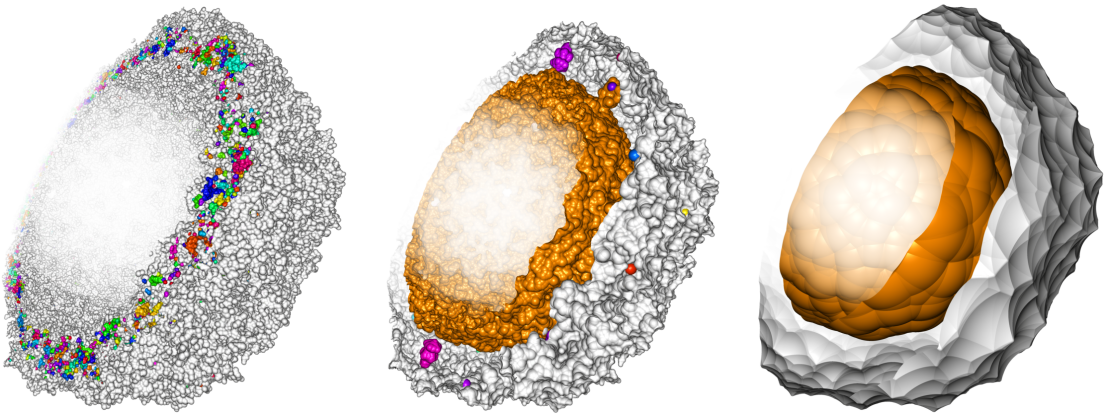
The abilities and rendering performance of the proposed method is discussed in this section and the method is also compared with others.

Figure 8.19 demonstrates the abilities of the proposed method on the model of a large virus capside. After a few seconds of preprocessing, we are able to compute the Connolly surface, detect cavities and compute their properties. We can hide the outer surface and show only cavities or quickly switch to another probe radius. Bigger probes allow to detect bigger cavities and result in less detailed surface.

The proposed method is similar to the methods that detect cavities by clipping parts of the Voronoi diagram [1, 63, 75–77]. They all extract information from the same Voronoi diagram. However, the proposed method explores only the parts of the diagram related to the empty space delineated by the Connolly surface, other parts are left intact. Our method also constructs the Connolly surface from the explored parts, whereas [75,76] create a skin surface on filling spheres and render it directly or use it for clipping the Connolly surface computed via [78].

The proposed method is better suited for static data than for large dynamic sequences because of the preprocessing step.

Table 9.1 illustrates the performance of our method for various molecular structures. The



(a) Probe radius 1Å (b) Probe radius 31Å (c) Probe radius 30Å

Figure 8.19: The Connolly surface and cavities for different probe radii, clipped by a half-space to see inside. PDB ID: 4RHV. Rendered by CAVER Analyst [67], using author's own plugin.

number of Voronoi edges is twice the number of vertices because every vertex is of degree four and no loop edges were detected. The method is implemented as a Java plugin to CAVER Analyst [67]. All experiments were performed on a system with CPU Intel(R) Core(TM) i7-4930K, 3.40GHz, 6-core (12 threads), 64 GB RAM, NVIDIA GeForce GTX 780. Rendering was performed in a 1024x1024 window.

- *Preprocessing:* Our edge-tracing implementation uses 12 CPU threads and filtering [84], the rest is computed in 1 thread. The edge-tracing takes 75% of the whole preprocessing time, the computation of bottlenecks 16%, probe intervals for free tori 6% and the sorting of vertices 2%.
- *Runtime:* The computation of Connolly surfaces for various probes uses only 1 CPU thread and its time includes data upload to GPU memory and rendering. The computation is slower for small probes (0.5Å) because a high number of surface elements needs to be processed.
- *Rendering:* Fps columns indicate the rendering performance when the probe radius is not changed and the surface does not have to be recomputed.

Table 8.2 and Figure 8.20 compare our method with others. BetaMol [14] also uses a Voronoi diagram computed in preprocessing and a triangular mesh for visualization, but computation and memory demands are high. VMD [42] uses Sanner's reduced surface method provided by MSMS library [104]. Lindow's ZIB-MolViewer implements the parallelized contour buildup algorithm [78]. Krone's MegaMol implements the reduced surface algorithm [68]. They both use ray casting for rendering, but they render the whole spheres and complete toroidal patches. Our method is slower because our shaders perform true clipping of all elements. More elements must be rendered because of this, e.g., 14% more spherical patches than the whole spheres and 36% more toroidal patches than complete toroidal patches (PDB ID: 1AON, probe 0.28Å). A performance bottleneck is in the rendering of toroidal patches. Our shaders must test all intersections of the ray with a torus or with a sphere. Lindow's shader is simpler because it locates only the first intersection. Also rasterization primitives are important. Lindow uses tight fitting rectangles

generated in a geometry shader (this is currently the fastest approach), Krone uses point sprites. We use point sprites as well except for spherical triangles, where we duplicate a bounding box by geometry instancing.

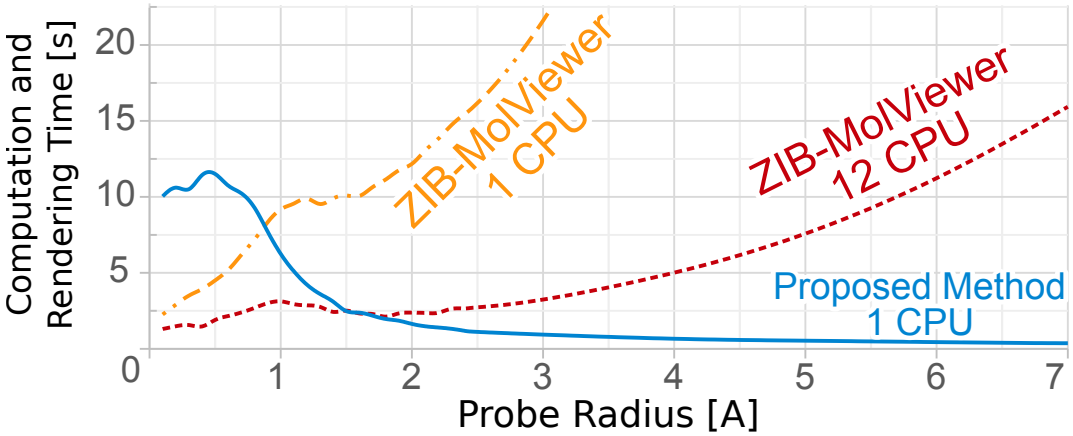


Figure 8.20: Our method compared with another. PDB ID: 4RHV (376 080 atoms). Our method gains more information thanks to volume exploration and it is able to classify the Connolly surface into cavities but requires preprocessing.

8.7 Chapter Conclusion

The proposed method computes the Connolly surface for any probe radius. Cavities are distinguished one from another and can be filtered by their properties. The surface can be observed from both sides and selected cavities as well as the outer surface can be hidden. When the probe radius is changed, the surface is quickly recomputed. Surface elements are found via volume exploration using the aw-Voronoi diagram. The diagram is also used for the computation of cavity properties. Surface rendering uses ray casting. All surface elements are truly clipped on GPU.

Table 8.1: The performance of the proposed method. PT is the preprocessing time in seconds. Next columns show the delay between the probe radius change and the moment when the Connolly surface was displayed. Frame rate is in fps columns.

				Computation and Rendering Time [s]					**Rendering Performance [fps]**				
PDB ID	**Atoms**	**VD vertices**	**PT [s]**	0.5Å	1.4Å	3Å	5Å	10Å	0.5Å	1.4Å	3Å	5Å	10Å
1AF6	10 050	67 774	0.9	0.28	0.14	0.06	0.05	0.04	24	45	63	67	71
1GKI	19 536	133 327	1.7	0.55	0.21	0.10	0.06	0.04	18	33	56	67	77
1VTZ	42 810	287 786	3.8	1.19	0.40	0.19	0.11	0.08	17	37	53	63	63
1AON	58 674	400 216	5.8	1.44	0.71	0.30	0.20	0.09	14	22	36	42	56
1JJ2	90 418	619 494	9.0	2.38	0.98	0.36	0.19	0.07	11	21	32	50	67
70S_RF2	298 820	2 043 328	32.5	7.73	2.39	1.42	0.63	0.28	5	8	13	22	37
4RHV	376 080	2 575 092	57.8	11.12	3.06	0.91	0.50	0.31	3	8	21	27	29

Table 8.2: Comparison of the proposed method with other approaches. Vis = visualization type, Tri = triangular mesh (low quality), RC = ray casting (high quality). Clip = surface clipping, full = all types (spherical triangles, toroidal and spherical patches), part = partial, only spherical triangles. Cav = surface of cavities. Filt = filtering of cavities (at least the ability to hide selected cavities).

PDB ID: 1AON	**Computation and Rendering Time [s]**					**Rendering Performance [fps]**					**Features**			
Method	0.5Å	1.4Å	3Å	5Å	10Å	0.5	1.4	3	5	10	**Vis**	**Clip**	**Cav**	**Filt**
BetaMol 0.92d	X	122.5	50.0	31.7	16.1	X	0.3	0.7	1	3	Tri	full	yes	yes
VMD + MSMS 2.6.1	34.5	10.0	21.5	12.9	N/A	20	33	58	60	N/A	Tri	full	X	X
ZIB-MolViewer 12 CPU	0.5	0.7	0.7	1.3	4.6	60	60	60	60	60	RC	part	yes	X
ZIB-MolViewer 1 CPU	0.9	1.7	3.2	7.4	32.0	60	60	60	60	60	RC	part	yes	X
MegaMol	2.9	2.1	2.0	2.1	3.3	24	39	45	45	40	RC	part	X	X
Proposed Method	1.4	0.7	0.3	0.2	0.1	14	22	36	42	56	RC	full	yes	yes

Chapter 9

All Cavities Detected by Continuously Shrinking Probe

The computation of cavities in protein models, where atoms are represented as balls, requires the probe radius to be specified before the computation. After the cavities for a particular probe radius are detected, they can be visually presented to domain experts, who can evaluate their importance. Sometimes, the experts may need to change the probe radius, because the set of cavities detected for a particular probe radius does not contain a cavity, which a human observer can clearly recognize in the model. As we have seen in Chapter 8, the change of the probe radius and the subsequent visualization of results can be done at interactive frame rates. The problem is that the visual interpretation of results is very subjective. In this chapter, a method is presented to overcome this problem by recording the development of cavities for a continuously shrinking probe. This method detects all possible cavities for all possible probes and organizes the results into a hierarchy, which can be further filtered by some user-defined criteria.

This problem has already been solved for balls of the same size [22], but only an approximate solution has been reported for balls of different sizes [115]. These solutions work on simplicial complexes derived from the dual structure of the ordinary Voronoi diagram of ball centers and use the mathematical concept of simplicial homology groups. These approaches can even measure the topological persistence of individual cavities and tunnels and hence their lifetime w.r.t. the probe radius [30]. Filtering short-living features out is known as a topological simplification. If the balls have different radii, it is more appropriate to use the aw-Voronoi diagram instead of the ordinary diagram or the power diagram. However, the dual structure is no longer a simplicial complex, so the previous approaches are not fully applicable in this case.

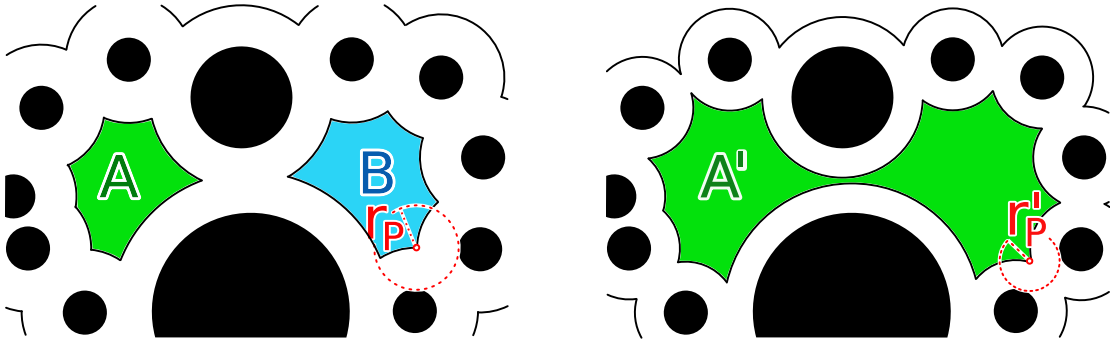
It is fair to note here that the previous approaches could be used with a little modification unless the aw-Voronoi diagram contains elliptic edges, which rarely occur in protein models. So the main contribution of the method presented in this chapter is the ability to handle elliptic edges and disconnected cases via bridges, and the secondary contribution is a hierarchical organization of cavities. The proposed method is not limited to protein models nor it is limited to simplicial complexes. It is applicable to general systems of balls. The full article can be found in [81].

The first section describes preliminaries regarding cavities and aw-distance. After that, a new concept of bridges is introduced to overcome difficulties with elliptic edges. Next, the method for finding all cavities is presented. The resulting tree graph captures the behavior of cavities during continuous shrinking of the probe radius. The next section describes how to transform the tree to

a more concise hierarchical representation. After the discussion about the time complexity, the results of experiments are listed. They were performed on protein models as well as on pseudo-random data and they indicate the real performance of the method, the dependency of the number of cavities on the probe radius in protein models, and filtered hierarchy of cavities.

9.1 Preliminaries

Cavities in this chapter are defined as the components of the space accessible to the probe center. Figure 9.1 illustrates the behavior of two cavities A and B for a shrinking probe radius. In this example, balls are expanded by the probe radius and the components of the complement of their union represent cavities. As the probe radius shrinks, the cavities will eventually merge into one.



(a) Two cavities A and B for the probe radius r_p

(b) One cavity A' for the probe radius r'_p

Figure 9.1: As the probe radius shrinks, both cavities will eventually merge into one

The key observation is that the creation of cavities occurs in Voronoi vertices and the merging of cavities occurs on Voronoi edges of the aw-Voronoi diagram [77, 105]. To detect these events, the proposed method works with the edges and vertices of the enriched aw-Voronoi diagram, which was discussed in Section 3.1 and illustrated in Figure 3.3. In this variant, Voronoi edges are split in the points of local extrema of the aw-distance function, so additional vertices (the split-points) represent the minima of the aw-distance function on Voronoi edges and the maxima of the aw-distance function on elliptic Voronoi edges. Furthermore, the edges in the enriched diagram are oriented in the direction of non-decreasing aw-distance. The edges and vertices of the enriched diagram will be represented as a directed graph $\vec{G}$.

In the following sections, the aw-distance of $x \in \mathbb{R}^3$ from the whole set S of sites (balls) is defined as $d_{aw}(x) = min_{s \in S}(d_{aw}(x,s))$. Note that for each k-dimensional Voronoi element x, the value of $d_{aw}(x)$ can be easily evaluated as $d_{aw}(x) = d_{aw}(x,s_i)$ for any $s_i \in S(x)$, where $S(x)$ is the set of $4-k$ sites (balls) that define the element x. A reasonable assumption is that $S(x)$ can be provided in time $O(1)$ for any element of $\vec{G}$, e.g., via additional pointers in the data structure.

9.2 Bridges

A new concept of bridges is proposed in this section. They solve problems with elliptic edges and disconnected cases. A *bridge* is an oriented connection between vertices of $\vec{G}$ with $d_{aw}(.)$

non-decreasing along the connection. For each vertex $v_m \in \vec{G}$ that represents the local maximum of $d_{aw}(.)$ on an elliptic Voronoi edge, the bridge target is computed by the following strategy:

1. Three balls defining the ellipse on which v_m lies are provided by an appropriate data structure. Among them, the pair $\{s_i, s_j\}$ is chosen, which defines the Voronoi face domain $F(s_i, s_j) = \{x \in \mathbb{R}^3 : d_{aw}(x, s_i) = d_{aw}(x, s_j)\}$ in which the ellipse carves a hole. It is the pair $\{s_i, s_j\}$ that maximizes $||c_i - c_j|| + r_i + r_j$ because the third ball lies in the convex hull of s_i and s_j [113].

2. A circle that lies on $F(s_i, s_j)$ and passes through v_m is defined as $\{x \in F(s_i, s_j) \ : \ d_{aw}(x, s_i) = d_{aw}(v_m, s_i)\}$. Examples can be seen in Figure 9.2 as the dotted circles for $v_m \in \{v_1, v_2, v_4\}$.

3. The intersection points of the circle and the geometry of edges of $\vec{G}$ are computed by a brute-force iteration over all edges that constitute the boundary of all Voronoi faces lying on $F(s_i, s_j)$. Each edge provides three balls that define the conic curve on which the edge lies. Thanks to the constraint that the aw-distance of each intersection point is $d_{aw}(v_m)$, the point can be computed from the three balls and the value of $d_{aw}(v_m)$ similarly as it was done for Voronoi vertices in Section 5.2.

4. Among the intersection points, the one that has the Euclidean distance from v_m minimal and non-zero is selected. The target vertex of the edge on which this point lies is the target point of the bridge. In Figure 9.2, $v_1 \to v_4$ and $v_2 \to v_3$ are such bridges.

5. If there is no intersection point except v_m as for $v_m = v_4$ in Figure 9.2, the vertices of $\vec{G}$ in $F(s_i, s_j)$ are searched for the vertex with minimal aw-distance greater than $d_{aw}(v_m, s_i)$, and this vertex will become the bridge target. In Figure 9.2, v_5 is such a target.

6. If such a vertex cannot be found, then any vertex at infinity is used as the bridge target.

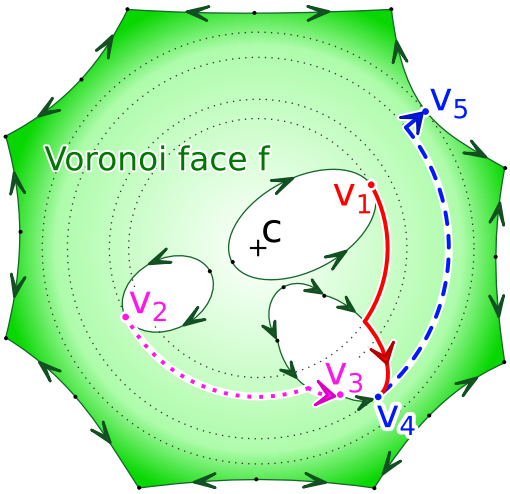


Figure 9.2: Bridges $v_1 \to v_4$, $v_2 \to v_3$ and $v_4 \to v_5$

For efficiency reasons, the required groups of edges and vertices of $\vec{G}$ are collected in advance, mapped to the corresponding pairs $\{s_i, s_j\}$, and retrieved later when the intersection points need to be computed. Each group contains all edges/vertices in the boundary of all faces that lie on $F(s_i, s_j)$. Thanks to this optimization, the boundary of all faces lying on $F(s_i, s_j)$ can be iterated in linear time w.r.t. the number of boundary elements.

Figure 9.3 illustrates bridges on a real example. The coordinates (x,y,z,r) of the balls are as follows: $(-2.5,7.51,0,1)$ $(2.5,7.51,0,1)$ $(0.01,0.01,80,78)$ $(0,0.01,-85,78)$ $(0.5,4,-10,8)$ $(100.01,0.01,0.01,80)$ $(0,104,0,80)$ $(-100,0,0,80)$ $(0,-100.01,0,80)$.

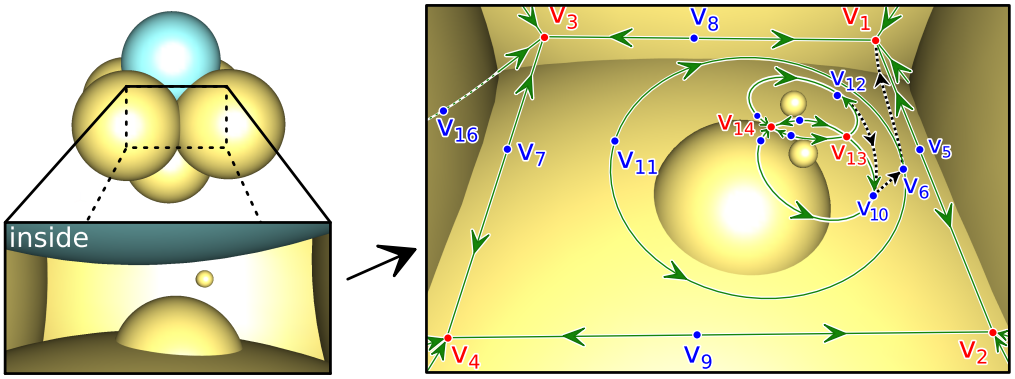


Figure 9.3: The graph $\vec{G}$ was constructed from a Voronoi 1-skeleton. The bridges are $v_{12} \to v_{10}$, $v_{10} \to v_6$ and $v_6 \to v_1$.

9.3 Finding All Cavities

The method presented in this section finds all possible cavities in a given set of balls for all possible probe radii. If a probe radius is added to the radii of all balls, the components of the space outside all these balls represent cavities. The method simulates a continuous shrinking of the probe radius. As the probe shrinks, new cavities are created or existing cavities merge. These events occur only on Voronoi vertices and edges. More precisely, in the local extrema of $d_{aw}(.)$ restricted to the Voronoi 1-skeleton. The method processes the diagram and constructs a tree graph of the detected events. Nodes in the tree are organized - this will be further discussed in Section 9.4. The method is formalized in Algorithm 7. Its description is given below.

First, the graph $\vec{G}$ is created from the Voronoi 1-skeleton by splitting Voronoi edges in their extreme points and orienting them from low values of $d_{aw}(.)$ to higher values. This was discussed in Section 5.3. The 1-skeleton and the split points can be computed, e.g., by AwVoronoi [15].

Next, vertices of the graph are sorted in the non-increasing order of $d_{aw}(.)$. This is necessary to detect cavities in the right order.

After that, bridge targets are computed for the vertices of $\vec{G}$ that lie in the maxima of elliptic Voronoi edges. Details to the computation of bridges are in Section 9.2.

Some initialization is performed on the line number 6. For the sake of simplicity, we will assume only one vertex at infinity v_0. The first cavity is detected at v_0 as if the probe radius was ∞. Groups of graph vertices will be maintained. Each group corresponds to exactly one cavity (it can be the result of several merges). The union-find system [19, 33] will be used to manage groups, similarly as it was done in the method of Delfinado and Edelsbrunner [22] that works with simplicial complexes. Each group will be realized by a tree stored in the array *parent*.

After the initialization, all vertices of $\vec{G}$ need to be processed. For each vertex v, the set N of its neighbors is obtained. These neighbors are the destination vertices of all outgoing edges of v. Several cases can occur. If the set N is empty, then $d_{aw}(v)$ is the local maximum of $d_{aw}(.)$ on the Voronoi 1-skeleton. The following two cases are distinguished from each other. If v is a Voronoi

Algorithm 7: FindCavities

Input : The set S of balls and the aw-Voronoi diagram $VD(S)$
Output: The root of the tree with cavities creation & merging

```
1  G⃗ ← the Voronoi graph from VD(S);
2  V ← vertices of G⃗, sorted: i ≤ j ⇒ d_aw(V[i]) ≥ d_aw(V[j]);
3  for v ∈ V do
4      if isEllipticEdgeMaximum(v) then
5          bridgeTarget[v] ← getTarget(v); // Sec. 9.2
6  v_0 ← V[0]; parent[v_0] ← v_0; result[v_0] ← root ← new Cavity(v_0); size[root] ← ∞;
7  for i ← 1; i < |V|; i ← i + 1 do
8      v ← V[i]; N ← neighbors of v in G⃗; // 0 ≤ |N| ≤ 2
9      if N == ∅ then // local maximum in v
10         if isVoronoiVertex(v) then
11             result[v] ← root ← new Cavity(v);
12             parent[v] ← v; size[root] ← d_aw(v);
13         else // elliptic edge maximum in v
14             parent[v] ← parent[bridgeTarget[v]];
15         end
16     else if isVoronoiVertex(v) then
17         parent[v] ← parent[any member of N];
18     else // local minimum in v ⇒ merge
19         R ← {find(n, parent) : n ∈ N}; // 1 ≤ |R| ≤ 2
20         parent[v] ← union(R, parent);
21         if |R| == 2 then
22             l ← result[R[0]]; r ← result[R[1]];
23             if size[l] < size[r] then
24                 swap(l, r); // Ensures size[l] ≥ size[r]
25             result[parent[v]] ← root ← new Merge(v, l, r);
26             size[root] ← size[l];
27         end
28     end
29 end
30 return root;
   // See [19, 33] for union-find.  Remaining functions are trivial
31 function union({a,b}, p) return rank(a) < rank(b) ? (p[a] ← b) : (p[b] ← a);
32 function find(n, p) return n == p[n] ? n : (p[n] ← find(p[n], p));
```

vertex (line 10), then a new cavity is created and a new group is started for v. Otherwise, v is necessarily the point of local maximum on an elliptic edge. Note that we already have bridges for all such points. In this case (line 13), v cannot trigger the creation of any cavity because any probe centered in v having a constant radius could escape along the bridge. Therefore, the group at v is simply inherited from the bridge target. Similarly, the group is also inherited if v is a Voronoi vertex with some neighbors (line 16). Thanks to the geometrical properties of aw-Voronoi diagrams, all the neighbors are already in the same group, so no merging is needed. Two cavities can merge only in points of local minima of $d_{aw}(.)$. These minima are on Voronoi

edges, not in Voronoi vertices. This case is handled in the end (line 18). If v is the point of local minimum, it has two outgoing edges, i.e., two neighbors. The groups at these neighbors are located and merged. If the groups were not the same, it means that two cavities just merged into one, so a new merge node is created. Children of the node are ordered for the purpose of building the hierarchy of cavities in Section 9.4.

The output of Algorithm 7 can also be used to count cavities in any interval of probe radii. Each cavity creation event increases the number of cavities by one and each merge event decreases the number by one. The events just need to be ordered by their creation time (the value of the probe radius at which the event occurred). The infinite cavity should not be counted by default, which means that both their initial and final counts in the interval $(-\infty, \infty)$ will be zero.

9.4 Cavities Hierarchy

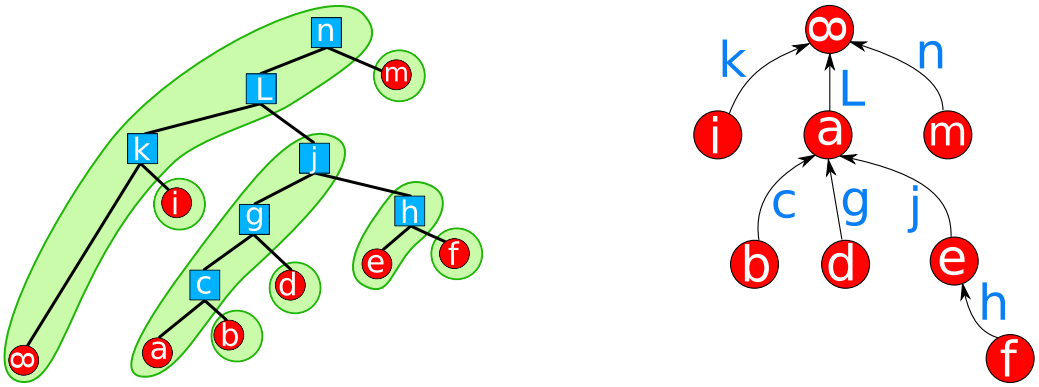
The output of Algorithm 7 is a tree graph, which describes how cavities were created and how they merged during the shrinking of the probe used for their detection. This graph is often very deep and not very informative, at least in molecular models. A hierarchy proposed in this section aims to provide a better representation.

The hierarchy is based on a simple rule: If two cavities merge, then the cavity that was detected by a smaller probe dies by merging into the cavity detected by a larger probe. In the case of a tie, the choice can be arbitrary. This way, the life of a cavity can be tracked from its creation, i.e., from a leaf node up to the root until the cavity dies by merging with another. This path is transformed to a single node in the hierarchy. Since the graph produced by Algorithm 7 is already organized, right sub-trees always merge into left sub-trees. The hierarchy is then constructed by Algorithm 8. An example of the transformation is shown in Figure 9.4.

Algorithm 8: Build Hierarchy

Input : The *root* of the tree with cavities creation & merging
Output: The hierarchy of cavities as a set of nodes and links

```
1  v_0 ← leftmostLeaf(root);
2  H ← {new Node(v_0)}; // The hierarchy of cavities.
3  stack.push(root);
4  while stack is not empty do
5      v ← stack.pop(stack);
6      if v is not a leaf then
7          lr ← leftmostLeaf(rightChild(v));
8          lv ← leftmostLeaf(v);
9          H ← H ∪ {new Node(lv)};
10         H ← H ∪ {new Link(lr, lv, v)}; // A link from the node of lr to lv.
11         stack.push(leftChild(v));
12         stack.push(rightChild(v));
13     end
14 end
15 return H;
   // push(),pop(),leftmostLeaf() and rightChild() are easy to implement
```



(a) The tree of cavities creation (leaves) and merging (inner nodes)

(b) The hierarchy

Figure 9.4: Creation of the hierarchy of cavities

The hierarchy is a tree rooted in the cavity of infinite size. Each child in the hierarchy was born after its parent and dies before its parent by merging into the parent. Therefore, we can consider parents to be representatives of their children. In this interpretation, the level of detail can be changed by hiding or showing children.

However, there is a catch. Not all children of a parent are so important. For instance, the parent can have a large number of tiny short-living children (less important) and a few of large long-living children (more important). Some sort of filtering based on cavity properties is necessary. Properties of cavities in the hierarchy may include the radius of the probe used to detect the cavity, the radius of the probe for which the cavity died by merging, the lifetime of the cavity as the difference between these two radii, the depth in the hierarchy tree, etc.

The decision about importance of cavities should be up to users by providing them a convenient way to define filters and specify thresholds. In the experimental implementation of the proposed methods, the hierarchy was exported to a graph and the Gephi software tool [5] was used for visualization and filtering.

We should mention here that a similar concept called topological persistence and simplification was proposed by Edelsbrunner et al. [30], but the lifetime of a cavity is measured differently. Perhaps the lifetime proposed by Yaffe et al. [115] is closer to our approach.

9.5 Time Complexity

The topology of the Voronoi diagram can be directly downloaded for most molecular structures from the VDRC website [110] or computed by the edge tracing algorithm [52, 87].

The edge tracing algorithm has the following time complexity. Let n be the number of balls. The diagram has $O(n^2)$ edges in the worst case [3, 10] and the edge tracing algorithm spends $O(n)$ time on each of them, which gives $O(n^3)$ in the worst case. The estimate is more optimistic on real data such as protein models: The diagram should have $O(n)$ edges. The algorithm should also spend much less time on every edge if spatial filters are used [13, 84, 93], ideally $O(1)$ on average, but it depends on the efficiency of the filters.

In Algorithm 7, the graph $\vec{G}$ is constructed from the Voronoi diagram in linear time w.r.t. the number of Voronoi vertices and edges. All Voronoi vertices will become $\vec{G}$ vertices. Each

Voronoi edge is analyzed for extreme points in $O(1)$. Voronoi edges are split in the extreme points and will become graph edges.

To construct a bridge, all edges in the boundary of all affected faces are examined. The worst case time complexity is quadratic w.r.t. the number of such edges. This is not very efficient in theory, but these cases are rare in practice and the number of such edges is often $O(1)$.

The rest of Algorithm 7 has its time complexity dominated by sorting the vertices of $\vec{G}$, i.e., $O(|V|\ log|V|)$. The merging of groups can be done with the union-find algorithm [19, 33], which has the amortized cost per operation $O(\alpha(n))$, where $\alpha(n)$ is the inverse of the fast growing Ackermann function; $\alpha(n)$ is often considered $O(1)$ for any practical value of n.

The time complexity of Algorithm 8 is linear if results of $leftmostLeaf(.)$ are precomputed during the construction of inner nodes of a merge graph.

9.6 Experiments and Results

The proposed methods were tested on sets of balls that produce disconnected aw-Voronoi diagrams and on models of molecular structures. The data of many molecular structures can be downloaded from public databases via their PDB ID.

Table 9.1 illustrates real running time on a few PDB files. The implementation is in Java and it was tested on CPU Intel i7 3.4 GHz (6x2 threads). $|V|$ is the number of Voronoi vertices. T_{VD} is the time taken by AwVoronoi [15] to compute the quasi-triangulation. $T_{\vec{G}}$ is the transformation time of the quasi-triangulation to the Voronoi graph $\vec{G}$, including bridges (only one in 1AON), i.e., lines 1-5 in Algorithm 7. T_{CAV} is the time of finding cavities in $\vec{G}$, i.e., lines 6-31. T_H is the time of hierarchy construction. A reasonable expectation is that the whole computation for small or medium-size proteins will take units of seconds and for large molecular structures tens of seconds. The last row in Table 9.1 is a special dataset of 10K points generated randomly in a cylinder with the radius 10 and the height 0.2 and two spheres with the radius 500, centered in $(0,\ 0,\ 500.1)$ and $(0,\ 0,\ -500.1)$. AwVoronoi [15] was inefficient on this dataset. The diagram is shown in Figure 9.6. It has 292 components (142 loop edges). 2118 bridges were created.

Table 9.1: Performance. K - thousands, s - seconds.

Data	**Atoms**	**|V|**	T_{VD}	$T_{\vec{G}}$	T_{CAV}	T_H
1GKI	20K	133K	1.9s	0.5s	0.2s	0.1s
1AON	60K	400K	5.6s	1.6s	0.4s	0.1s
1JJ2	90K	620K	8.0s	2.4s	0.7s	0.2s
70S_RF2	300K	2060K	31.0s	9.0s	2.0s	0.4s
Random	10K	35K	25.1s	2.9s	0.1s	0.1s

An interesting property is the dependency of the number of cavities on the probe radius. The results of the proposed method were checked against an alternative method, which uses weighted alpha shapes for cavities detection [73] and samples the probe radius interval by a constant step.

Figure 9.6 shows the results of one of such experiments. It is the set of balls from Figure 9.3. The configuration of balls was carefully designed to get a short-living cavity in the vertex v_{14}. Our method detected all cavities in this setting of balls without any problem. Bridges were used to propagate the information about cavities in disconnected cases. The results were checked by counting cavities with the method based on weighted alpha shapes, but the interval of probe radii had to be sampled densely to detect also the short-living cavity in the probe interval $(2, 3)$.

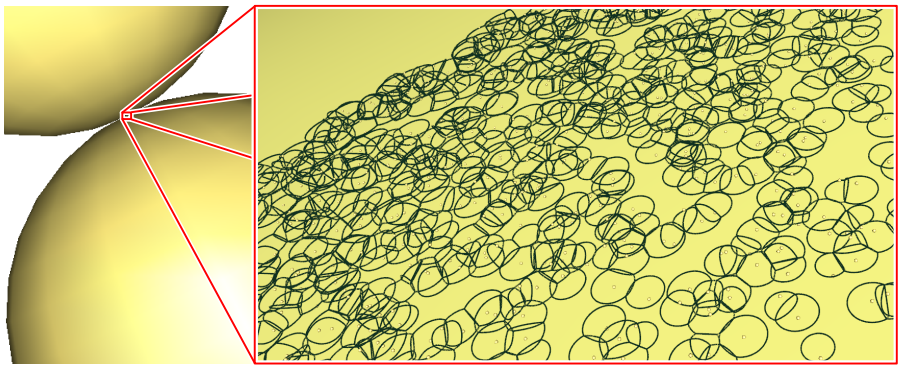


Figure 9.5: Random data with many elliptic edges

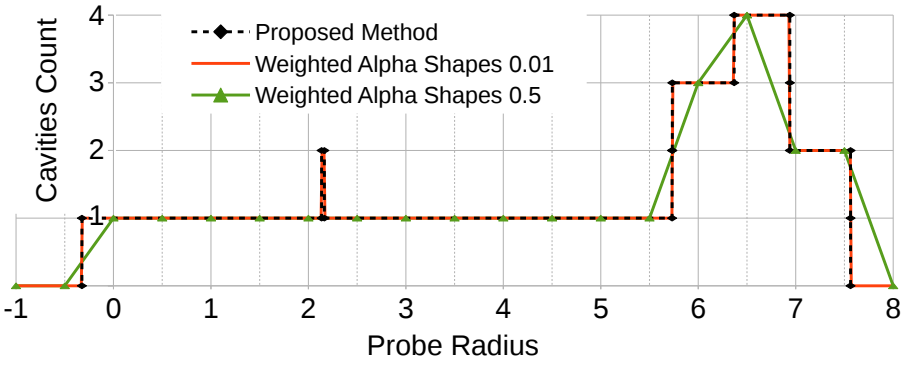


Figure 9.6: The proposed method detected all cavities in a set of balls that caused disconnected aw-Voronoi diagram. The method based on weighted alpha shapes missed one cavity when the probe interval was regularly sampled using the step size 0.5. The cavity was detected when the step was set to 0.01.

The proposed method has been compared to other approaches, which specialize on the detection of cavities for a fixed probe radius. Results are shown in Figure 9.7. With these approaches, the only way to get the graph of the number of cavities w.r.t. the probe radius is to sample a probe interval and get the result for each sample. The values obtained in these samples should be the same as in the case of the proposed method. BetaMol [14] uses the aw-Voronoi diagram of atoms to extract cavities [56]. Voroprot [92] also uses the aw-Voronoi diagram, but cavities are defined differently, so the results do not match. The results of an approach based on weighted alpha shapes [73] is also shown. With a dense probe axis sampling step 0.01, the alpha-shapes method has results very close to the proposed method. However, the whole alpha-shape needs to be fully recomputed and analyzed for each sample, since the topology of the underlying structure may change for different probe radii. All these recomputations may take quite a long time and, in the end, some cavities can still be missing.

More experiments on molecular structures revealed that the number of cavities depends on the percentage of helix formations. A small probe can be trapped by atoms in a helix formation and create a micro-cavity. The periodic nature of helices results in a high number of such micro-cavities. Normal cavities are shown in Figure 9.8(a) whereas micro-cavities in Figure 9.8(b). Atom balls are hidden in these figures, but the backbone of the molecular structure is shown to highlight helix formations. Cavities are visualized by their Connolly surface and distinguished by different colors.

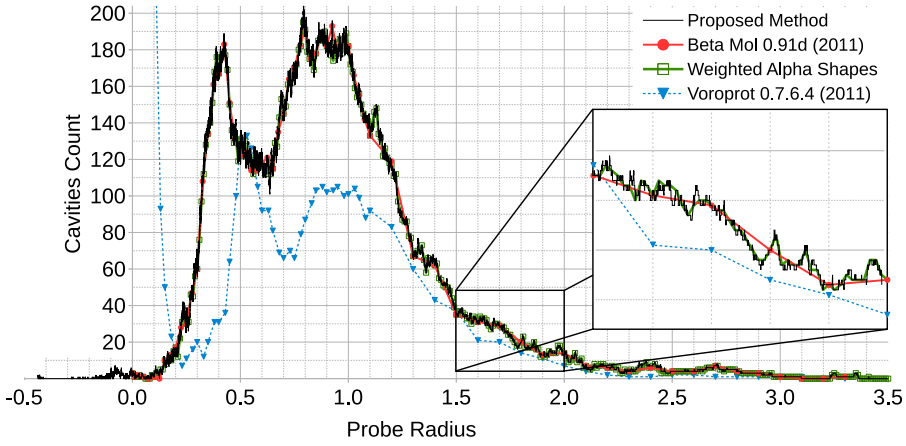
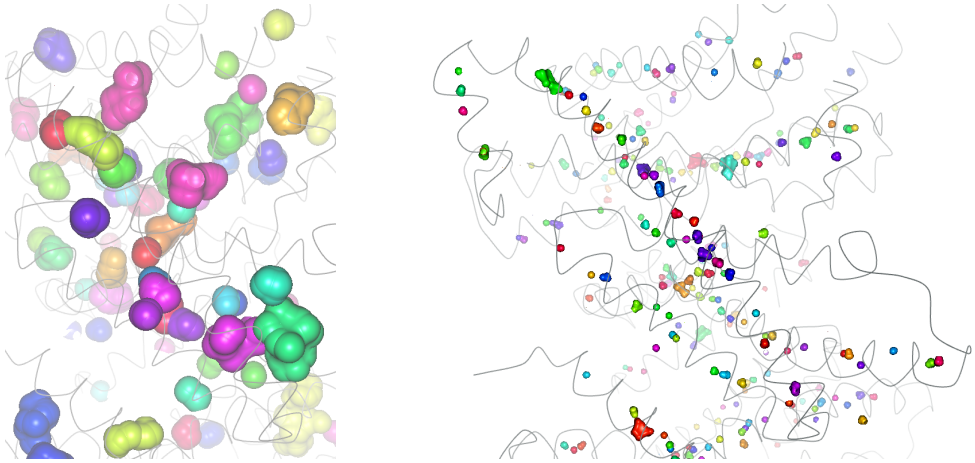


Figure 9.7: Comparison of the proposed method with BetaMol [14], Voroprot [92] and the method based on alpha-shapes [73]. Voroprot gives different results because it defines cavities differently. PDB ID: 4P1T (5075 atoms).

Figure 9.9 shows the difference between molecular structures having a low percentage of helices and structures having a high percentage of helices. The peek around the probe radius 0.35 in Figure 9.9(b) is caused by helix formations.

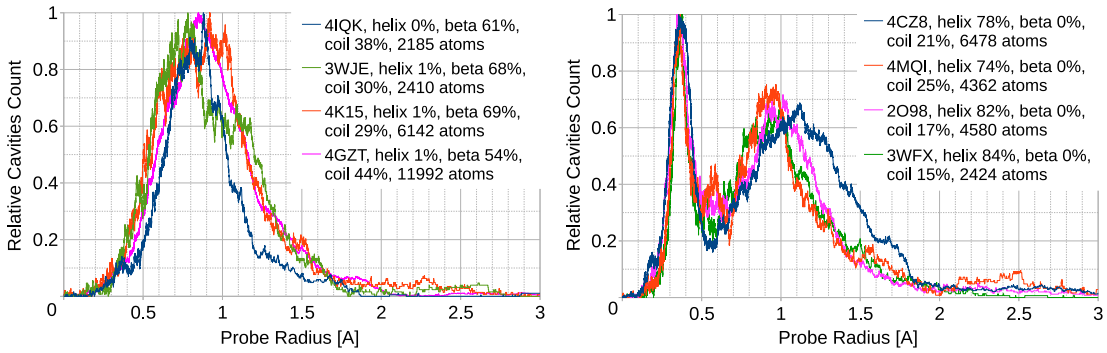


(a) Cavities for $r_P = 1.4$Å (b) Micro cavities for $r_P = 0.35$

Figure 9.8: Cavities detected and rendered by CAVER Analyst [67]. PDB ID: 4CZ8.

The rest of this section demonstrates the advantages of the proposed hierarchy of cavities. The following examples were computed by the implementation of the proposed method, converted to a directed graph and rendered by the Gephi software tool [5]. Gephi allows to bind the visual appearance of nodes to their attributes, filter the graph interactively, define layouts, etc. In the following graphs, nodes are labeled by the birth attribute (the radius of the largest probe that detected the cavity) and edges are labeled by the merge attribute (the radius of the probe for which the cavity merged into its parent). Numbers are rounded to 3 decimal places.

The first structure on which we will demonstrate the hierarchy of cavities is a large chaperonin complex with PDB ID: 1AON (58 674 atoms). The hierarchy has 69 930 nodes and maximal depth 6. However, 99.45% of all nodes have their depth ≤ 3, and 95.81% have their depth ≤ 2,



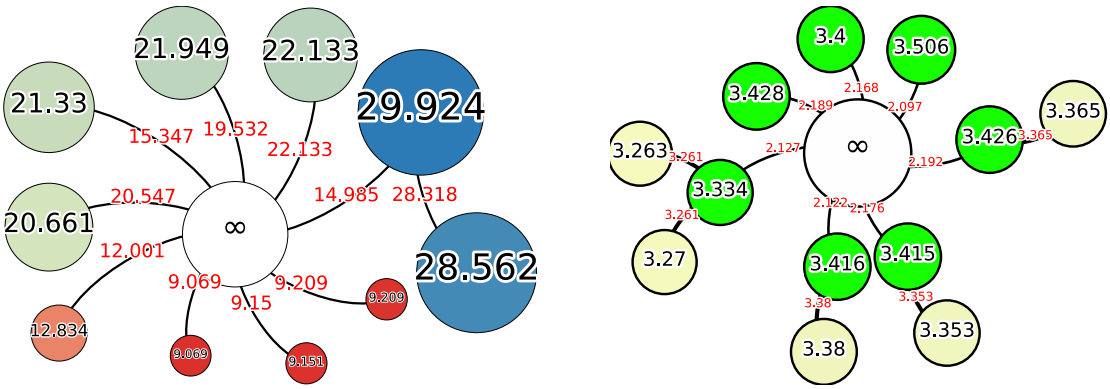
(a) Low percentage of helices

(b) High percentage of helices

Figure 9.9: Cavities count in molecular structures

and 75.92% have their depth ≤ 1. We are usually interested in nodes of depth 1 because they represent the cavities that will directly merge with the infinite cavity. The number of nodes on this level is still too high. Some filtering is needed.

Figure 9.10(a) shows the hierarchy of cavities filtered to keep only the nodes born ≥ 9Å. This filter reduces the hierarchy to 0.02% of the total number of nodes. This hierarchy graph tells us that 10 cavities were born in the probe interval $[9,\infty)$, but only 9 of them merge directly with the cavity representing infinity. The four large cavities born at 20.661, 21.33, 21.949 and 29.924 are shown in Figure 9.11. They are visualized by their Connolly surface and distinguished from each other by different colors. The cavity 22.133 is not shown in Figure 9.11. It would be visible if the probe radius in CAVER Analyst was set to, e.g., $r_P = 22.1332$. Figure 9.10(b) shows the hierarchy for 1AON filtered to keep only the cavities born in the interval $[3\text{Å}, 5\text{Å}]$ and with a lifetime ≥ 1.11Å. The seven nodes on the level 1 of the hierarchy correspond to the cavities where ADP molecules are located (the PDB model has ADP molecules in these places). If the lifetime filter was not used, there would be many more cavities.



(a) Only the cavities born ≥ 9Å

(b) Only the cavities with a strong lifetime, born between $2-5$Å

Figure 9.10: PDB ID: 1AON. The filtered hierarchy of cavities. Processed with Gephi [5].

The next structure on which we will demonstrate the hierarchy of cavities is an enzyme with PDB ID: 1CQW (2 358 atoms). The hierarchy has 2 947 nodes. The maximum depth is 6, but 97.73% of all nodes have their depth ≤ 3, and 92.53% have their depth ≤ 2, and 70.65% have their depth ≤ 1. Figure 9.12 illustrates how complicated the situation is without filtering.

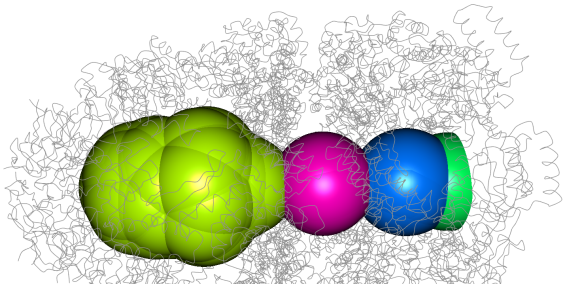


Figure 9.11: PDB ID: 1AON. Four large cavities, $r_p = 20.66$Å. Rendered by CAVER Analyst

Figure 9.13 shows the hierarchy filtered by a composite filter. The filter restricts the cavities directly connected with ∞ to the ones that were born ≥ 1.4Å, died ≥ 1.0Å, and lived for ≥ 0.2Å . In other words, we are interested in cavities, which can accommodate at least one water molecule, are not entirely closed, and are relatively stable when the probe expands or shrinks a little bit. Only 0.98% nodes survived. The node 2.31 corresponds to the active site of the enzyme. The cavity is born at 2.31Å and dies at 1.477Å. If we had used a standard approach with the fixed probe radius 1.4Å, we would have missed the cavity. Figure 9.14 shows the Connolly surface representation of the cavity node and its sub-nodes in the hierarchy.

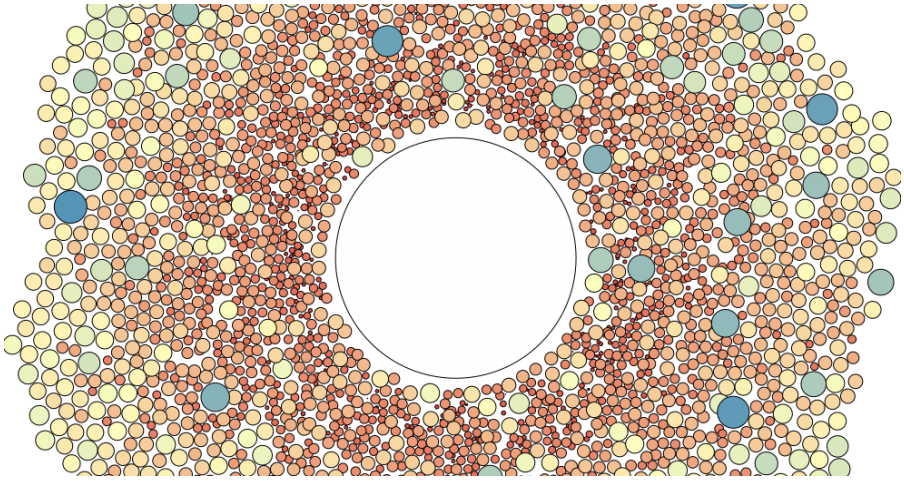


Figure 9.12: PDB ID: 1CQW. The hierarchy without filtering.

9.7 Chapter Conclusion

The number of cavities among balls depends on the radius of a probe used to inspect the empty space. The proposed method simulates a continuous probe shrinking process, during which the empty space reachable by the probe is explored, and the behavior of cavities is recorded. The probe center moves along aw-Voronoi edges. Newly proposed bridges overcome problems with disconnected cases. The proposed hierarchy of cavities provides a better insight into the structure of the empty space among the balls than traditional approaches, which require the probe radius to be specified prior to cavities detection. The intended application is the analysis of bio-molecules, but the method is not limited to molecular data.

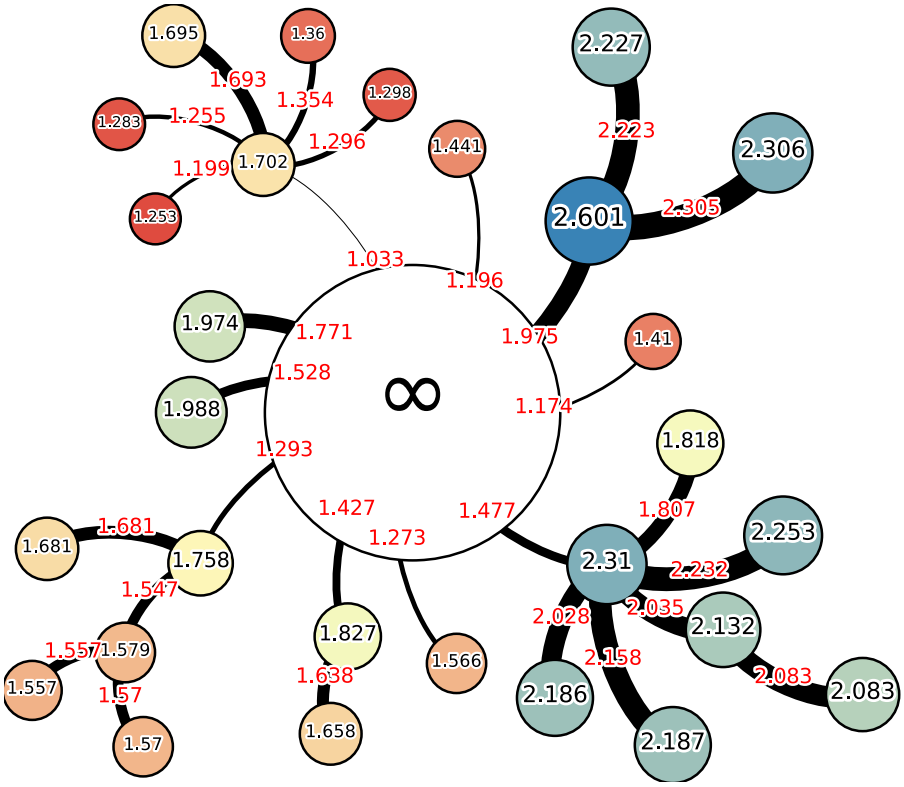


Figure 9.13: PDB ID: 1CQW. The filtered hierarchy of cavities. Nodes are labeled with the birth radius (the greatest sphere that fits in the cavity), edges are labeled with the merge radius. The node 2.31 corresponds to the active site. The cavity ∞ corresponds to the space outside the molecular structure. Processed with Gephi [5].

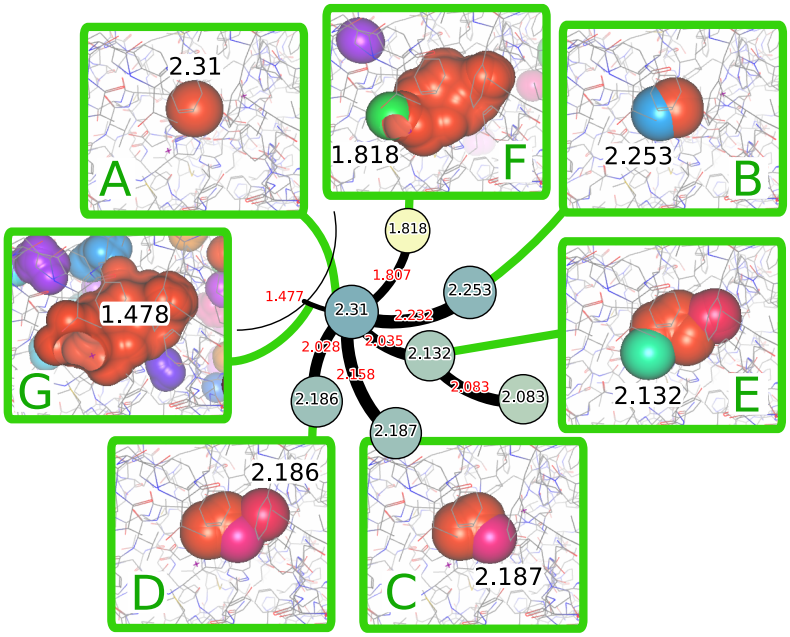


Figure 9.14: The Connolly surface of the cavities corresponding to one branch of the hierarchy. The most important cavity here is the one with the label 2.31. The sequence of windows $A, \ldots, F$ visualizes the lifetime of the cavity – the sub-nodes and how they merge with the main cavity.

Chapter 10

Future Work

One of the possible directions of future research is the application of aw-Voronoi diagrams to dynamic protein models. This has been only partially addressed by Lindow et al. [76]. The problem is that the computational and memory demands are still high. For instance if the aw-Voronoi diagrams need to be computed for all snapshots that describe the movement of atoms in a dynamic protein, the computation may take days of CPU time and at least tens GB of memory. Any substantial improvement in this area would be beneficial.

One of the possibilities, which needs to be explored, is a new concept of *hybrid Voronoi diagram* proposed by the author of this thesis. It combines aw-Voronoi diagrams, which are non-linear, with power diagrams, which are linear. The idea is illustrated in Figure 10.1, where two power diagrams and one aw-Voronoi diagram are combined into one hybrid diagram. The aw-Voronoi part of the hybrid diagram can still be used for spatial analysis, but the probe radius is now limited by an interval. The author expects that this new combination of two different diagram types will reduce the time needed for the construction of the whole aw-Voronoi diagram and it will still offer a reasonable structure for spatial analysis.

If the information provided by the power diagrams is not needed, components of the power diagrams can be collapsed into vertices or even removed as illustrated in Figure 10.2. Preliminary experiments indicate that such a removal could save a huge amount of memory. An example can be seen in Figure 10.3. If we were interested in performing some spatial analysis only for probe radii ≥ 1Å, then we could save about 65% of memory.

Storing aw-Voronoi diagrams for dynamic protein models is another challenge. Current methods store the dual quasi-triangulation in text files. Kim et al. proposed a text file format [55] similar to the PDB file format. The software tool AwVoronoi [15] stores the data in XML files. Text files can be compressed, but the compression ratio is still bad and reading the data requires additional parsing. One of the possible alternatives for dynamic proteins could be the NetCDF format[1]. It is a general-purpose data format for vector data. It is heavily used in meteorology and it is also used by GROMACS to store dynamic protein models.

[1] `http://www.unidata.ucar.edu/software/netcdf`

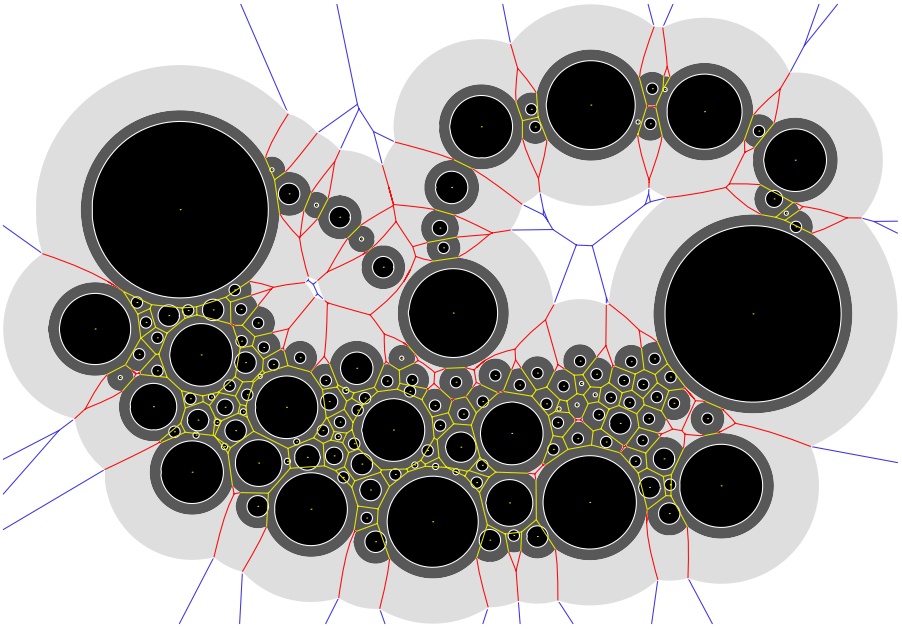


Figure 10.1: A hybrid Voronoi diagram combines two power diagrams (blue and yellow) with the aw-Voronoi diagram (red). Power diagrams are computed on enlarged atom spheres.

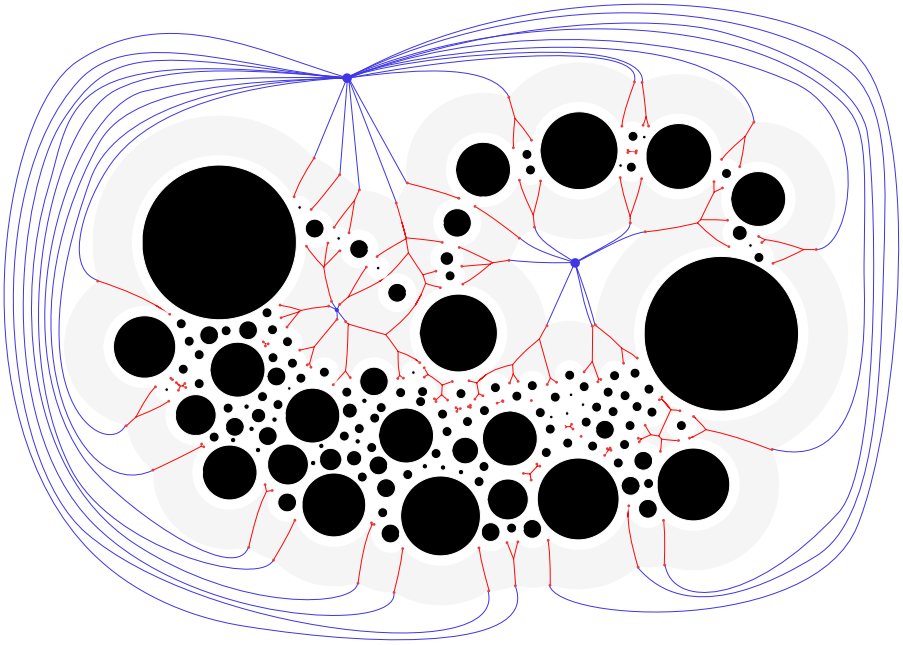


Figure 10.2: The reduction of a hybrid Voronoi diagram. The components of one power diagram (blue) are reduced to vertices, and the other power diagram (which was yellow) is removed. This structure can still be used for a spatial analysis for limited probe radii.

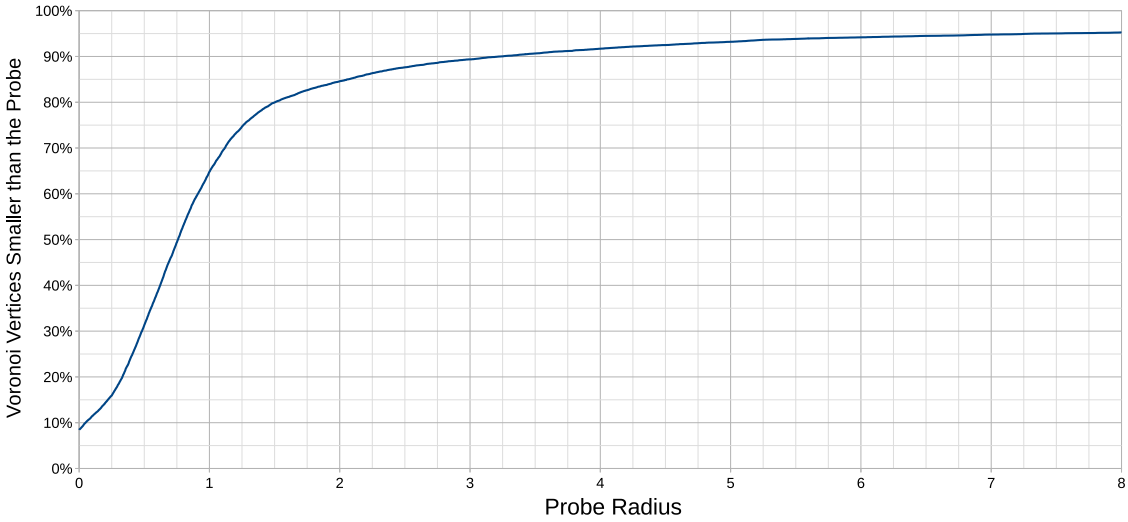


Figure 10.3: The percentage of Voronoi vertices, which have their radii smaller than the probe radius. PDB ID: 1CQW.

Another issue, which still needs to be addressed, is the support of insertion and removal of sites. The edge-tracing algorithm [52, 85, 87] constructs the aw-Voronoi diagram from scratch. The removal of sites could be accomplished by removing the affected parts from the diagram and restarting the edge-tracing algorithm on the incomplete parts of the diagram. The insertion of sites will be more complicated. We should mention here that a theoretical insertion algorithm has been described by Boissonnat, Karavelas and Delage [9, 10] but still no implementation is available. Their algorithm is based on different principles than tracing Voronoi edges.

Another challenge is the spatial analysis of proteins with respect to more complex probes. The probe is no longer a simple sphere but a union of spheres instead, and the task is to find collision-free paths, along which the probe could escape from a deeply buried cavity. Such cases can be solved by approaches from robotics, which explore the space of possible configuration via random sampling. The problem is that the number of samples can be extremely high, especially when the passage is very narrow. Furthermore, no sampling-based approach can verify the non-existence of a path because uncountably many samples would be needed. The idea for future research is to use the aw-Voronoi diagram to generate promising paths for a spherical approximation of the complex probe and generate more samples allong these paths. This might improve the reliability and the performance of sampling-based approaches.

Chapter 11

Thesis Conclusion

This thesis was about the application of computational geometry concepts to the analysis of protein models. The fundamental structure for the analysis is the additively weighted Voronoi diagram, which provides all the information that is needed to navigate a spherical probe through the protein model without a collision. Aw-Voronoi diagrams have been used for decades to describe spatial relations among atoms in proteins, but there was enough space for contributions.

One of the contributions is an improved acceleration technique for the construction of these diagrams. Experiments indicate that this method is considerably faster than the currently best approach. Saving a few seconds per one protein model may seem to be not much at first sight. The difference will be more apparent in dynamic protein models having tens of thousands of snapshots. Saving a few seconds per snapshot will then turn into saving a few hours or even days of computation.

Perhaps the most important contribution is the extension of the edge-tracing algorithm, which enables the algorithm to handle disconnected Voronoi diagrams. This extension makes the algorithm applicable to more general input data.

A practical contribution is an interactive method for the detection and visualization of cavities in proteins. These cavities are detected by a spherical probe. The proposed method allows to change the probe radius interactively. Cavities are detected and visualized by their Connolly surface. The aw-Voronoi diagram is also used to compute cavity properties, which can be used, e.g., to asses the importance of cavities.

At last, the method for the detection of all cavities discoverable by all possible probes and a hierarchical organization of cavities was presented. This approach utilizes maximum information, which aw-Voronoi diagrams can possibly provide. Nevertheless, additional bridges were needed to overcome elliptic edges and disconnected cases. The major advantage of this approach is that the probe radius does not have to be specified prior to cavities computation.

All proposed methods except the detection of cavities via continuously shrinking probe have been implemented into the software tool CAVER Analyst, in the context of a long-term cooperation with Masaryk University in Brno.

Appendix A

Activities

A.1 Publications on International Conferences

[83] M. Manak and I. Kolingerova. Inverted ball as a boundary-constraint for Voronoi diagrams of 3D spheres. In *25th European Workshop on Computational Geometry*, EuroCG, pages 289–292, 2009. URL: `http://2009.eurocg.org`.

[84] M. Manak and I. Kolingerova. Fast discovery of Voronoi vertices in the construction of Voronoi diagram of 3D balls. In M. A. Mostafavi, editor, *Seventh International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'10, pages 95–104, Laval University, Quebec, QC, Canada, 2010. IEEE Computer Society.
DOI: 10.1109/ISVD.2010.22.

[116] M. Zemek, M. Manak, and I. Kolingerova. Advanced space filtering for the construction of 3D additively weighted Voronoi diagram. In *The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences*, ADVCOMP 2011, pages 37–43, Lisbon, Portugal, 2011. IARIA.

A.2 Publications in Impacted Journals

[67] B. Kozlikova, E. Sebestova, V. Sustr, J. Brezovsky, O. Strnad, L. Daniel, D. Bednar, A. Pavelka, M. Manak, M. Bezdeka, P. Benes, M. Kotry, A. W. Gora, J. Damborsky, and J. Sochor. CAVER Analyst 1.0: graphic tool for interactive visualization and analysis of tunnels and channels in protein structures. *Bioinformatics*, 30(18):2684–2685, 2014.
DOI: 10.1093/bioinformatics/btu364. IF 5.766.

[85] M. Manak and I. Kolingerova. Extension of the edge tracing algorithm to disconnected Voronoi skeletons. *Information Processing Letters*, 116(2):85–92, 2016.
DOI: 10.1016/j.ipl.2015.09.017. IF 0.605.

[82] M. Manak, L. Jirkovsky, and I. Kolingerova. Interactive analysis of Connolly surfaces for various probes. *Computer Graphics Forum*, 2016. DOI: 10.1111/cgf.12870. IF 1.542.

[81] M. Manak. Exploration of empty space among spherical obstacles via additively weighted Voronoi diagram. *Computer Graphics Forum*, 35(5):249–258, 2016.
DOI: 10.1111/cgf.12980. IF 1.542.

A.3 Participation in Scientific Projects

- 2009–2011: Triangulated Models for Haptic and Virtual Reality. Czech Science Foundation (GACR), GA201/09/0097. Project leader I. Kolingerová.
- 2010–2012: Analysis and Visualization of Protein Structures. Czech Science Foundation (GACR), GAP202/10/1435 (2010-2012). Project leader J. Sochor.
- 2013–2015: SGS – Advanced Computing and Information Systems. University of West Bohemia (UWB), SGS-2013-029. Project leader J. Šafařík.
- 2015–2015: NTIS – New Technologies for the Information Society. European Regional Development Fund (ERDF), CZ.1.05/1.1.00/02.0090.
- 2015–2016: PUNTIS. Czech Ministry of Education, Youth and Sports (MEYS), LO1506.
- 2016–2016: SGS – Advanced Graphics and Computational Systems. University of West Bohemia (UWB), SGS-2016-013. Project leader I. Kolingerová.

Bibliography

[1] A. V. Anikeenko, M. G. Alinchenko, V. P. Voloshin, N. N. Medvedev, M. L. Gavrilova, and P. Jedlovszky. Implementation of the Voronoi-Delaunay method for analysis of intermolecular voids. In *Computational Science and Its Applications – ICCSA 2004*, volume 3045 of *Lecture Notes in Computer Science*, pages 217–226. Springer Berlin / Heidelberg, 2004.

[2] F. Anton, D. Mioc, and M. Santos. Exact computation of the topology and geometric invariants of the Voronoi diagram of spheres in 3D. *Journal of Computer Science and Technology*, 28(2):255–266, 2013.

[3] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM J. Comput.*, 16(1):78–96, 1987.

[4] D. Avis, B. Bhattacharya, and H. Imai. Computing the volume of the union of spheres. *The Visual Computer*, 3(6):323–328, 1988.

[5] M. Bastian, S. Heymann, and M. Jacomy. Gephi: An open source software for exploring and manipulating networks. In *International AAAI Conference on Weblogs and Social Media*, pages 361–362, San Jose, California, 2009. AAAI Press.

[6] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[7] J. D. Bernal. A geometrical approach to the structure of liquids. *Nature*, 183(4655):141–147, January 1959.

[8] J. D. Bernal and J. L. Finney. Random close-packed hard-sphere model. II. geometry of random packing of hard spheres. *Discussions of the Faraday Society*, 43:62, 1967.

[9] J.-D. Boissonnat and C. Delage. Convex hull and Voronoi diagram of additively weighted points. In G. Brodal and S. Leonardi, editors, *Proceedings of the 13th Annual European Conference on Algorithms*, volume 3669 of *ESA'05*, pages 367–378, Palma de Mallorca, Spain, 2005. Springer-Verlag, Berlin, Heidelberg.

[10] J.-D. Boissonnat and M. I. Karavelas. On the combinatorial complexity of Euclidean Voronoi cells and convex hulls of d-dimensional spheres. In *Proceedings of the fourteenth annual ACM-SIAM symposium on discrete algorithms*, SODA'03, pages 305–312, Baltimore, Maryland, USA, 2003. ACM and SIAM.

[11] A. Bondi. Van der Waals Volumes and Radii. *Journal of Physical Chemistry*, 68(3):441–451, March 1964.

[12] F. Cazals, H. Kanhere, and S. Loriot. Computing the volume of a union of balls: A certified algorithm. *ACM Transactions on Mathematical Software*, 38(1):3:1–3:20, 2011.

[13] Y. Cho, D. Kim, H.-C. Lee, J. Y. Park, and D.-S. Kim. Reduction of the search space in the edge-tracing algorithm for the Voronoi diagram of 3D balls. In M. Gavrilova, O. Gervasi, V. Kumar, C. Tan, D. Taniar, A. Laganá, Y. Mun, and H. Choo, editors, *Computational Science and Its Applications – ICCSA 2006, Proceedings, Part I*, volume 3980 of *Lecture Notes in Computer Science*, pages 111–120, Glasgow, UK, 2006. Springer Berlin Heidelberg.

[14] Y. Cho, J.-K. Kim, J. Ryu, C.-I. Won, C.-M. Kim, D. Kim, and D.-S. Kim. Betamol: A molecular modeling, analysis and visualization software based on the beta-complex and the quasi-triangulation. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, 6(3):389–403, 2012.

[15] Computational library awVoronoi, `http://awvoronoi.sf.net`. Accessed on June 26, 2016.

[16] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16(5):548–558, 1983.

[17] M. L. Connolly. Solvent-accessible surfaces of proteins and nucleic acids. *Science*, 221(4612):709–713, 1983.

[18] M. L. Connolly. Molecular surface triangulation. *Journal of Applied Crystallography*, 18(6):499–505, Dec 1985.

[19] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (Third edition)*, chapter 21: Data structures for Disjoint Sets, pages 561–585. MIT Press, 2009.

[20] R. de Toledo, B. Levy, and J.-C. Paul. Iterative methods for visualization of implicit surfaces on GPU. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, N. Paragios, S.-M. Tanveer, T. Ju, Z. Liu, S. Coquillart, C. Cruz-Neira, T. Müller, and T. Malzbender, editors, *Advances in Visual Computing – ISVC 2007, Proceedings, Part I*, volume 4841 of *Lecture Notes in Computer Science*, pages 598–609, Lake Tahoe, NV, 2007. Springer Berlin Heidelberg.

[21] B. Delaunay. Sur la sphère vide. A la mémoire de Georges Voronoï. *Bulletin de l'Académie des Sciences de l'URSS. Classe des sciences mathématiques et naturelles*, 1934(6):793–800, 1934.

[22] C. J. A. Delfinado and H. Edelsbrunner. An incremental algorithm for Betti numbers of simplicial complexes. In *Proceedings of the ninth annual symposium on Computational Geometry*, SCG'93, pages 232–239, San Diego, California, USA, 1993. ACM.

[23] J. Dundas, Z. Ouyang, J. Tseng, A. Binkowski, Y. Turpaz, and J. Liang. CASTp: computed atlas of surface topography of proteins with structural and topographical mapping of functionally annotated residues. *Nucleic Acids Research*, 34(suppl 2):W116–W118, 2006.

[24] H. Edelsbrunner. *Weighted alpha shapes*. Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[25] H. Edelsbrunner. Weighted alpha shapes doc. electronique. Technical Report UIUCDCS-R-92-1760, University of Illinois at Urbana-Champaign. Urbana (IL US), 1992.

[26] H. Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13(1):415–440, 1995.

[27] H. Edelsbrunner, M. Facello, P. Fu, and J. Liang. Measuring proteins and voids in proteins. In *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, volume 5 of *HICSS'95*, pages 256–264, Wailea, HI, 1995. IEEE Computer Society.

[28] H. Edelsbrunner and P. Fu. Measuring space filling diagrams and voids. Technical Report UIUC-BI-MB-94-01, Beckman Inst., Molecular Biophysics Group, Univ. Illinois at Urbana-Champaign, 1994.

[29] H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983.

[30] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 28(4):511–533, 2002.

[31] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. In *Proceedings of the 1992 Workshop on Volume Visualization*, VVS'92, pages 75–82, Boston, MA, USA, 1992. ACM.

[32] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, Jan. 1994.

[33] B. A. Galler and M. J. Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, May 1964.

[34] M. Gavrilova. *Proximity and Applications in General Metrics*. PhD thesis, University of Calgary, Calgary, Alta., Canada, Canada, 1999. AAINQ38468.

[35] M. L. Gavrilova. A reliable algorithm for computing the generalized Voronoi diagram for a set of spheres in the Euclidean d-dimensional space. In *Proceedings of the 14th Canadian Conference on Computational Geometry*, CCCG'02, pages 82–87, University of Lethbridge, Alberta, Canada, 2002.

[36] M. L. Gavrilova. An explicit solution for computing the Euclidean d-dimensional Voronoi diagram of spheres in a floating-point arithmetic. In V. Kumar, M. L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, editors, *Computational Science and Its Applications – ICCSA 2003*, volume 2669 of *Lecture Notes in Computer Science*, pages 827–835. Springer Berlin Heidelberg, 2003.

[37] B. J. Gellatly and J. L. Finney. Calculation of protein volumes: An alternative to the voronoi procedure. *Journal of Molecular Biology*, 161(2):305–322, 1982.

[38] J. E. Gentle, W. Härdle, and Y. Mori. *Handbook of Computational Statistics: Concepts and Methods*. Springer-Verlag, Berlin, Heidelberg, 2004.

[39] M. Gerstein, J. Tsai, and M. Levitt. The volume of atoms on the protein surface: Calculated from simulation, using Voronoi polyhedra. *Journal of Molecular Biology*, 249(5):955–966, 1995.

[40] A. Goede, R. Preissner, and C. Frömmel. Voronoi cell: New method for allocation of space among atoms: Elimination of avoidable errors in calculation of atomic volume and density. *Journal of Computational Chemistry*, 18(9):1113–1123, 1997.

[41] J. Greer and B. L. Bush. Macromolecular shape and surface maps by solvent exclusion. *Proceedings of the National Academy of Sciences of the United States of America*, 75(1):303—307, 1978.

[42] W. Humphrey, A. Dalke, and K. Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[43] H. Imai, M. Iri, and K. Murota. Voronoi diagram in the Laguerre geometry and its applications. *SIAM Journal on Computing*, 14(1):93–105, 1985.

[44] W. Kahan. Pracniques: Further remarks on reducing truncation errors. *Commun. ACM*, 8(1):40, 1965.

[45] D. Kauker, M. Krone, A. Panagiotidis, G. Reina, and T. Ertl. Rendering molecular surfaces using order-independent transparency. In F. Marton and K. Moreland, editors, *Eurographics Symposium on Parallel Graphics and Visualization*, volume 13 of *EGPGV'13*, pages 33–40, Girona, Catalonia, Spain, 2013. The Eurographics Association.

[46] D. Kim, C.-H. Cho, Y. Cho, J. Ryu, J. Bhak, and D.-S. Kim. Pocket extraction on proteins via the Voronoi diagram of spheres. *Journal of Molecular Graphics and Modelling*, 26(7):1104 – 1112, 2008.

[47] D. Kim, Y. Cho, and D.-S. Kim. Anomaly occurrences in quasi-triangulations and beta-complexes. In M. L. Gavrilova and K. Vyatkina, editors, *Proceedings of the 2013 Tenth International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'13, pages 82–88, Saint Petersburg, Russia, 2013. IEEE Computer Society.

[48] D. Kim, Y. Cho, J.-K. Kim, Y.-S. Lee, and D.-S. Kim. How similar are quasi, regular, and Delaunay triangulations in R3? In B. Murgante, S. Misra, A. Rocha, C. Torre, J. Rocha, M. Falcão, D. Taniar, B. Apduhan, and O. Gervasi, editors, *Computational Science and Its Applications – ICCSA 2014*, volume 8580 of *Lecture Notes in Computer Science*, pages 381–393. Springer International Publishing, 2014.

[49] D. Kim, J. Ryu, H. Shin, and Y. Cho. Beta-decomposition for the volume and area of the union of three-dimensional balls and their offsets. *Journal of Computational Chemistry*, 33(13):1252–1273, 2012.

[50] D.-S. Kim. A single beta-complex solves all geometry problems in a molecule. In *Proceedings of the 2009 Sixth International Symposium on Voronoi Diagrams*, ISVD'09, pages 254–260, Copenhagen, Denmark, 2009. IEEE Computer Society.

[51] D.-S. Kim, Y. Cho, and D. Kim. Edge-tracing algorithm for Euclidean Voronoi diagram of 3D spheres. In *Proceedings of the 16th Canadian Conference on Computational Geometry*, CCCG'04, pages 176–179, Concordia University, Montréal, Quebec, Canada, 2004.

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

[53] D.-S. Kim, Y. Cho, D. Kim, S. Kim, and J. Bhak. Euclidean Voronoi diagram of 3D spheres and applications to protein structure analysis. In *Proceedings of the 2004 First International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'04, pages 137–144, University of Tokyo, Hongo, Tokyo, Japan, 2004.

[54] D.-S. Kim, Y. Cho, D. Kim, S. Kim, J. Bhak, and S.-H. Lee. Euclidean Voronoi diagrams of 3D spheres and applications to protein structure analysis. *Japan Journal of Industrial and Applied Mathematics*, 22(2):251–265, 2005.

[55] D.-S. Kim, Y. Cho, J.-K. Kim, and J. Ryu. QTF: Quasi-triangulation file format. *Computer-Aided Design*, 44(9):835–845, 2012.

[56] D.-S. Kim, Y. Cho, J.-K. Kim, and K. Sugihara. Tunnels and voids in molecules via Voronoi diagrams and beta-complexes. In M. L. Gavrilova, C. J. K. Tan, and B. Kalantari, editors, *Transactions on Computational Science XX*, volume 8110 of *Lecture Notes in Computer Science*, pages 92–111. Springer Berlin Heidelberg, 2013.

[57] D.-S. Kim, Y. Cho, J. Ryu, J.-K. Kim, and D. Kim. Anomalies in quasi-triangulations and beta-complexes of spherical atoms in molecules. *Computer-Aided Design*, 45(1):35 – 52, 2013. Computer-aided multi-scale materials and product design.

[58] D.-S. Kim, Y. Cho, and K. Sugihara. Quasi-worlds and quasi-operators on quasi-triangulations. *Computer-Aided Design*, 42(10):874–888, 2010.

[59] D.-S. Kim, Y. Cho, K. Sugihara, J. Ryu, and D. Kim. Three-dimensional beta-shapes and beta-complexes via quasi-triangulation. *Computer-Aided Design*, 42(10):911–929, 2010.

[60] D.-S. Kim, D. Kim, Y. Cho, and K. Sugihara. Quasi-triangulation and interworld data structure in three dimensions. *Computer-Aided Design*, 38(7):808–819, 2006.

[61] D.-S. Kim, J.-K. Kim, Y. Cho, and C.-M. Kim. Querying simplexes in quasi-triangulation. *Computer Aided Design*, 44(2):85–98, 2012.

[62] D.-S. Kim, J. Seo, D. Kim, J. Ryu, and C.-H. Cho. Three-dimensional beta shapes. *Computer-Aided Design*, 38(11):1179–1191, 2006.

[63] D.-S. Kim and K. Sugihara. Tunnels and voids in molecules via Voronoi diagram. In *Proceedings of the 2012 Ninth International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'12, pages 138–143, Rutgers University, Piscataway, New Jersey, USA, 2012. IEEE Computer Society.

[64] D.-S. Kim, C.-I. Won, and J. Bhak. A proposal for the revision of molecular boundary typology. *Journal of Biomolecular Structure and Dynamics*, 28(2):277–287, 2010.

[65] J.-K. Kim, Y. Cho, D. Kim, and D.-S. Kim. Voronoi diagrams, quasi-triangulations, and beta-complexes for disks in R^2: the theory and implementation in BetaConcept. *Journal of Computational Design and Engineering*, 1(2):79–87, 2014.

[66] J.-K. Kim, Y. Cho, R. A. Laskowski, S. E. Ryu, K. Sugihara, and D.-S. Kim. BetaVoid: Molecular voids via beta-complexes and Voronoi diagrams. *Proteins: Structure, Function, and Bioinformatics*, 82(9):1829–1849, 2014.

[67] B. Kozlikova, E. Sebestova, V. Sustr, J. Brezovsky, O. Strnad, L. Daniel, D. Bednar, A. Pavelka, M. Manak, M. Bezdeka, P. Benes, M. Kotry, A. W. Gora, J. Damborsky, and J. Sochor. CAVER Analyst 1.0: Graphic tool for interactive visualization and analysis of tunnels and channels in protein structures. *Bioinformatics*, 30(18):2684–2685, 2014.

[68] M. Krone, K. Bidmon, and T. Ertl. Interactive visualization of molecular surface dynamics. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1391–1398, 2009.

[69] M. Krone, C. Dachsbacher, and T. Ertl. Parallel computation and interactive visualization of time-varying solvent excluded surfaces. In A. Zhang, M. Borodovsky, G. Özsoyoglu, and A. R. Mikler, editors, *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, BCB'10, pages 402–405, Niagara Falls, NY, USA, 2010. ACM.

[70] M. Krone, S. Grottel, and T. Ertl. Parallel contour-buildup algorithm for the molecular surface. In J. Kennedy and J. Roerdink, editors, *IEEE Symposium on Biological Data Visualization 2011*, BioVis'11, pages 17–22, Providence, RI, USA, 2011. IEEE Computer Society.

[71] B. Lee and F. Richards. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379–IN4, 1971.

[72] J. Liang and K. A. Dill. Are proteins well-packed? *Biophysical Journal*, 81(2):751–766, August 2001.

[73] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam. Analytical shape computation of macromolecules: II. inaccessible cavities in proteins. *Proteins: Structure, Function, and Bioinformatics*, 33(1):18–29, 1998.

[74] J. Liang, C. Woodward, and H. Edelsbrunner. Anatomy of protein pockets and cavities: Measurement of binding site geometry and implications for ligand design. *Protein Science*, 7(9):1884–1897, 1998.

[75] N. Lindow, D. Baum, A.-N. Bondar, and H.-C. Hege. Dynamic channels in biomolecular systems: Path analysis and visualization. In J. Roerdink and M. Hibbs, editors, *IEEE Symposium on Biological Data Visualization 2012*, BioVis'12, pages 99–106, Seattle, WA, 2012. IEEE Computer Society.

[76] N. Lindow, D. Baum, A.-N. Bondar, and H.-C. Hege. Exploring cavity dynamics in biomolecular systems. *BMC Bioinformatics*, 14(S-19):S5, 2013.

[77] N. Lindow, D. Baum, and H.-C. Hege. Voronoi-based extraction and visualization of molecular paths. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2025–2034, 2011.

[78] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege. Accelerated visualization of dynamic molecular surfaces. *Computer Graphics Forum*, 29(3):943–952, 2010.

[79] V. Luchnikov, M. Gavrilova, N. Medvedev, and V. Voloshin. The Voronoi-Delaunay approach for the free volume analysis of a packing of balls in a cylindrical container. *Future Generation Computer Systems*, 18(5):673–679, 2002. ICCS2001.

[80] V. A. Luchnikov, N. N. Medvedev, L. Oger, and J. P. Troadec. Voronoi-Delaunay analysis of voids in systems of nonspherical particles. *Phys. Rev. E*, 59(6):7205–7212, 1999.

[81] M. Manak. Exploration of empty space among spherical obstacles via additively weighted Voronoi diagram. *Computer Graphics Forum*, 35(5):249–258, 2016.

[82] M. Manak, L. Jirkovsky, and I. Kolingerova. Interactive analysis of Connolly surfaces for various probes. *Computer Graphics Forum*, 2016. doi: 10.1111/cgf.12870. To appear.

[83] M. Manak and I. Kolingerova. Inverted ball as a boundary-constraint for Voronoi diagrams of 3D spheres. In *25th European Workshop on Computational Geometry*, EuroCG, pages 289–292, 2009.

[84] M. Manak and I. Kolingerova. Fast discovery of Voronoi vertices in the construction of Voronoi diagram of 3D balls. In M. A. Mostafavi, editor, *Seventh International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'10, pages 95–104, Laval University, Quebec, QC, Canada, 2010. IEEE Computer Society.

[85] M. Manak and I. Kolingerova. Extension of the edge tracing algorithm to disconnected voronoi skeletons. *Information Processing Letters*, 116(2):85–92, 2016.

[86] B. McConkey, V. Sobolev, and M. Edelman. Quantification of protein surfaces, volumes and atom–atom contacts using a constrained Voronoi procedure. *Bioinformatics*, 18(10):1365–1373, 2002.

[87] N. N. Medvedev, V. P. Voloshin, V. A. Luchnikov, and M. L. Gavrilova. An algorithm for three-dimensional Voronoi S-network. *Journal of Computational Chemistry*, 27(14):1676–1692, 2006.

[88] J. Newman, T. S. Peat, R. Richard, L. Kan, P. E. Swanson, J. A. Affholter, I. H. Holmes, J. F. Schindler, C. J. Unkefer, and T. C. Terwilliger. Haloalkane dehalogenases: Structure of a rhodococcus enzyme. *Biochemistry*, 38(49):16105–16114, 1999.

[89] T. Nishida and K. Sugihara. Precision necessary for d-dimensional sphere Voronoi diagrams. In K. Sugihara and D.-S. Kim, editors, *Proceedings of the 5th International Symposium on Voronoi Diagrams*, ISVD'08, pages 157–167, Drahomanov National Pedagogical University, Kyiv, Ukraine, 2008.

[90] A. Okabe, B. Boots, and K. Sugihara. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*. John Wiley & Sons, Inc., New York, NY, USA, 1992.

[91] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd edition, 2000. 671 pages.

[92] K. Olechnovič, M. Margelevičius, and Č. Venclovas. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics*, 27(5):723–724, 2011.

[93] K. Olechnovič and Č. Venclovas. Voronota: A fast and reliable tool for computing the vertices of the Voronoi diagram of atomic balls. *Journal of Computational Chemistry*, 35(8):672–681, 2014.

[94] J. Parulek and I. Viola. Implicit representation of molecular surfaces. In H. Hauser, S. Kobourov, and H. Qu, editors, *IEEE Pacific Visualization Symposium 2012*, PacificVis'12, pages 217–224, Songdo, South Korea, 2012. IEEE Computer Society.

[95] A. Poupon. Voronoi and Voronoi-related tessellations in studies of protein structure and interaction. *Current Opinion in Structural Biology*, 14(2):233–241, 2004.

[96] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 2002.

[97] Z. Prokop, J. Damborsky, F. Oplustil, A. Jesenska, and Y. Nagata. Method of detoxification of yperite by using haloalkane dehalogenases. Masaryk University, Brno, Czech Republic. Patent WO 2006/128390 A1, 2005.

[98] F. M. Richards. The interpretation of protein structures: Total volume, group volume distributions and packing density. *Journal of Molecular Biology*, 82(1):1–14, 1974.

[99] F. M. Richards. Areas, volumes, packing, and protein structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977. PMID: 326146.

[100] J. Ryu, Y. Cho, and D.-S. Kim. Triangulation of molecular surfaces. *Computer-Aided Design*, 41(6):463–478, 2009.

[101] J. Ryu, D. Kim, Y. Cho, R. Park, and D.-S. Kim. Computation of molecular surface using Euclidean Voronoi diagram. *Computer-Aided Design and Applications*, 2(1-4):439–448, 2005.

[102] J. Ryu, R. Park, and D.-S. Kim. Molecular surfaces on proteins via beta shapes. *Computer-Aided Design*, 39(12):1042–1057, 2007.

[103] J. Ryu, R. Park, J. Seo, C.-M. Kim, H.-C. Lee, and D.-S. Kim. Real-time triangulation of molecular surfaces. In *Computational Science and Its Applications – ICCSA 2007: Proceedings, Part I*, pages 55–67, Kuala Lumpur, Malaysia, 2007.

[104] M. F. Sanner, A. J. Olson, and J.-C. Spehner. Reduced surface: An efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, 1996.

[105] D. Siersma. Voronoi diagrams and Morse theory of the distance function. In O. E. Barndorff-Nielsen and E. B. V. Jensen, editors, *Geometry in Present Day Science*, pages 187–208, University of Aarhus, Denmark, 1999. World Scientific.

[106] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. GPU-Based Ray-Casting of Quadratic Surfaces. In M. Botsch, B. Chen, M. Pauly, and M. Zwicker, editors, *Symposium on Point-Based Graphics*, PBG'06, Massachusetts, USA, 2006. The Eurographics Association.

[107] M. Tanemura, T. Ogawa, and N. Ogita. A new algorithm for three-dimensional Voronoi tessellation. *Journal of Computational Physics*, 51(2):191–207, 1983.

[108] M. Totrov and R. Abagyan. The contour-buildup algorithm to calculate the analytical molecular surface. *Journal of Structural Biology*, 116(1):138–143, 1996.

[109] A. Varshney, F. P. Brooks, Jr., J. William, and W. V. Wright. Linearly scalable computation of smooth molecular surfaces. *In IEEE Computer Graphics and Applications*, 14(5):19–25, 1994.

[110] VDRC - Voronoi Diagram Research Center, `http://voronoi.hanyang.ac.kr`. Accessed on June 26, 2016.

[111] V. P. Voloshin, A. V. Anikeenko, N. N. Medvedev, and A. Geiger. An algorithm for the calculation of volume and surface of unions of spheres. application for solvation shells. In I. C. Society, editor, *Proceedings of the 2011 Eighth International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD'11, pages 170–176, Qingdao, China, 2011.

[112] G. F. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Premier mémoire. Sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik*, 1908(133):97–102, 1908.

[113] H. M. Will. *Computation of additively weighted Voronoi cells for applications in molecular biology*. PhD thesis, Swiss Federal Institute of Technology, Zurich, 1999.

[114] M. A. Williams, J. M. Goodfellow, and J. M. Thornton. Buried waters and internal cavities in monomeric proteins. *Protein Science*, 3(8):1224–1235, 1994.

[115] E. Yaffe and D. Halperin. Approximating the pathway axis and the persistence diagrams for a collection of balls in 3-space. *Discrete & Computational Geometry*, 44(3):660–685, 2010.

[116] M. Zemek, M. Manak, and I. Kolingerova. Advanced space filtering for the construction of 3D additively weighted Voronoi diagram. In *The Fifth International Conference on Advanced Engineering Computing and Applications in Sciences*, ADVCOMP 2011, pages 37–43, Lisbon, Portugal, 2011. IARIA.