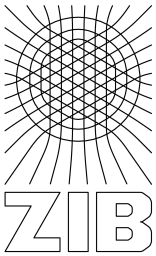


Konrad-Zuse-Zentrum für
Informationstechnik Berlin
&
Freie Universität Berlin
Fachbereich Mathematik
und Informatik

Visual Analysis of Atomic Structures Based on the Hard-Sphere Model

Dissertation

zur Erlangung des akademischen Grades
des Doktors der Naturwissenschaften (Dr. rer. nat.)

eingereicht am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

vorgelegt von

Norbert Lindow

Berlin, 2017

Erstgutachter:	Prof. Dr. Christof Schütte *Freie Universität Berlin, Germany*
Zweitgutachter:	Prof. Dr. Thomas Ertl *Universität Stuttgart, Germany*
Tag der Disputation:	02.06.2017

Abstract

Visualization and Analysis of atomic compositions is essential to understand the structure and functionality of molecules. There exist versatile areas of applications, from fundamental researches in biophysics and materials science to drug development in pharmaceutics. For most applications, the hard-sphere model is the most often used molecular model. Although the model is a quite simple approximation of reality, it enables investigating important physical properties in a purely geometrical manner. Furthermore, large data sets with thousands up to millions of atoms can be visualized and analyzed. In addition to an adequate and efficient visualization of the data, the extraction of important structures plays a major role. For the investigation of biomolecules, such as proteins, especially the analysis of cavities and their dynamics is of high interest. Substrates can bind in cavities, thereby inducing changes in the function of the protein. Another example is the transport of substrates through membrane proteins by the dynamics of the cavities. For both, the visualization as well as the analysis of cavities, the following contributions will be presented in this thesis:

1. The rendering of smooth molecular surfaces for the analysis of cavities is accelerated and visually improved, which allows showing dynamic proteins. On the other hand, techniques are proposed to interactively render large static biological structures and inorganic materials up to atomic resolution for the first time.

2. A Voronoi-based method is presented to extract molecular cavities. The procedure comes with a high geometrical accuracy by a comparatively fast computation time. Additionally, new methods are presented to visualize and highlight the cavities within the molecular structure. In a further step, the techniques are extended for dynamic molecular data to trace cavities over time and visualize topological changes.

3. To further improve the accuracy of the approaches mentioned above, a new molecular surface model is presented that shows the accessibility of a substrate. For the first time, the structure and dynamics of the substrate as hard-sphere model is considered for the accessibility computation. In addition to the definition of the surface, an efficient algorithm for its computation is proposed, which additionally allows extracting cavities.

The presented algorithms are demonstrated on different molecular data sets. The data sets are either the result of physical or biological experiments or molecular dynamics simulations.

Zusammenfassung

Die Visualisierung und Analyse atomarer Strukturen ist essenziell für das Verständnis des Aufbaus und der Funktionsweise von Molekülen. Es gibt vielfältige Anwendungsgebiete, angefangen von Grundlagenforschungen in der Biophysik und den Materialwissenschaften bis hin zur Medikamentenentwicklung in der Pharmazie. Das Modell harter Kugeln, auch Kalottenmodell genannt, ist dabei das am häufigsten verwendete Molekülmodell. Obwohl es ein sehr vereinfachtes Modell ist, ermöglicht es die geometrische Betrachtung wichtiger physikalischer Eigenschaften und erlaubt zudem, große Daten mit Tausenden bis hin zu Millionen von Atomen darzustellen und zu analysieren. Neben einer adequaten und performanten Visualisierung der Daten spielt vor allem die Extraktion von Strukturen eine große Rolle. Bei der Untersuchung von Biomolekülen, wie Proteinen, ist besonders die Analyse und Dynamik der Kavitäten von großem Interesse. In den Kavitäten können Substrate binden, die damit die Funktionsweise eines Proteins ändern, oder sie können durch die Dynamik der Kavitäten durch Membranen transportiert werden. Sowohl für die Visualisierung als auch für die Analyse der Kavitäten werden in dieser Dissertation die folgenden Beiträge geleistet:

1. Zum einen wird die Darstellung glatter Oberflächen, die sich für die Analyse von Kavitäten eignen, beschleunigt und visuell verbessert, wodurch sie auf dynamische Proteine angewendet werden können. Zum anderen werden Methoden vorgestellt, die erstmals erlauben große statische biologische Strukturen und anorganische Materialien bis auf atomare Auflösung interaktiv darzustellen.
2. Für die Extraktion von Kavitäten wird ein Voronoi-basiertes Verfahren mit einer hohen geometrischen Genaugkeit bei einer vergleichsweise hohen Geschwindigkeit vorgestellt. Dazu werden neue Methoden präsentiert, welche die Kavitäten innerhalb der molekularen Struktur darstellen und hervorheben. Des Weiteren werden die Methoden für dynamische Daten erweitert, um Kavitäten über die Zeit verfolgen und topologische Veränderungen visualisieren zu können.
3. Um die Genauigkeit der oben genannten Verfahren weiter voranzutreiben, wird eine neue Molekülоberfläche vorgestellt, welche die erreichbaren Regionen eines Substrates zeigt. Dabei wird erstmals die Struktur und Dynamik des Substrates in Form des Kalottenmodells berücksichtigt. Neben der Definition der Oberfläche wird ein effizienter Algorithmus für dessen Berechnung präsentiert, der es zudem erlaubt Kavitäten zu extrahieren.

Die vorgestellten Algorithmen werde an verschiedenen molekularen Daten demonstriert. Die Daten sind das Ergebnis physikalischer und biologischer Experimente oder stammen aus molekularen Simulationen.

Acknowledgements

This work has been carried out at the Zuse Institute Berlin (ZIB) in the department of Visual Data Analysis. After such a long period of time to finish this thesis I would like to thank all people who supported me during this intense work.

First and foremost I would like to thank Hans-Christian Hege and Daniel Baum for the excellent and close collaboration in all our projects. Christian, it is a pleasure to investigate difficult problems with you. Your knowledge and experience inspired my work in all directions. Daniel, thank you so much for pushing me to the outer limits in all aspects of my work. Your unlimited support and encouragement made this work possible.

I am grateful to Prof. Dr. Christof Schütte, president of the ZIB, who gave me the possibility to graduate at the institute and for his support and trust in me.

Being part of the department of Visual Data Analysis means working with many experts in many different fields. This automatically leads to new ideas and critical discussions. I want to thank all of you not only for the fruitful and inspiring conversations, but also for the amazing atmosphere. In particular, I would like to thank Kai, Uli, Steffen, Andrea, Morgan, Vincent, Jens, Malte and Olaf.

Special thanks to my family and friends for the continuous support and the faith in me.

Finally, I would like to thank Marina for her love and patience during all the time, and my daughter Marlene for all these wonderful moments.

Thank You!

Contents

1 Introduction

1.1 The Molecular World

Studying the structure and function of materials requires their investigation at different length and time scales. At the lower end of the length scale, often the atomic or molecular level is studied. Atoms are the smallest stable components of all ordinary matter. They are held together by strong forces, called chemical bonds, thereby forming structures such as molecules. Based on the large number of possible combinations, a huge variation of organic and inorganic materials is built. Typically, atoms and molecules are arranged in patterns. These patterns are responsible for the stability and functionality of the materials.

Most organic materials consist of a hierarchical composition of atoms and molecules. Starting from the atomic level, molecules are formed that can be composed to macromolecules and sub-cellular units [16] (Figure 1.1). This further proceeds to the levels of cellular structures, cells, tissues, organs, systems, and organisms. Typically, the molecules of organic materials are denoted by the term *biomolecules*. Biomolecules differ in size, shape, and structure. Some are small such as lipid molecules and build components of more complex structures like membranes. Larger molecules are often called macromolecules. The atoms and molecules of most of these biomolecules are arranged in chain-like structures, such as proteins. Due to their diverse functionalities in all living organisms, proteins are of particular interest in many research fields including biophysics, biochemistry, and pharmaceutics. Most proteins work like enzymes, which are activated by smaller molecules, called substrates or ligands. The ligands interact with the proteins, thereby inducing changes in the geometry, which modify its function. Typical ex-

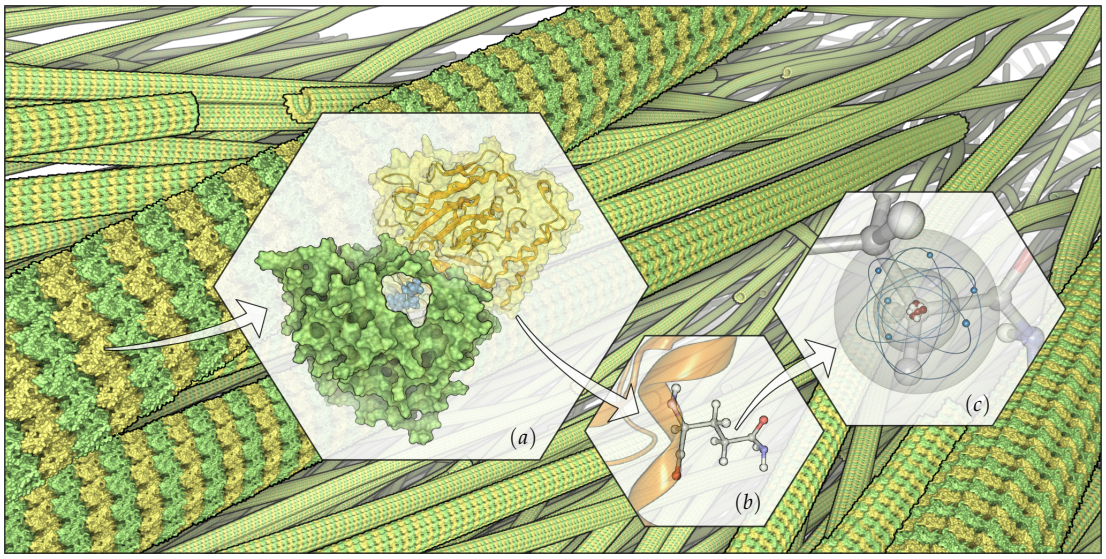


Figure 1.1: *From microtubules strands to single atoms. Each strand consists of proteins arranged in a spiral structure (a). A protein is composed of chains of amino acids (b), which are compounds of a small number of atoms (c).*

amples for such interactions with high interest in pharmaceutics are binding processes of antibiotics or painkillers. Membrane proteins are the target of most of the modern medicinal drugs. These proteins are integrated in biological membranes or interact with them. Some of them work as receptors to transfer signals about the internal and external environment. Others are responsible for the transport of molecules and ions through the membrane. Furthermore, proteins perform structural or mechanical tasks, such as microtubules and actin filaments in the cytoskeleton or myosin in muscles. Other important tasks include DNA replication, catalyzing metabolic reactions, and metabolite transport.

Inorganic materials consist of all compounds that are not organic [240]. This is often roughly defined as all materials without carbon compounds, in particular carbon-hydrogen compounds. However, there are many exceptions that create much overlap between organic and inorganic materials. Typical inorganic materials are metals, salts, minerals, many acids and gases. The atoms and molecules of many solid inorganic materials are arranged in lattice structures. The stability of these materials is tightly coupled with the type of lattice and its regularity. Defects and dislocations have a major impact on the stability and functioning of the materials. Studying these properties of the lattices structures is a major task in materials science.

1.2 Understanding the Molecular World

Experiments and simulations are the two main techniques to get insights into the huge world of molecular structures. While experiments are often used to

acquire structural information, simulations are performed to study molecular processes and interactions on an atomic level. Both fields are supported by visualization techniques that provide tools to analyze, validate, and modify molecular structures.

Experimental Structure Determination

Because of the small size of atoms it is difficult to create a real picture of their structure. Even on the molecular level, it is only partially possible to determine the arrangement of the atoms and this only for static snapshots. The two main techniques to solve the structure of biomolecules such as proteins are nuclear magnetic resonance spectroscopy (NMR) [244] and X-ray crystallography [243]. X-ray crystallography in general achieves a higher resolution and has no size limitation for macromolecules. However, the molecule needs to be crystallized and the crystal must be well ordered to diffract measurable at high angles. This process might influence the conformation of the molecule or might damage the structure. For NMR, the molecule needs to be in a soluble state and typically isotopically labeled. This state is usually closer to the natural state of biomolecules. On the other hand, the approach provides a lower resolution and is only applicable for molecules with less than 50 kDa. An upcoming technique to acquire macromolecular structures in near-atomic resolution is single particle cryo-electron microscopy [231], a special variant of cryo-electron microscopy that is often used in structural biology. Although its resolution is yet lower, it can reconstruct the chain-like structure based on a large number of specimen in different orientations. In contrast to crystallization, cryo-electron microscopy allows capturing different conformations and conformational transitions

Furthermore, electron microscopy and tomography are often used in material science to study the lattice structure of inorganic materials such as metals and alloys. Another approach for inorganic materials is atom probe tomography (APT) [105] which provides close to atomic resolution and allows reconstruction of millions of atoms. The drawbacks of APT are the lower limits in the specimen size and the restrictions in the detection efficiency, where approximately 40% of the atoms get lost. Furthermore, APT is a destructive technique, which means the specimen will be destroyed during the process. This destruction has additional effects on the structure that needs to be controlled.

For all techniques, often the preparation stage is already challenging and can partially destroy or change the structure of the specimen because of changes in the environment.

Simulation

Due to the limitations in acquiring molecular structures, it is even more challenging to obtain information about the functionality of molecular processes,

for example, how an ion channel works inside a membrane or how a substrate interacts with a protein. In many cases the only way to get information about molecular processes on the atomic level is to perform molecular dynamics simulations. To do so, the structure and interaction of atoms and molecules need to be mathematically modeled. Many of such models have been developed in the past with the purpose to reflect most realistically their physical properties measured by experiments [49, 20, 221, 118, 88, 212]. Note that the result of a molecular dynamics simulation is restricted to the physical correctness of the underlying model.

In most cases, complex molecular systems with hundreds and thousands of atoms will be simulated using classical molecular models [67, 210]. In such models, the molecular system is represented by particles whose motion is defined by a force field containing Coulomb, van der Waals and bonding forces. Two very often used tools to simulate molecules in a classical way are NAMD [191] and Gromacs [196]. Although quantum mechanical models are physically more accurate, due to the high computational costs, the application of these models is only possible for very small molecular systems or small parts of biomolecular systems. On the other hand, classical models seem to be sufficient to study most molecular interactions where the covalent bonds between the atoms are fixed and the behavior of the molecular system does not depend sensitively on fine-tuned energy values. The spatial extension of molecules based on the classical model is most often represented by the *hard-sphere model*. In this model, each atom is represented as a 'hard' sphere which is assumed to be impenetrable except for bonded atoms. The radii of the atoms are determined based on the van der Waals forces.

Visualization

As mentioned before, it is difficult to acquire a real image of atomic structures. All results of experimental techniques are reconstructions based on non-visual measurements. Hence, molecular visualization models are necessary to create a specific abstraction of the reality. These abstractions highlight relevant properties, thereby supporting the understanding of the structures. This is important to communicate and discuss results and to illustrate processes. Visualizations are required in nearly all fields of molecular analysis, starting with validations and modifications over preparations of simulation setups to detailed structure analyzes and full trajectory investigations. Even before the first computer graphic visualizations, people developed physical models and hand-drawn graphical depictions for atoms and molecules to understand and analyze their structure and behavior [106, 98, 220]. These early tools strongly influenced the 3-dimensional computer visualizations of molecules and form a basis of today's classical visualization models. Depending on the size and type of the molecular data, different visualization models are required to analyze the structure. Most of these visual models are based on the corresponding physical models. The focus of this thesis lies on visualization and analysis

of molecules based on the *hard-sphere model*. With this model it is possible to represent molecules in a purely geometric manner including the definition of molecular surfaces and accessibility.

Visualization techniques for biomolecular structures fall into two categories: The first deals with explicit representation of the structure. To this category belong classical representations like the ball-and-stick model, which shows the atoms and bonds, or more abstract models such as the secondary structure representation, which highlights stable patterns of a protein. Surface models that show the spatial extension of molecules are of particular interest. These models create an imaginary hard boundary that separates the space inside the molecule and the space that is accessible by other molecular structures. Users apply surface models to analyze the shape of molecules and to detect and study cavities, for example, potential binding sites for drugs. The second category comprises methods for computing, analyzing, and visualizing the complementary space of molecules, that is the space consisting of all regions within the molecular structures that are not occupied by the atoms of the molecule. Cavities are the main places for molecular interactions and transport processes. Understanding interactions related to cavities is the key to many open questions in biochemistry, molecular biology, and pharmaceutics. This is because molecular interactions are the driving forces for many biochemical processes taking place in all kinds of organisms, from the most simple to highly developed ones.

With the increasing performance of modern hardware and algorithms, the size and complexity of the data sets have increased, too. Furthermore, the demands in studying dynamic structures from simulations in real-time have raised. Acceleration and modernization of the most important classical molecular surface visualizations is therefore one focus of this thesis. In addition, a new geometric approach for the detection of cavities is proposed to overcome limitations of previous methods. In combination with this detection, novel techniques for the visualization and time-dependent analysis of cavities are presented. Taking the geometrically oriented molecular analysis a step closer to physical reality is another focus of this thesis. This finally leads to a novel molecular surface model called *ligand excluded surface*. In the following the contributions of this thesis are described in more detail.

1.3 Contributions

The focus of this thesis lies on the development of methods for the visualization and analysis of biological macromolecules, such as proteins, with up to 100 000 atoms. Nevertheless, an excursion into the visualization of larger molecular data sets is given that can consist of sub-cellular structures or inorganic materials with millions or even billions of atoms. Most of the presented techniques are not restricted to the specific structure of proteins: they can be

applied to other molecular data sets or even to particle data sets from other fields.

Smooth Molecular Surfaces

As mentioned before, molecular simulations are performed to study the functionality of molecules such as their binding affinity. Binding processes are of particular interest for the development and analysis of drugs. Visualization of the results of such simulations is a key step for investigating the process of interest but also to verify the simulation. For this purpose, surface models based on the hard sphere model are well suited because of two main reasons. First, they provide a good approximation of the spatial extension and shape of the molecules. This allows analyzing the cavities in proteins where, for example, the binding process takes place. And second, the visualization of surfaces based on the hard sphere model can be done efficiently such that it can be used for molecular dynamics investigations. Of particular interest are smooth molecular surfaces, especially the *solvent excluded surface* that provides a very suitable visualization of the accessibility e.g. of solvents. However, the visualization of this smooth surface model requires some computation time. Thus, in the past it was not possible to use the surface in real-time for protein data. In Chapter 3, an accelerated and visually improved computation of the solvent excluded surface is presented. The techniques are also utilized for another surface model, called *molecular skin surface*. Apart from molecular surface visualization, the latter surface has additional potential for the rendering of molecular cavities.

Van der Waals Surface

While smooth surfaces are suitable to investigate the molecular accessibility of single proteins, their computation is often too expensive for larger structures. The variability of materials emerges at the atomic level and continues at higher length scales with many kinds of different structures. Particularly biological materials, but also bio-inspired materials, use hierarchical structures to create multifunctional materials [129, 30]. Many investigations of such materials start at the atomic level and proceed to mesoscopic scales, where the transition, for example, to continuum models, can be made. The corresponding measurements and simulations involve millions or even billions of individual atoms [170]. Therefore the need arises for visualization techniques that bridge atomic and mesoscopic length scales. In Chapter 4, a novel approach is presented to cover these length scales for the classical hard sphere model of the atoms. This allows interactive visualization of inorganic materials from atom probe tomography with several hundred millions of atoms or even cellular data such as actin filaments or microtubules with billions of atoms.

Molecular Cavities

Although it is possible to simulate molecular dynamics of single proteins or proteins inside a membrane as well as interacting forces between well-placed ligands and proteins, it is quite difficult to completely simulate interaction processes, like the transport of an ion through a membrane protein or the path of a ligand from outside the protein to its binding site. Biophysicists can use simulations to answer the question whether a ligand binds to a protein or not, but they can often not answer the question if the ligand can reach the binding site. Thus the necessity arises for methods that allow the user to quickly analyze the cavity structure inside a protein and to simultaneously detect possible paths for a ligand. To achieve this in a reasonable time, the computation of these molecular paths and cavities is often restricted to a geometrical analysis of the hard-sphere model. A survey of the most important geometrical techniques is provided in Chapter 5 and a new approach to compute and analyze the full geometric cavity structure in a molecule is presented in Chapters 6 and 7. The technique provides a high geometrical accuracy with respect to the underlying model while keeping the computational time low enough, such that it is applicable to large proteins within a few seconds.

Cavity Dynamics

The internal cavities of proteins are dynamic structures and their dynamics may be associated with conformational changes which are required for the functioning of the protein. It is therefore of particular interest to understand how conformational changes that accompany the reaction path of molecular transporters depend on changes in the number, shape, and volume of internal cavities. In order to study the dynamics of protein cavities, appropriate tools are required that allow rapid identification of the cavities as well as assessment of their time-dependent structures. There are numerous tools to inspect and analyze molecular dynamics trajectories, including VMD [234], PyMol [197], Amira [223], and MegaMol [165]. However, there are only a few tools that allow the investigation of the dynamics of the cavities. A tool has been developed that allows one to interactively trace and analyze cavities over time with a higher geometrical accuracy. To study the cavity dynamics, the analysis is supported by new visualizations techniques. The tool will be presented in Chapter 8.

Ligand Excluded Surface

The most popular molecular surface is the solvent excluded surface, described in Chapter 3. It provides information about the accessibility of a molecule with respect to a sphere approximating a solvent molecule. During a period of almost four decades, the SES has served many purposes, including pure visualization, analysis of molecular interactions, and study of cavities in

molecular structures. However, if one is interested in the surface of a molecule that is accessible to a second molecule whose shape differs significantly from a sphere, a different concept is necessary. The fundamental problem that almost *no* molecule, not even water, is well approximated by a spherical shape, has so far not been addressed in the computation and visualization of molecular surfaces.

To overcome this limitation, a new molecular surface is proposed in Chapter 9, called *ligand excluded surface* (LES). It represents the surface of a receptor that is accessible to a specific individual ligand, which is represented by its spatial configuration of atom spheres (rather then a single 'approximating' sphere). Thus receptor and ligand are geometrically represented in the same manner. Apart from the definition of the ligand excluded surface, an efficient algorithm for its discrete computation is proposed. Furthermore, this algorithm can be easily extended to compute also cavities that are large enough to host the ligand molecule. In addition to the geometry of the cavities, also information about how the ligand is positioned inside the cavities are obtained. This might be of particular interest for the application of subsequent docking simulations.

1.4 Publications

This thesis is based on the following publications.

- Norbert Lindow, **Dynamische Molekülоberflächen**, Diplomarbeit, Technical University Berlin, Daniel Baum, Marc Alexa, Hans-Christian Hege (Advisors), 2010.
- Norbert Lindow, Daniel Baum, Steffen Prohaska, Hans-Christian Hege, **Accelerated Visualization of Dynamic Molecular Surfaces**, *Computer Graphics Forum*, Vol.29, pp. 943–952, 2010.
- Norbert Lindow, Daniel Baum, Hans-Christian Hege, **Voronoi-Based Extraction and Visualization of Molecular Paths**, *IEEE Transactions on Visualization and Computer Graphics*, 17(12), pp. 2025–2034, 2011.
- Norbert Lindow, Daniel Baum, Hans-Christian Hege, **Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale**, *Computer Graphics Forum*, 31(3), pp. 1325–1334, 2012.
- Norbert Lindow, Daniel Baum, Ana-Nicoleta Bondar, Hans-Christian Hege, **Dynamic Channels in Biomolecular Systems: Path Analysis and Visualization**, *Proceedings of IEEE Symposium on Biological Data Visualization (biovis'12)*, pp. 99–106, 2012.
- Norbert Lindow, Daniel Baum, Ana-Nicoleta Bondar, Hans-Christian Hege, **Exploring cavity dynamics in biomolecular systems**, *BMC Bioinformatics*, Vol.14, 2013.
- Norbert Lindow, Daniel Baum, Hans-Christian Hege, **Ligand Excluded Surface: A New Type of Molecular Surface**, *IEEE Transactions on Visualization and Computer Graphics*, 20(12), pp. 2486–2495 , 2014.

In addition, two state of the art reports about molecular visualization and cavity analysis were published in cooperation with other domain experts in these fields.

- Barbora Kozlikova, Michael Krone, Norbert Lindow, Martin Falk, Marc Baaden, Daniel Baum, Ivan Viola, Julius Parulek, Hans-Christian Hege, **Visualization of Biomolecular Structures: State of the Art**, *Eurographics Conference on Visualization (EuroVis) - STARs*, 2015.
- Michael Krone, Barbora Kozlikova, Norbert Lindow, Marc Baaden, Daniel Baum, Julius Parulek, Hans-Christian Hege, Ivan Viola, **Visual Analysis of Biomolecular Cavities: State of the Art**, *Computer Graphics Forum*, Vol.35(3), pp.1467–8659, 2016.

1.5 Data Courtesy

To show the usefulness and improvements of new approaches and techniques it is necessary to demonstrate them on practical data sets. For this reason, all methods that will be proposed in this thesis are evaluated on different data sets. Many of these data sets were retrieved from data bases such as the protein data bank PDB [184] or the virus data base VIPERdb [233], both of which provide a huge variation on molecular data sets. Furthermore, specific data sets were provided by cooperation partners and colleagues of different universities and institutes. Special thanks to all of them. In the following, a list of all the specific data sets and their courtesy is given:

- To test the performance of smooth molecular surfaces in Chapter 3, several dynamic molecular trajectories were provided by Markus Weber and Bernd Kallies from Zuse Institute Berlin (ZIB), Germany. The trajectories are mainly protein docking simulations.
- Jean-Marc Verbavatz, Garrett Greenan and Anthony Hyman from the Max Planck Institute Dresden, Germany, Thomas Müller-Reichert and Stefanie Redemann from Technical University Dresden, Germany, and Eileen O'Toole from the University of Colorado, USA, provided some microtubule tomograms. The microtubules were reconstructed by Britta Weber from ZIB. The data sets are used to demonstrate the rendering of large atomic data sets (Chapter 4).
- Alexander Rigort and Wolfgang Baumeister from the Max Planck Institute Munich, Germany, provided the tomograms containing actin filaments and ribosomes that were reconstructed by David Günther from ZIB. Analogously to the microtubule data sets, the data is used for the rendering of large atomic data sets (Chapter 4).
- The atom probe tomography data sets of inorganic materials were given by Hisham Aboulfadl and Frank Mücklich from Saarland University, Germany. These data sets were additionally used to test the rendering technique in Chapter 4.
- The dendritic core multi-shell nano-transporter was provided by Marcus Weber and Amir Sedighi from ZIB. It was used for the cavity detection and cavity analyses in Chapters 6 and 7.
- For the dynamic cavity analysis in Chapter 8, Ana-Nicoletta Bondar from FU Berlin, Germany, provided several trajectories of bacteriorhodopsin.
- Greg Bowman from the Miller Institute, University of California, Berkeley, USA, provided a molecular dynamics trajectory to test the ligand excluded surface for dynamic data sets. The data is presented in Chapter 9.

2 Basics

This chapter contains the basics that will be important for this thesis. It starts with the description of some fundamental geometrical structures for molecular visualization. Afterwards, a short introduction about the structure of molecules is given, where the focus lies on biomolecules, especially proteins. Subsequently, the most important 3-dimensional molecular visualization models are described. Then, a formal definition about molecular paths and cavities is presented. And finally, grid data structures are proposed to accelerate most visualization and analyzes algorithms.

2.1 Geometry

The three most important basic geometrical structures for this thesis are implicit surfaces, Voronoi diagrams, and skin surfaces. Before these structures are described, all necessary mathematical notations are given.

2.1.1 Notations

The real space is denoted by $\mathbb{R}$ and the n-dimensional real space by $\mathbb{R}^n$. Consider a point $x \in \mathbb{R}^n$. The ith component of this point is given by x_i with $i \in \mathbb{N}$ and $1 \leq i \leq n$. The vector pointing to x is denoted by $\vec{x}$. If all components of a point or vector are 0, it is also denoted by 0. For a set of m points $x_j \in \mathbb{R}^n$ with $j \in \mathbb{N}$ and $1 \leq j \leq m$, the last index always describes the component of the point, so x_{ji} is the ith component of the jth point. In the following, a list of all notations for operations on points and vectors that are used in this thesis is given. Let $x, y \in \mathbb{R}^n$ be two points and let $Z \subset \mathbb{R}^n$ be a finite set containing m points $z_1, ..., z_m$.

- $\langle x, y \rangle = \sum_{i=1}^{n} x_i \cdot y_i$ denotes the dot product.
- $x^2 = \langle x, x \rangle$.
- $\|x\| = \sqrt{x^2}$ is the Euclidean norm.
- $x \times y = \begin{pmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{pmatrix}$ is the cross product (only for $n = 3$).
- $\mathrm{Aff}(Z) = \{\sum_{i=1}^{m} \lambda_i \cdot z_i \mid z_i \in Z,\ \sum_{i=1}^{m} \lambda_i = 1\}$ is the affine hull.
- $\mathrm{Conv}(Z) = \{\sum_{i=1}^{m} \lambda_i \cdot z_i \mid z_i \in Z,\ \sum_{i=1}^{m} \lambda_i = 1,\ \lambda_i \geq 0\}$ is the convex hull.
- $\mathrm{sgn} : \mathbb{R} \to \mathbb{R}$, with $\mathrm{sgn}(x) = \begin{cases} 1, & x \geq 0 \\ -1, & x < 0 \end{cases}$

2.1.2 Implicit Surfaces

In 3-dimensional Euclidean geometry, a surface is a 2-dimensional subspace of the 3-dimensional real space $\mathbb{R}^3$. The *implicit surface* is one possibility to describe a surface in a single equation. It is often used in computer graphics to visualize the surface of simple objects, whose triangulation is expensive, like spheres or cylinders. Most of the molecular visualization models are composed of simple implicit surfaces. For this reason, a short abstract of this type of representation of a surface is given here.

Consider a function $f : \mathbb{R}^3 \to \mathbb{R}$, which maps a point to a real value. The corresponding implicit surface F is defined as the set of all points for which f evaluates to 0, so formally

$$F = f^{-1}(0) = \left\{ p \in \mathbb{R}^3 \mid f(p) = 0 \right\}.$$

The sign of f indicates on which side of a surface a point lies. For closed surfaces, one can distinguish between inner and outer points for $f(p) < 0$ and $f(p) > 0$, respectively. The normal vector in a point on the surface is given by the gradient of f,

$$\nabla f(p) = \left(\frac{\delta f}{\delta p_1}, \frac{\delta f}{\delta p_2}, \frac{\delta f}{\delta p_3} \right)^T.$$

Note that with the current definition, it is not guaranteed that the surface has not any singularity. Furthermore, there is not necessarily a normal in each point of the surface. In order to achieve these requirements, f must be continuous and differentiable with gradients unequal to 0 on the surface.

In this thesis, a specific subset of implicit surfaces is often used, called algebraic surfaces. These surfaces are defined by polynomial functions. A further important subset of algebraic surfaces are quadrics. For quadrics, the

polynomials have exactly degree two. For example, a torus can be described as an algebraic surface of degree 4, and a sphere is a typical quadric.

A similar definition of surfaces is given by *isosurfaces*. In contrast to an implicit surface, an isosurface consists of all points $p \in \mathbb{R}^3$ for which $f(p) = c$, where c is a user-defined constant. Note that each isosurface can be transformed into an implicit surface by creating a new function $\hat{f}(p) = f(p) - c$. In this thesis, the usage of implicit surfaces is preferred. The interested reader can find more information about implicit surfaces in the book by Bloomenthal and Wyvill [19].

2.1.3 Voronoi Diagrams

In 1908, the Ukrainian and Russian mathematician Georgy Feodosevich Voronoi defined a decomposition of the n-dimensional real space based on a finite set of points, which was later called *Voronoi diagram* [236]. Voronoi diagrams are a powerful tool in many fields, like geometry or optimization. In this thesis, they are used for the computation of surfaces and the cavity analysis of molecules. A short introduction of Voronoi diagrams is given in the following.

Let P be a finite set of m input points $p_i \in \mathbb{R}^n$ with $i \in I = \{1, ..., m\}$. For each input point, a region is determined that contains all points whose distance to this input point is less or equal than to any other input point. These regions are called *Voronoi regions* based on a distance function $d : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$. A formal definition of the Voronoi region of the ith input point is given by the set

$$V_i = \left\{ p \in \mathbb{R}^n \mid \forall j \in I,\ j \neq i,\ d(p_i, p) \leq d(p_j, p) \right\}.$$

For the classical Voronoi diagram, the distance function d is equal to the Euclidean distance, so $d(p, q) = \|p - q\|$ for $p, q \in \mathbb{R}^n$. Based on the definition of the regions, the whole Voronoi diagram Vor_P is defined as the set of all nonempty intersections of Voronoi regions, so

$$Vor_P = \left\{ V_K \ \mid \ V_K = \bigcap_{i \in K} V_i \neq \emptyset,\ K \subseteq I \right\}.$$

The dual structure of a classical Voronoi diagram is called Delaunay complex. For the 2-dimensional case it is called Delaunay triangulation [51]. For each k-dimensional Voronoi element, there exits exactly one corresponding l-dimensional Delaunay element with $k + l = n$. Based on a given Voronoi diagram, the Delaunay complex can be defined as

$$Del_P = \left\{ D_K \mid D_K = \mathrm{Conv}\left\{ p_i \mid i \in K \right\}, V_K \in Vor_P \right\}.$$

For molecular analysis and visualization, the 3-dimensional Voronoi diagram is of particular interest. Thus, the focus of the following descriptions lies on this dimension, but 2-dimensional diagrams are often used for illustrations,

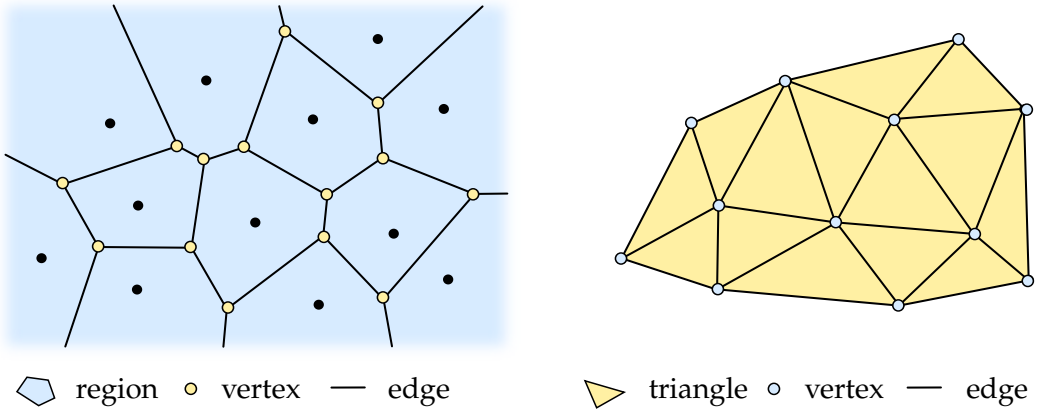


Figure 2.1: *Illustration of a Voronoi diagram (left) for a set of 2-dimensional input points (black) and the dual Delaunay triangulation (right). Each Voronoi region corresponds to exactly one Delaunay vertex, which is equal to an input point. Furthermore, each Voronoi vertex corresponds to exactly one Delaunay triangle and each Voronoi edge to exactly one orthogonal Delaunay edge.*

(Figure 2.1). The 3-dimensional Voronoi diagram consists of Voronoi regions, faces, edges, and vertices that describe the 3-, 2-, 1-, and 0-dimensional intersections of Voronoi regions. A 3-dimensional Voronoi diagram is denoted as non-degenerate if each face is the intersection of exactly 2 regions, each edge is the intersection of exactly 3 regions, and each vertex is the intersection of exactly 4 regions. Note that most algorithms can compute only non-degenerate Voronoi diagrams. In order to avoid degenerated cases, one can use simulation of simplicity or a simple perturbation of the input points. Edelsbrunner presented simulation of simplicity [58] as a concept to remove all degenerated cases in geometrical algorithms, without changing the input data. Therefore it is necessary to modify all basic geometrical operations in the algorithm, which can be difficult in specific cases. However, the main problem of this concept is, that it is based on an exact floating point arithmetic or at least a very high floating point accuracy, which can decrease the performance of the algorithm. Although this concept is geometrically correct, in practice a simple perturbation of the input data is often preferred. In particular, this means a random vector is added to each point. This vector must be small enough, so that the resulting error is irrelevant for the corresponding application. Furthermore, the algorithm must be still able to detect degenerations, abort the current computation, and start again with a new perturbation. For many applications this is easier to implement and does not influence the efficiency of the algorithms. From now on, only non-degenerate Voronoi diagrams are considered.

After the first definition of the classical Voronoi diagram, a lot of variations have been presented. A comprehensive overview of variations can be found in the book of Aurenhammer et al. [8]. There are two possibilities to modify a Voronoi diagram. The first is to change the distance function. For example,

the 2-dimensional Voronoi diagram on the surface of a sphere requires a distance function that measures the arc length of the shortest curve between two points on a sphere. The second possibility is to extend the input structure. Instead of points, one can consider spheres or arbitrary other geometrical objects as input. The distance measurements usually become more complex for these objects. Note that, in general, the more complex the input data the more difficult is the computation of the Voronoi diagram. For molecular visualization, the Voronoi diagrams of weighted points and spheres are often utilized.

2.1.4 Skin Surface

In 1999, Edelsbrunner presented a new approach to generate a smooth surface for a finite set of input spheres, called *skin surface* [54]. The skin surface is a C^1 continuous surface that can be decomposed into patches of quadrics. Its shape depends only on a single parameter $s \in (0,1) \subset \mathbb{R}$, called shrink factor. In this thesis, the skin surface is mainly used to visualize cavities inside molecules, but also as molecular surface representation. Therefore, the description of the surface definition and its construction by Edelsbrunner is summarized in the following.

Sphere Algebra

First, all spheres are transformed into weighted points. Consider a weighted point $x \in \mathbb{R}^3 \times \mathbb{R}$ with position p_x and weight w_x. The distance of an arbitrary point $p \in \mathbb{R}^3$ to x is defined as $d_x(p) = (p - p_x)^2 - w_x$. For a sphere with position p_x and radius $\sqrt{w_x}$, the point p lies inside the sphere if $d_x(p) < 0$. Respectively, the point lies outside this sphere if $d_x(p) > 0$. So each weighted point with its distance function corresponds to a sphere. Note that the term imaginary sphere is used for points with negative weights and that a sphere is degenerated if $w_x = 0$.

The weighted points will be embedded into a 4-dimensional vector space. Therefore, Edelsbrunner defined the addition and scalar multiplication for two weighted points x, y and a scalar $s \in \mathbb{R}$ as:

$$\begin{aligned} p_{x+y} &= p_x + p_y \\ w_{x+y} &= w_x + w_y + 2 \cdot \langle p_x, p_y \rangle \\ p_{s \cdot x} &= s \cdot p_x \\ w_{s \cdot x} &= s \cdot w_x + (s^2 - s) \cdot p_x^2 \end{aligned}$$

With these operations, the scaling of the weight of a point can be represented by a linear interpolation of the point and an artificial helper point. Let again x be a point with positive weight. The helper point $\tilde{x}$ is selected such that $p_{\tilde{x}} = p_x$ and $w_{\tilde{x}} = 0$. The scaling x^s of x by $s \in \mathbb{R}$ can now be formulated as

$$x^s = (1 - s) \cdot \tilde{x} + s \cdot x.$$

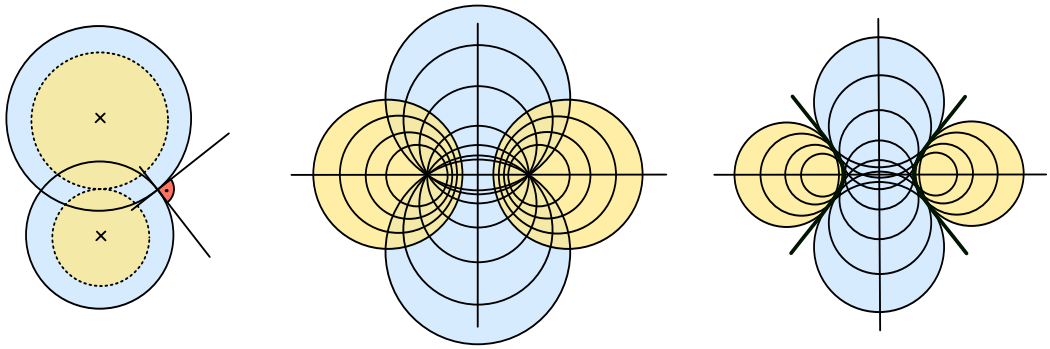


Figure 2.2: *Left: two orthogonal weighted points (blue) which can be scaled with s and $1-s$ such that they touch each other (yellow). Middle: two orthogonal 1-flats along the coordinate axes with the focus in the origin. Right: the scaled flats of the middle image, the intersection of which is a quadric.*

With a small auxiliary calculation one can see, that x^s has still the same position as x, but its weight is $s \cdot w_x$. In addition to the definition of the distance between a weighted point and a point, the distance between two weighted points x and y is defined as $d(x,y) = (p_x - p_y)^2 - w_x - w_y$. The two weighted points are said to be orthogonal if $d(x,y) = 0$. Visually, this means, the corresponding spheres intersect each other and the tangent planes of both spheres through a point on the intersection circle are orthogonal. For two orthogonal points x and y, there exists a scalar value $s \in [0,1]$, with $s \cdot w_y = (1-s) \cdot w_x$, such that the corresponding spheres x^s and y^{1-s} touch each other in a single point (Figure 2.2).

Instead of single weighted points, now sets of weighted points are considered. Let X be the affine hull of $k+1$ affine independent points. Edelsbrunner called such a set *k-flat*. For each k-flat, there exists an orthogonal $(3-k)$-flat Y. This means, each weighted point in X is orthogonal to each weighted point in Y and vice versa. In addition, there exists a point $x \in X$ and a point $y \in Y$ with $p_x = p_y$ and by definition of orthogonality $w_x + w_y = 0$. The 3-dimensional point at this position is called *focus*. Obviously, one of the two spheres is imaginary or both are degenerated. An example of two orthogonal flats and the corresponding focus is illustrated in Figure 2.2. As for single spheres, a whole flat can also be scaled, which is denoted by X^s and scales each sphere in the flat X by s. Furthermore, for two orthogonal flats X and Y, there exists again a scalar $s \in (0,1)$, such that the spheres of the scaled flats X^s and Y^{1-s} touch each other and the intersection is a quadric (Figure 2.2). If X is a k-flat that spans the space of the first k unit vectors, the quadric is given by the implicit surface of the function

$$f : \mathbb{R}^3 \to \mathbb{R} \quad \text{, with} \quad f(p) = -\frac{1}{1-s}\sum_{i=1}^{k} p_i^2 + \frac{1}{s}\sum_{i=k+1}^{3} p_i^2 - w_0,$$

where w_0 is the weight of the point in the focus, which in this case lies at 0. Depending on k and w_0, the implicit surface describes a sphere, a one- or two-sheeted hyperboloid, or for degenerated flats, a double cone or a point.

Surface Composition

The idea of the skin surface is to create for each input sphere a weighted point, whose weight is scaled up depending on the selected shrink factor s. The weighted points define flats in a local neighborhood, which results in quadric surface patches by scaling them down again with s, as described above. The remaining task is to define the neighborhood and the patch boundaries such that the patches fit together in a C^1 continuous surface.

Consider a set of n spheres with positions $p_i \in \mathbb{R}^3$ and radii $r_i \in \mathbb{R}$, with $i \in I = \{1, ..., n\}$. Furthermore, let $s \in (0, 1)$ be the shrink factor. Then, the corresponding weighted points $x_i \in X \subset \mathbb{R}^3 \times \mathbb{R}$, with $i \in I$ are defined by

$$p_{x_i} = p_i \text{ and } w_{x_i} = \frac{r_i^2}{s}.$$

The neighborhood and the flats are defined by the Voronoi diagram and Delaunay complex of the weighted points. Therefore the Voronoi regions are given by the distance functions of the weighted points, so

$$V_i = \left\{ p \in \mathbb{R}^3 \mid \forall j \in I,\ j \neq i,\ d_{x_i}(p) \leq d_{x_j}(p) \right\}.$$

The elements of the Delaunay complex define the flats for the quadric patches. For example, a Delaunay edge defines a subset of the positions of a 1-flat, which is given by the two affine independent weighted points at the end of the edge. The orthogonal 2-flat is represented by the dual Voronoi element, which is in this case a Voronoi face. The focus of two orthogonal flats is given by the intersection of the corresponding affine extended Delaunay and Voronoi elements. So for the previous example, the focus is the intersection of the line, which includes the Delaunay edge and the plane, which includes the Voronoi face. The weight w_0 at the focus can be computed by the weighted points that created the flat. Thus, a quadric patch is uniquely defined by the orientation of the Delaunay and Voronoi elements and the focus with weight w_0. For a Delaunay element D_K, the corresponding implicit surface of the quadric is denoted as F_K.

The last remaining task is to compute the boundaries of the quadric patches. Therefore, Edelsbrunner defined the *mixed complex* as a combination of the Voronoi diagram and the Delaunay complex. Let $V_K \in Vor_X$ be an element of the Voronoi diagram and $D_K \in Del_X$ be the dual Delaunay element. The combination of both, depending on the shrink factor s, is called *mixed cell* and given by

$$M_K = s \cdot V_k + (1 - s) \cdot D_K.$$

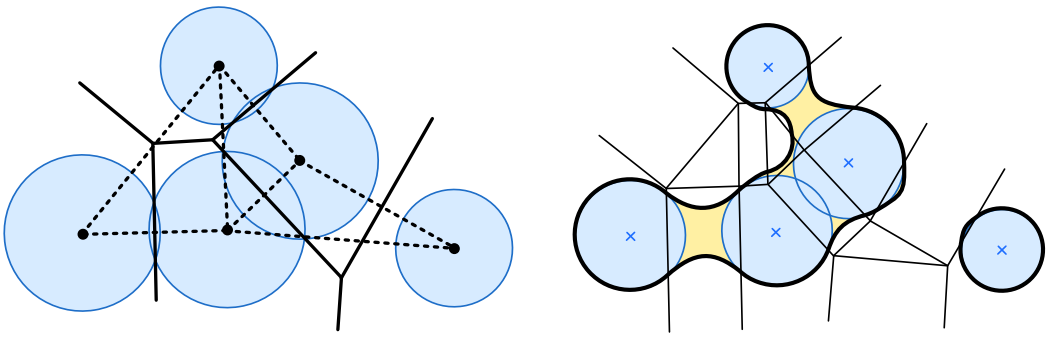


Figure 2.3: *The left illustration shows a set of weighted points (blue) together with its corresponding weighted Voronoi diagram and the dual Delaunay triangulation (dotted lines). On the right side the skin surface is shown together with the mixed complex.*

The complete mixed complex Mix_X is the set of all mixed cells. Note that for $s = 0$, the mixed complex is equal to the Delaunay complex and for $s = 1$ it is equal to the Voronoi diagram. A quadric patch F_K is bounded by the corresponding mixed complex cell M_K. This means, only the parts of the quadric inside the mixed complex cell belong to the skin surface. Formally, we can define the skin surface F_{SS} as

$$F_{SS} = \bigcup_{M_K \in Mix_X} M_K \cap F_K.$$

An illustration of a mixed complex and the skin surface is shown in Figure 2.3. The smaller the shrink factor the more morphs the skin surface to the convex hull of the input spheres. In contrast, the larger the shrink factor the more morphs the shape to the surface of the union of the input spheres. In Figure 2.4, a skin surface is visualized with different shrink factors.

2.2 Molecules

In general, a molecule is a stable arrangement of atoms. An atom consists of a hull and a nucleus. While the nucleus contains the electrically neutral neutrons and positively charged protons, the hull contains the negatively charged electrons. Over 99.9 % of the atom's weight is stored in the nucleus. However, most atomic interactions are triggered by the electrons. The arrangement of the atoms in a molecule can describe different structures. For example, biomolecules like proteins often build some kind of chain. The stability of the arrangement is mainly achieved by chemical bonds between the atoms. In case of biomolecules, the strongest bonds are *covalent bonds*. Two atoms are connected by a covalent bond if they share one or more electron pairs. The sharing leads to an attraction by the negatively charged electrons which overcomes the repulsion of the positively charged atom nuclei. This holds the two atoms in a stable arrangement with minor fluctuations. In classical molecular dynamics simulations, these fluctuations are modeled by harmonic

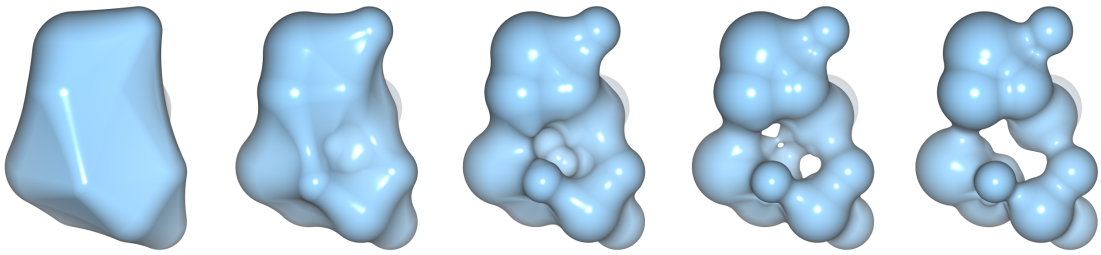


Figure 2.4: *Skin surface of a set of* 15 *random spheres for different shrink factors.*

springs for bond stretching, bond angle bending, and bond twisting motions. The two other strong chemical bonds are *metallic* and *ionic bonds*, which occur, for example, between atoms or molecules in lattice structures, like crystals. In addition to these strong bonds, there also exist weak bonds like *van der Waals interactions* or *hydrogen bonds*. Especially the latter ones play an important role for the shape and function of proteins. However, weak bonds may be constantly broken and created. A more detailed description of van der Waals interactions and hydrogen bonds is given in the Sections 2.2.1 and 2.2.2, respectively.

For the following descriptions, consider a molecule with n atoms. Each atom can be described by its type, which is often represented by the atomic number in the periodic table of elements and several other properties. The most important properties for the visualization and geometrical analysis are the atom positions $p_i \in \mathbb{R}^3$ and the atom radii $r_i \in \mathbb{R}$ with $1 \leq i \leq n$. Note that this description only represents a molecule at an arbitrary but fixed point of time. However, molecules are dynamic structures, so the atom positions are functions of time $p_i(t) : \mathbb{R} \longrightarrow \mathbb{R}^3$, where t represents the time. With the atom positions and the radii, each atom can be modeled as a *hard sphere*. Throughout this thesis, the radii are constant over time as well as the number of atoms in the molecules. The definition of the atom radii depends on the purpose of the analysis and visualization, but most often, van der Waals radii are used.

2.2.1 Van der Waals Radii

Johannes Diderik van der Waals was the first who recognized that atoms have a finite size. By modeling atoms and complete molecules as hard spheres with attractive and repulsive interactive forces, he developed an equation of state for real gases and liquids. Note that these forces describe the non-covalent interactions between atoms and that they are much weaker than the forces given by covalent bonds. Consider two non-bonded atoms. The van der Waals interactions between these two atoms can be modeled by the Lennard-Jones potential (Figure 2.5). For distances between 3 Å and 4 Å, the atoms start to attract each other. The attraction increases with decreasing atom distance. At a certain distance, the outer electron regions of both atoms start to

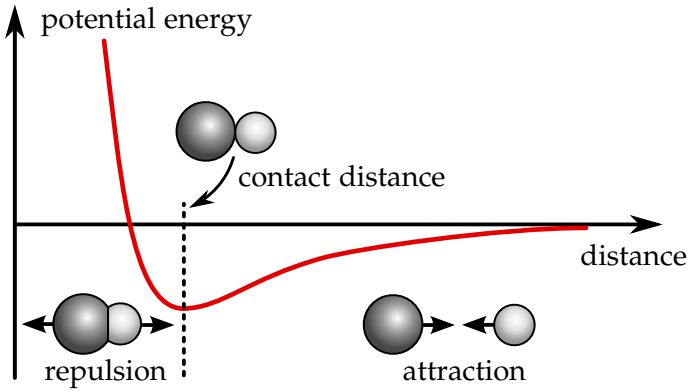
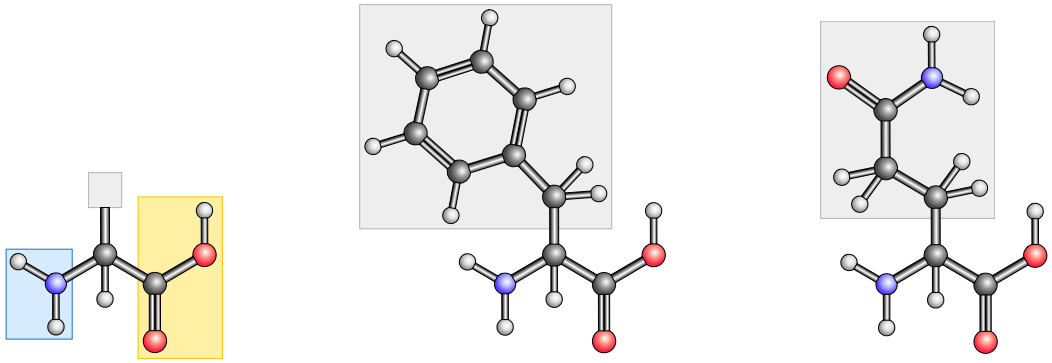


Figure 2.5: *Illustration of the van der Waals forces between hydrogen and carbon. The forces are modeled by the Lennard-Jones potential.*

overlap each other, which results in a strong repulsive force [16]. The distance where attractive and repulsive forces balance each other is defined as *van der Waals contact distance*. This distance is often used to determine the van der Waals radii of the atoms. Note that there is no single definition, but several approaches to determine the radii from the contact distance. Often, the radii are computed based on the properties of the elements such that the sum of the radii is equal to the contact distance. Since the Lennard-Jones potential depends on the pair of atoms, this leads, for example, to a different radius for a hydrogen atom which interacts with an oxygen atom than a hydrogen atom which interacts with a carbon atom. Often these different radii for the same element are combined in a single radius. Overall, the concept of van der Waals radii is an imaginary construct to model the physical size of an atom as hard sphere. A short overview of the determination of atom radii is given in the following.

Since the PhD thesis of van der Waals (1873), there has been a long history on determining atom radii. Important landmarks are the work of Bragg [27], presenting radii for major chemical elements (and even visualizing crystal structures, composed of atoms depicted as spheres), and the book of Pauling [181], giving a general account of different types of atom radii in bonded (covalent, metallic, and ionic) and non-bonded states. Slater [218] presented improved sets of empirical atom radii, such that the sum of the radii of two atoms forming a bond in a crystal or molecule gives an approximate value of the internuclear distance. Intermolecular van der Waals radii of the non-metallic elements have been assembled into a list of recommended values for volume calculations by Bondi [23]. Various methods have been devised – mechanical, crystallographical, electrical, optical and computational – to empirically determine van der Waals radii. They give similar, but different values. For physical reasons, van der Waals radii vary with the particular chemical environment, see e.g. Bondi [23]. Rowland et al. [204] presented new results considering the non-bonded contact distances in organic crystals. They recognized that the radii defined by Bondi are quite similar to their results and



(a) (b) (c)

Figure 2.6: *Illustration of the general structure of an α-amino acid (a) and the structure of Phenylalanine (b) and Glutamine (c). The amine group is highlighted by the blue box, the carboxylic group by the yellow box, and the side chain by the gray box. The three groups are connected by the α-carbon C_α.*

only differ for a few elements. An extensive overview of van der Waals radii of elements was given by Batsanov [13]. In a more recent work, Batsanov [14] extended this work by atom radii for metals, derived from empirical data and an equation of state for solids.

2.2.2 Hydrogen Bonds

A *hydrogen bond* is an attractive interaction between two electronegative atoms that share a hydrogen atom. The hydrogen atom is connected by a covalent bond to one of these atoms, which is called *hydrogen donor* while the other one is called *hydrogen acceptor*. Typical electronegative atoms are oxygen, nitrogen, and fluorine. A hydrogen bond between, for example, nitrogen and oxygen can be written as $N - H \cdots O$, where N is the hydrogen donor and O the acceptor. The strength depends mainly on three conditions: the pair of electronegative atoms, the distance between the atoms and the arrangement of the atoms. A hydrogen bond is strongest for atoms in collinear position, which means the 3 atoms lie on a line. In general, hydrogen bonds are stronger than van der Waals interactions but much weaker than covalent bonds. Hydrogen bonds are important for the stability and function of a molecule.

2.2.3 Protein Structure

In this thesis the focus mainly lies on the analysis of proteins, a specific group of biomolecules. For this reason, a short description of the structure of proteins is given here. Additional details can be found in the book by Stryer et al. [16]. Proteins belong to the group of macromolecules, which can consist of one or more molecules. These molecules are often called *monomers*. Typically, the number of atoms in a monomer ranges from a few hundred up to several thousand atoms. The atoms in each monomer are arranged in a

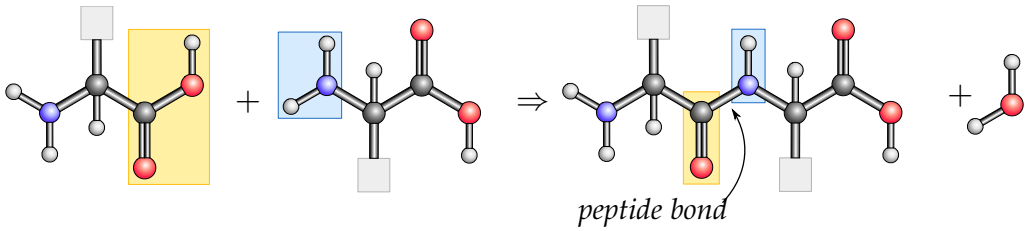


Figure 2.7: *Construction of polypeptide chains in proteins. The connection between the carboxylic group (yellow box) of one amino acid with the amine group (blue box) of another amino acid results in a peptide bond and a free water molecule. The side chains are represented by the gray boxes.*

chain of $\alpha-$*amino acids*. An *amino acid* is an organic compound that consists of three main parts: an amine group $-NH_2$, a carboxylic group $-COOH$, and a side chain. The class of α-amino acids is characterized by the arrangement of the carboxylic group and the amine group. These groups are connected by covalent bonds over the first carbon atom of the amino acid. For this reason, this carbon atom is often denoted as α-carbon C_α. The general structure of an α-amino acid and two specific examples are illustrated in Figure 2.6. Until now, 23 different α-amino acids are known in proteins.

In order to build the chain of a monomer, the amino acids get linked by *peptide bonds*. A peptide bond is a covalent bond, which connects the carboxylic group of one amino acid with the amine group of another amino acid by loss of a water molecule (Figure 2.7). This can be extended by connecting further amino acids at both ends to form the chain of a monomer. Such a chain is called *polypeptide chain* and the amino acids that are linked in this chain are called *residues*. A polypeptide chain can be decomposed into the *backbone* and the already mentioned side chains. The backbone consists of repeating groups of atoms that are equal in each residue. It forms the skeleton of the protein. In 1952, Ulrik defined 3 levels of structural descriptions for proteins [230]. Today, 4 levels will be distinguished.

Primary Structure

The primary structure of a protein describes the composition of the polypeptide chains. It is given as a set of sequences of residues. The first residue of each sequence has a free amine group and is denoted as *amine-terminal* or *N-terminal*. Respectively, the last residue has a free carboxylic group and is denoted as *carboxylic-terminal* or *C-terminal*. An excerpt of the primary structure of the protein with *pdb:1OGZ* is

$$...-ALA-VAL-VAL-GLN-ARG-TYR-VAL-...$$

Here ALA denotes Alanine, VAL Valin, GLN Glutamine, ARG Arginine, and TYR Tyrosine.

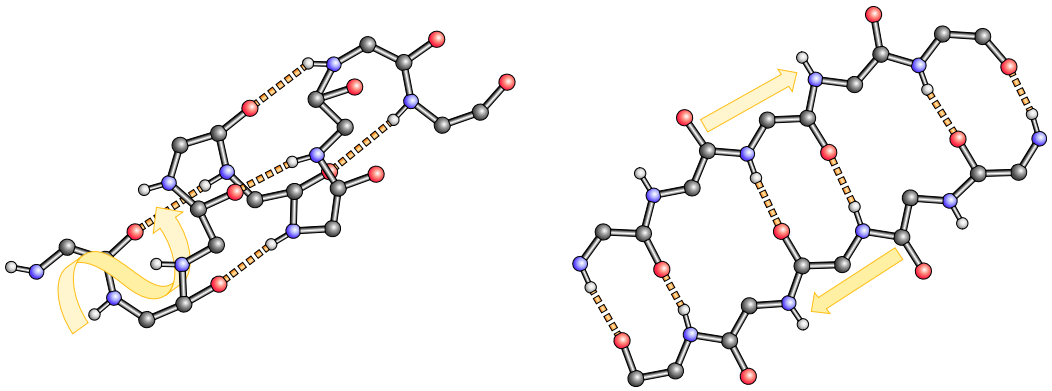


Figure 2.8: *Illustration of an α-helix (left) and a β-sheet (right). Only a part of the backbone is shown here and the side chains are hidden. The hydrogen bonds are depicted by the dashed lines in orange. The β-sheet consists of a single anti-parallel ladder.*

Secondary Structure

The secondary structure describes regular patterns of hydrogen bonds between amine and carboxylic groups of the backbone. These patterns were first defined by Pauling et al. [182, 183]. An extended definition and overview was given by Kabsch and Sander [102], which is summarized here. The main patterns of hydrogen bonds are *α-helices* and *β-sheets*. Assume the residues are enumerated over all polypeptide chains of a protein in the primary structure, starting with 1 for the first residue of the first chain till m for the last residue of the last chain. Furthermore, let H be the set of all hydrogen bonds between backbone atoms, where each hydrogen bond $h \in H$ is a tupel of $\{1, ..., m\} \times \{1, ..., m\}$. Additionally, the first element of the tupel always represents the residue with the oxygen atom, which is bonded to the nitrogen atom of the residue, represented by the second element.

A *k-turn* at position i is a hydrogen bond $(i, i+k) \in H$, with $k = 3, 4, 5$. Two consecutive k-turns build a minimal *k-helix* of length k. Formally, the minimal k-helix between the residues i and $i+k-1$ is defined by the two k-turns at the positions $i-1$ and i. Note that it is not necessary that there exist further k-turns as for example at position $i+1$. Two or more overlapping minimal helices build a longer helix. Typically, 3-, 4-, and 5-helices are called 3_{10}-helices, α-helices, and π-helices respectively. An illustration of an α-helix is given in Figure 2.8 (left). This type of helices occurs most frequently in proteins.

Apart from these helices, bridges of hydrogen bonds are defined, as described in the following. Let therefore, $(i-1, i, i+1)$ and $(j-1, j, j+1)$ be two non-overlapping triples of residues. A bridge consists of exactly two hydrogen bonds and two types will be distinguished that are represented by a tupel of residues:

1. A parallel bridge (i, j) is defined by the two bonds $(i-1, j), (j, i+1) \in H$ or $(j-1, i), (i, j+1) \in H$.

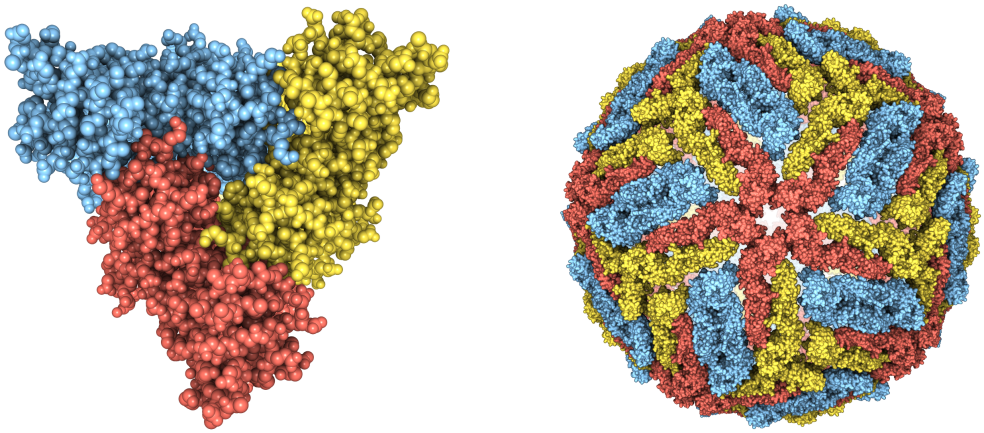


Figure 2.9: *The 3-fold rotational symmetry of the polypeptide chains of the sodium ion channel (pdb: 3HGC, left) and the 180 chains of the hull of the dengue virus (pdb: 1K4R, right).*

2. An anti-parallel bridge (i, j) is defined by the two bonds $(i, j), (j, i) \in H$ or $(i-1, j+1), (j-1, i+1) \in H$.

A set of one or more consecutive bridges of the same type is often called *ladder*. For the parallel case, the bridges (i, j) and $(i+1, j+1)$ are consecutive and for the anti-parallel case the bridges (i, j) and $(i+1, j-1)$. A β-sheet is a set of one or more ladders that share a subset of residues. Note that a sheet can consist of both types of ladders, parallel and anti-parallel. Figure 2.8 (right) illustrates a β-sheet that consists of a single anti-parallel ladder.

Overall, the secondary structure of a protein is a formal description of all patterns of hydrogen bonds. It is important to mention that the secondary structure does not contain any 3-dimensional information about the atoms or bonds. There exist several different suggestions for the formal description of the secondary structure. For example, the α-helices can be described as a set of tupels, that represent the start and end residues of the helices. In contrast, the β-sheets can be stored as sets of ladders, where each ladder can be described as a tuple of start and end bridge.

Apart from giving a definition, Kabsch and Sander developed a tool, called DSSP [102], to compute the secondary structure based on the atom positions. A similar program was presented by Frishman and Argos, which they called STRIDE [69, 87]. In contrast to the DSSP algorithm, which only considers the hydrogen bond energy, STRIDE also analyzes the dihedral angles along the backbone. This is done using Ramachandran plots [198].

Tertiary Structure

This level describes the 3-dimensional structure of the protein. Most often it is given by the atom positions in Cartesian coordinates. Another description of the tertiary structure of a molecule is given by the covalent bond lengths, the bond angles and the dihedral angles between the atoms. This representation

is often named inner coordinates. While the transformation from Cartesian to inner coordinates is straightforward the other way around requires some considerations to provide numerical stability with good performance. One of the most efficient algorithms for this purpose was described by Parsons et al. [176].

Quaternary Structure

The quaternary structure describes the spatial arrangement of the polypeptide chains of a protein. Most proteins are composed by more than one polypeptide chain whose arrangement can be often described by symmetry operations. For example, the sodium ion channel *pdb:3HGC* consists of 3 equally structured chains in a 3-fold rotational symmetry, (Figure 2.9). Proteins that consist of two or more chains are often called *multimers*. In case of two chains the protein is denoted as *dimer* and in case of three chains as *trimer*. While most proteins consist of up to 8 chains, the proteins of many virus capsids consist of a multiple of 60 chains. An example of such a hull for the dengue virus is shown in Figure 2.9.

2.3 Molecular Visualization Models

Many visual representations have been developed to illustrate the different types of structures that were described in the previous section. Without these visual models it is almost impossible to analyze the structure of large molecules. In this section, the most commonly used models and their practical applications are described. Especially, molecular surfaces will be highlighted, the visualization of which is one focus of this thesis. An overview about biomolecular visualization was presented in "*Visualization of Biomolecular Structures: State of the Art*" [117] in 2015.

2.3.1 Ball-and-Stick

The ball-and-stick representation comprises the visualization of the primary, tertiary and quaternary structure. The classical model represents each atom as a small sphere and each covalent bond by a small cylinder connecting the two corresponding atom spheres. Sometimes double or triple bonds are visualized using two or three cylinders, respectively. Most often the color of the atoms and cylinders represent the type of the atoms, but in general the colors can represent an arbitrary atom or bond attribute. The ball-and-stick model can be extended by visualizing also the hydrogen bonds as regularly discontinued cylinders connecting the hydrogen atom of the donor with the acceptor atom. With this extension, also the secondary structure is shown. In modern visualizations, the sphere radii are often equal to the radii of the cylinder, which results in a more compact representation, called licorice or

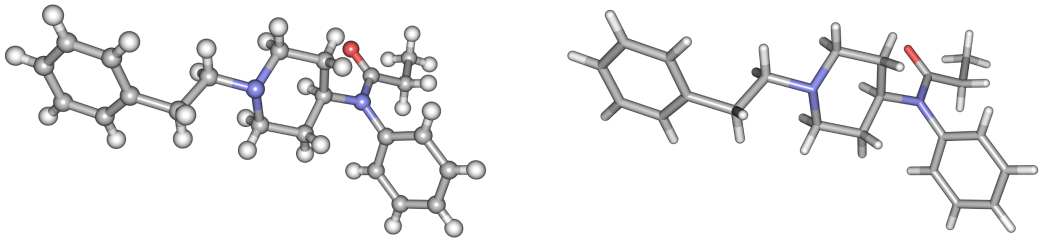


Figure 2.10: *Fentanyl, depicted by its ball-and-stick representation in the classical style (left) and the modern style (right). The colors represent the atom types.*

stick model. An example of the ball-and-stick model for fentanyl is shown in Figure 2.10.

2.3.2 Secondary Structure

Although it is possible to visualize the secondary structure with the ball-and-stick representation, the image would be too overloaded for large molecules to get a quick overview of all α-helices and β-sheets. For this reason, particular visualizations have been developed showing abstract representations of the secondary structure. They are also called "secondary structure", which might confusing, because this term is already used for the formal description, which includes only the patterns of hydrogen bonds and not the 3-dimensional structure. The main reason for this is, that the focus of the visual secondary structure representation lies on the illustration of the formal secondary structure, although it includes also the other structures. In this thesis, the visualization model of the secondary structure is the one that is more often used, so if not explicitly mentioned the term "secondary structure" is related to the visual representation.

The secondary structure model illustrates the spatial shape of the backbone and emphasizes the patterns of the formal secondary structure. Note that there is no clear definition of this visualization model. Hence, the representations of the patterns vary from program to program. However, most of them are based on the illustrations by Richardson [202]. Here, two of the most often used secondary structure representations are described in detail. The main part of the representations consist of 3-dimensional curves. Each curve is oriented along the carbon and nitrogen atoms of the backbone of a polypeptide chain. Commonly, one atom per residue is used as control point for an interpolation which generates a smooth 3-dimensional curve. In order to render the curve, it is discretized and approximated by a piecewise linear curve. Typically the linear pieces are visualized by cylinders with equal radii. Furthermore the cylinders are cut in a way that two neighboring cylinders exactly fit together. There are several possibilities to emphasize the secondary structure patterns along the backbone. A simple but effective way is, to deform the circular cross section of the cylinders into an elliptic crossection for α-helices and β-sheets. For this kind of visualization, the larger diameter of the ellipse

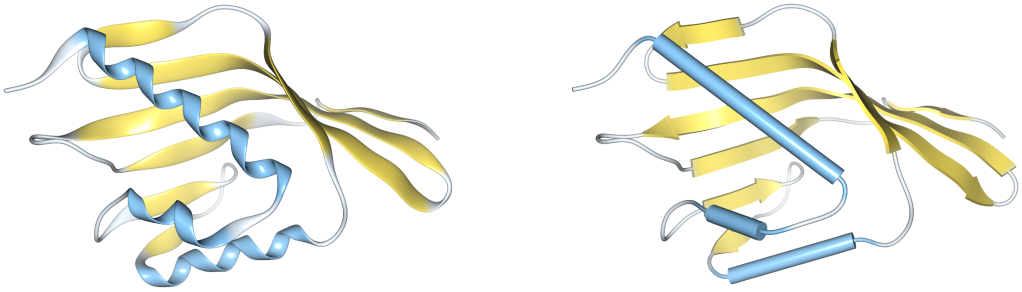


Figure 2.11: *Illustration of the secondary structure of the protein ketosteroid isomerase (pdb:1OGZ) by the ribbon representation (left) and the cartoon representation (right). The α-helices are depicted in blue and the β-sheets in yellow.*

is oriented into the direction of the hydrogen bond and the smaller remains equal to the diameter of the cylinder. This type of visualization is sometimes denoted as *ribbon representation* (Figure 2.11). Additionally, the direction of the β-sheets can be visualized by placing arrow icons onto the elliptic surface. The arrows allow one to quickly distinguish between parallel and anti-parallel ladders in a β-sheet. An even more abstract representation type of the secondary structure is the *cartoon representation*. Here, the 3-dimensional curve is discontinued for α-helices and β-sheets. The discontinuities are filled by cylinders for α-helices and 3-dimensional arrows with rectilinear cross section for β-sheets. Similar to the ribbon visualization, the arrows are wider into the direction of the hydrogen bonds. An example of the cartoon representation is shown in Figure 2.11. In order to visualize the secondary structure interactively for dynamic proteins of simulations, Krone et al. [119] presented a fast GPU-based approach.

2.3.3 Molecular Surfaces

The main purpose of molecular surface models is to visualize the tertiary and quaternary structure of a molecule. Of course these structures can be already visualized with the ball-and-stick or secondary structure representation, however both representations are not suitable to visualize the spatial dimension of a molecule. In contrast, molecular surfaces are particularly designed to model the shape of a molecule by approximating the size of bonded and non-bonded atoms. This is important in order to analyze regions around and inside the molecule that can be possibly reached by other molecules. Thus molecular surface representations are often used to investigate the results of protein-substrate docking simulations. In the following, a description of the most often used molecular surface models is given. Therefore, consider a static molecule with n atoms, whose positions are $p_i \in \mathbb{R}^3$, with $i \in I = \{1, ..., n\}$.

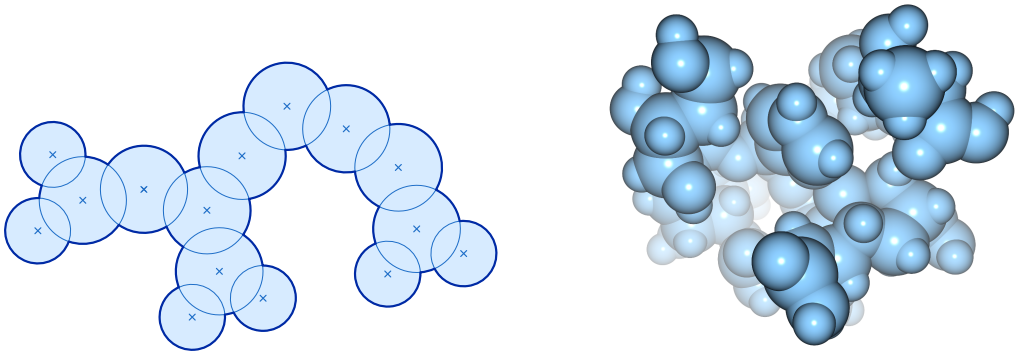


Figure 2.12: *Illustration of the van der Waals surface in 2D (left) and the 3-dimensional surface representation of copper(I)-bleomycin (pdb: 1UGT, right).*

Van der Waals Surface

The *van der Waals surface* [72, 201] is probably the most often used molecular surface representation. Besides, it is one of the simplest surface models and it is the basis of most of the other molecular surface representations. As the name reveals, each atom is modeled as a sphere with the corresponding van der Waals radius $r_i \in \mathbb{R}$, with $i \in I$. The surface is then defined as the outer surface of the union of all atom spheres. In detail, it consists of all points that lie at least on the surface of one atom sphere but not inside of any atom sphere (Figure 2.12). Formally, the van der Waals surface can be described as implicit surface F_{vdW} of the function

$$f_{vdW} : \mathbb{R}^3 \to \mathbb{R} \quad \text{, with} \quad f_{vdW}(p) = \min_{i \in I} \|p - p_i\| - r_i.$$

In 1971, Lee and Richards [139] presented a program to draw the van der Waals surface of a molecule.

Solvent Accessible Surface (SAS)

Apart from the rendering of the van der Waals surface, Lee and Richards defined one of its first extensions, which later became known as the *solvent accessible surface* (SAS) [139]. The idea of this surface is to show all regions in a molecule that can be accessed by a solvent molecule. In case of protein visualization, the most interesting solvent is water because proteins are usually surrounded by water. The size and shape of the complete solvent molecule is approximated by a single sphere, which is called probe. The radius of the probe is denoted as r_p. Commonly, a probe with a radius of 1.4 Å is used to approximate a water molecule. The solvent accessible surface is equal to the van der Waals surface except that each van der Waals radius is extended by the radius of the probe. Formally the SAS can be described as implicit surface F_{SAS} of the function

$$f_{SAS} : \mathbb{R}^3 \to \mathbb{R} \quad \text{, with} \quad f_{SAS}(p) = f_{vdW}(p) - r_p.$$

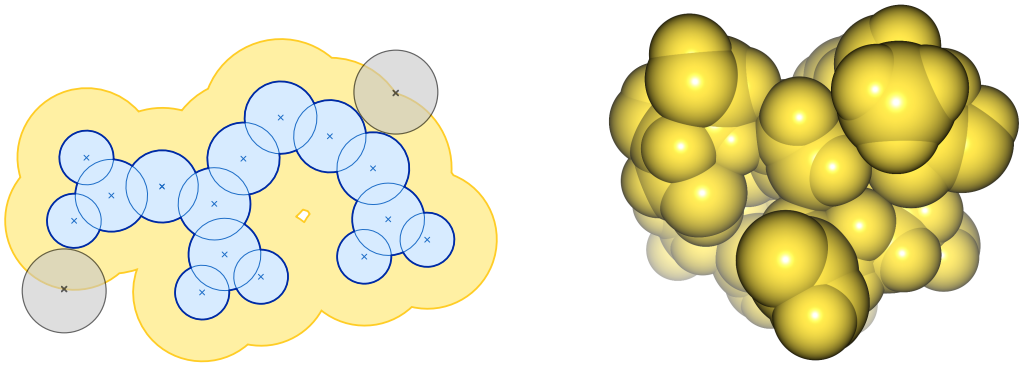


Figure 2.13: *Illustration of the solvent accessible surface together with the probe in 2D (left) and the 3-dimensional surface representation of copper(I)-bleomycin (pdb: 1UGT, right).*

Note that r_p is a user defined parameter to approximate the solvent of interest. Visually, the SAS describes the center of the probe, which rolls over the van der Waals surface. During this process, the probe always touches the van der Waals surface but never penetrates it. Hence, all points outside the surface can be geometrically accessed by the center of the probe and thus probably also by the solvent (Figure 2.13).

Solvent Excluded Surface (SES)

In 1977, Richards [201] defined the first smooth molecular surface based on the idea of the SAS. But instead of taking the center of the probe which rolls over the atoms, he suggested to use the track of the outer shell of the probe (Figure 2.14). This combines the advantages of both surfaces, the better size representation of the atoms of the van der Waals surface and the accessibility visualization of the SAS. He defined the points where the probe touches the van der Waals surface as *contact surface* and the remaining parts of the track of the probe as *reentrant surface*. Richard called this surface also SAS, but nowadays the term *solvent excluded surface* (SES) is preferred, the name of which was proposed by Greer and Bush [76] in 1978. To give a simple formal description of the SES, let d_{SAS} be the signed distance function of the corresponding SAS, so

$$d_{SAS} : \mathbb{R}^3 \to \mathbb{R} \ \text{, with } \ d_{SAS}(p) = sgn(f_{SAS}(p)) \cdot \min_{q \in F_{SAS}} \|p - q\| \, .$$

Based on this distance function, the SES for a probe with radius r_p can be defined as the implicit surface F_{SES} of the function

$$f_{SES} : \mathbb{R}^3 \to \mathbb{R} \ , \ f_{SES}(p) = d_{SAS}(p) + r_p.$$

While this formal definition is very compact, in practice it is easier to visualize the surface by decomposing it into its 3 types of patches. These types can be identified by 3 different cases while the probe rolls over the van der Waals surface:

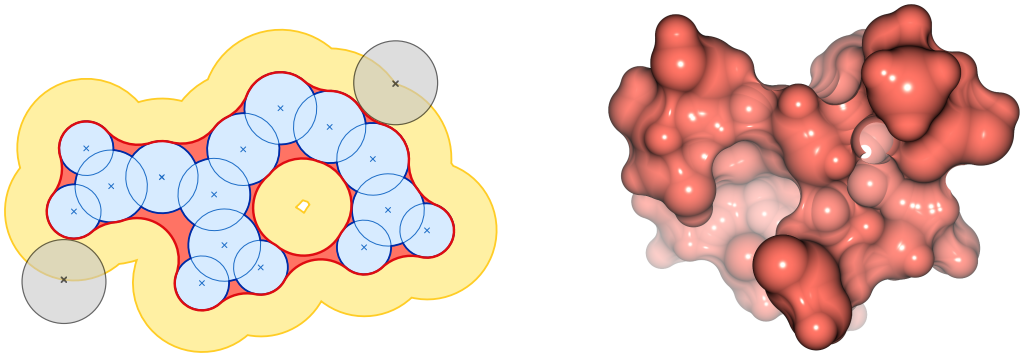


Figure 2.14: *Illustration of the solvent excluded surface in 2D (left) and the 3-dimensional surface representation of copper(I)-bleomycin (pdb:1UGT, right). The probe is depicted in gray.*

1. The probe touches the van der Waals surface in exactly one point. These points are a subset of the van der Waals atom spheres and describe spherical patches. They are called *convex spherical patches*, because the outer parts of the spheres are visible for the user.

2. The probe touches the van der Waals surface in exactly two points. This means the probe rolls between two atoms. The track of the probe describes the inner part of a torus, for this reason these patches are called *toroidal patches*.

3. The probe touches the van der Waals surface in three or more points. This means the probe is in a fix position. The track of the probe is in this case an inner part of the surface of the probe sphere. It is bounded by spherical arcs through the touching points. These patches are called *concave spherical patches* because the inner parts of the spheres are visible.

At the patch boundaries, where two ore more patches fit together, the surface is C^1 continuous, so that the SES is a smooth surface. However, the surface can contain self-intersections. At these intersections the surface has sharp edges and is only C^0 continuous. Two different types of self-intersections occur when the atoms lie too far away from each other. The first type is the self-intersection of toroidal patches. This type occurs when the probe rolls between two atoms and intersects the imaginary line through the two atom positions. This creates two sharp tips (Figure 2.15, left). The second type occurs when two ore more concave spherical patches intersect each other. Two different examples for this kind of intersection are shown in Figure 2.15 (middle, right). In contrast to the first type, it is more difficult to detect these intersections.

Molecular Skin Surface (MSS)

The *molecular skin surface* (MSS) [54] is the application of the skin surface (Section 2.1.4) to the van der Waals spheres of the atoms with the purpose to

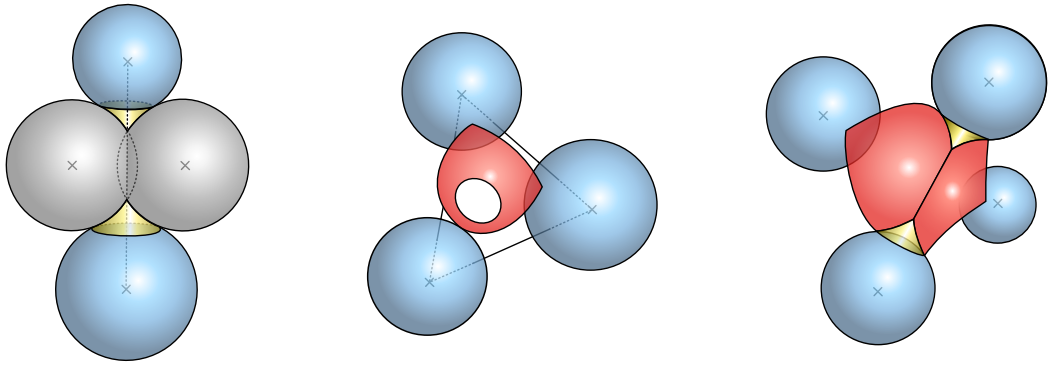


Figure 2.15: *Illustration of singularities of the SES. A self-intersection of a toroidal patch (left), an intersection of two concave spherical patches (middle), and an intersection of two concave spherical patches between a self-intersecting toroidal patch (right).*

approximate the SES. The main advantage of the MSS in contrast to the SES is that the surface is completely C^1 continuous. Furthermore, it can be decomposed into patches of quadrics, that can be easily visualized. On the other hand, the surface has not a bio-physical background and it is not trivial and sometimes even not possible to select a suitable shrink factor to approximate the solvent probe sphere. An example of the MSS can be seen in Figure 2.16 (left).

Kernel-based Molecular Surfaces (KMS)

In 1982, Jim Blinn presented a new technique to visualize a finite set of weighted points as an organic looking surface, called *metaballs* [18]. The idea is to create some kind of density function for the weighted points and to render the implicit surface of this function. Therefore, the density function is described as a sum of density kernels representing the weighted points. Each kernel function maps an arbitrary point $p \in \mathbb{R}^n$ to a density value in $\mathbb{R}$. Typically, the kernel function has its maximum at the point position and monotonically decreases to 0 with increasing distance of p. The *kernel-based molecular surfaces* (KMS) are a subset of metaballs. These surfaces have all in common, that they use a density kernel to approximate the size of a single atom. The main purpose of these surfaces is again to approximate the SES or sometimes even an isosurface of the electron density. Formally, a KMS can be described as the implicit surface of a density function f_{KMS}, with

$$f_{KMS} : \mathbb{R}^3 \to \mathbb{R} \ , \ f_{KMS}(p) = \sum_{i=1}^{n} k_i(p) - c,$$

where $k_i : \mathbb{R}^3 \to \mathbb{R}$ is the kernel function and c is a constant to steer the size of the shape of the surface. Typical kernel functions are:

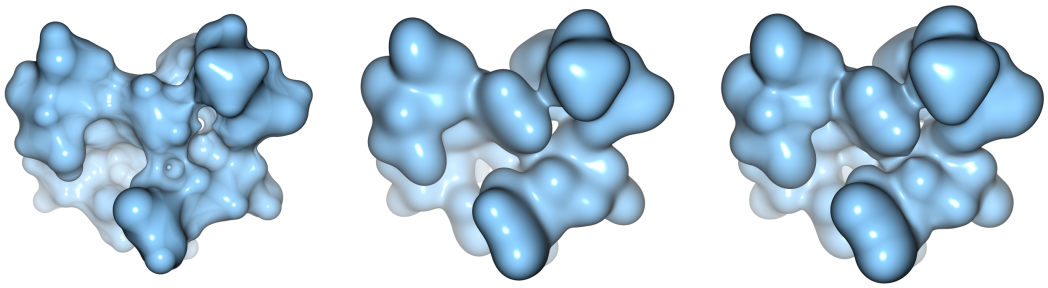


Figure 2.16: *The molecular skin surface (left) and two kernel-based molecular surfaces (middle, right) of copper(I)-bleomycin (pdb: 1UGT). While for the middle surface a Gaussian kernel was used, the right surface was created by the soft object function.*

- the inverse squared function with a parameter $a_i \in (0, 1]$

$$k_i(p) = \frac{a_i}{(p - p_i)^2}$$

- the Gaussian kernel, where d is the standard deviation, and a_i is a scale of the ith atom.

$$k_i(p) = a_i e^{-\frac{(p-p_i)^2}{2d^2}}$$

- the soft object function [245] with $d = p - p_i$ and the parameters a_i, and b.

$$k_i(p) = \begin{cases} a_i \left(1 - \frac{4d^6}{9b^6} - \frac{17d^4}{9b^4} - \frac{22d^2}{9b^2}\right), & d \leq b \\ 0, & d > b \end{cases}$$

The parameters are often selected visually and depend on the purpose of the user. While the first two kernel functions have to be evaluated for each atom at each position, this is not necessary for the soft object function, which becomes 0 at a certain distance. For this reason, the soft object function can be computed much faster using 3-dimensional data structures. However, since most kernel functions decrease fast with increasing distance, it is also possible to select a cutoff value, at which the function becomes 0. Although the kernel functions are not continuous anymore, the resulting surface is still visually smooth. The most often used kernel function for KMS is the Gaussian function [18]. Depending on discretization of the density function, KMS can be quickly computed and provide a smooth surface of the molecule. However, similar but even more difficult than for the MSS is the setup of the parameters. Grant and Pickup presented a good parameter choice for the Gaussian kernel function to approximate the SES with a probe of water [74]. Two different KMS are shown in Figure 2.16.

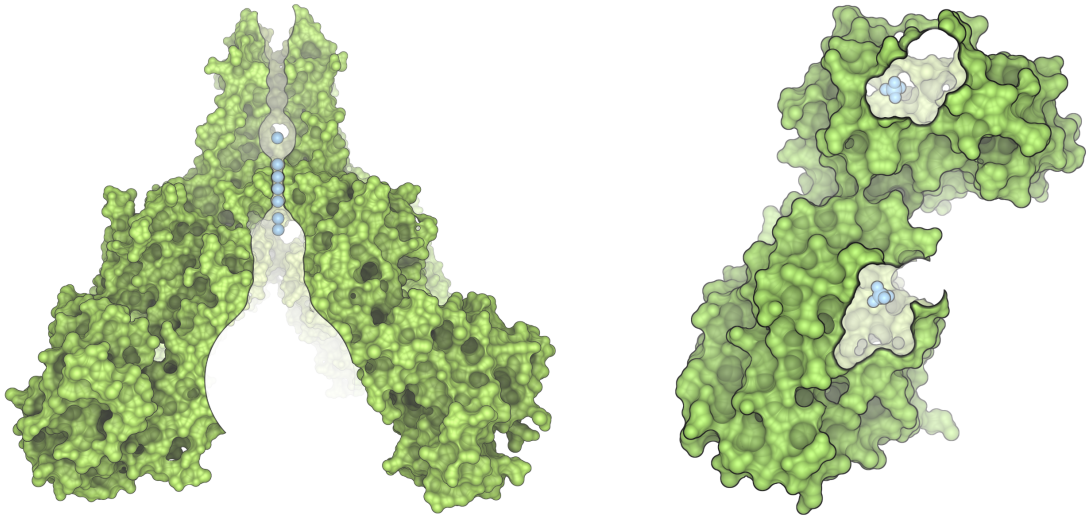


Figure 2.17: *Molecular surface cuts showing the structure of the potassium channel pdb:1K4C (left) and two molecular pockets of the enzyme pdb:1PII (right).*

2.4 Molecular Paths and Cavities

In addition to the classical molecular visualizations, it is as least as important to investigate the complementary space of the atoms. This space consist of all regions inside and around the molecule where no atomic structures of the investigated molecule can be found. These regions are often called cavities. They can be potentially accessed by ions or small other molecules, called substrates or ligands, which can result in molecular interactions (Figure 2.17). Such interactions are important in many molecular research topics. Although there are many algorithms to compute molecular cavities, there does not exist a clear formal definition for these structures. Often they are defined implicitly by the developed algorithms. In this section, a formal definition of molecular paths and cavities is proposed.

In general a molecular path is a path of a small molecule or ion in the region of a larger molecule. This could be, for example, a path of a substrate to its binding site or the path of an ion through a tunnel of a membrane protein. Note that there is no restriction for the start state of the small molecule. However, both molecules are dynamic structures, which makes the paths time-dependent. Furthermore, a molecular cavity is defined as the continuous volumetric space that can be accessed by the small molecule. Thus, each cavity is described by the space around possible molecular paths that are connected. Additionally, cavities usually require the definition of a volumetric boundary based on the large molecule that separates inside and outside. Without this boundary, all channels and pockets would belong to the same cavity, because they are connected by paths outside the large molecule (Figure 2.19). In contrast to the formal description of paths and cavities, it is quite difficult to define the boundary, because it depends on the application.

2.4.1 Formal Definition

Let X be the current state of a molecule. It includes all properties to describe the molecule based on the underlying physical model. For example, for the classical physical model, the state includes the atom positions and electrostatic potentials as well as the bonding and non-bonding forces. If the molecule changes over time to another state Y, for the following definitions, it will be assumed that a continuous parametric function exists that connects these two states.

Consider two molecules, a larger one, which could be a protein and a smaller one, which could be a substrate, solvent, or ion. First, observe the static state $\hat{X}$ of the large molecule. Let $S_{\hat{X}}$ be the set of all states the smaller molecule can adopt under the influence of the large molecule in state $\hat{X}$. A *molecular path* is then defined as a parametric continuous curve c in the space of $S_{\hat{X}}$. So for a path between $X_1 \in S_{\hat{X}}$ and $X_2 \in S_{\hat{X}}$, the curve c can be defined as

$$c : [0,1] \subset \mathbb{R} \to S_{\hat{X}} \ \text{, with } \ c(0) = X_1, \ \ c(1) = X_2.$$

Furthermore, let $b_{\hat{X}}$ be a tertiary boundary function that evaluates if a state of the small molecule lies inside or outside of the region of the large molecule or if it lies on the boundary. The restriction $\tilde{S}_{\hat{X}} \subset S_{\hat{X}}$ is the set of all states $X \in S_{\hat{X}}$ for which $b_{\hat{X}}(X)$ evaluates the state as inside or on the boundary. In the nondegenerate case, $\tilde{S}_{\hat{X}}$ consists of a network of paths with one or more connected components.

Now consider the spatial region $V_{\hat{X},X}$ in $\mathbb{R}^3$ representing the volume of the small molecule in state X under the influence of the large molecule in state $\hat{X}$. Note that a unique formal definition of $V_{\hat{X},X}$ is not available. Again it depends on the underlying physical model. However, reasonable heuristics to approximate $V_{\hat{X},X}$ exist. For a state $\hat{X}$ of the large molecule, a *molecular cavity* is defined as the union of all volume sets whose corresponding states are connected by molecular paths in $\tilde{S}_{\hat{X}}$. Note that two cavities can possibly intersect each other, but there does not exists a molecular path between any two states of two different cavities.

Consider now the case where the large molecule is dynamic, i.e., $\hat{X}$ is a function of time $\hat{X}(t)$. Let $X_1 \in S_{\hat{X}(t_1)}$ and $X_2 \in S_{\hat{X}(t_2)}$ be two valid states of the small molecule for different times. A *dynamic molecular path* between X_1 and X_2 is defined as a time-dependent continuous function c, with

$$c : [t_1, t_2] \subset \mathbb{R} \to S_{\hat{X}(t)} \ \text{, with } \ c(t_1) = X_1, \ \ c(t_2) = X_2.$$

Furthermore, a *dynamic molecular cavity* is defined as the union of all $V_{\hat{X}(t),X}$ that are connected either by a dynamic molecular path or by a molecular path. Thus, four possible topological events can be distinguished for the change of a molecular cavity over time. It can appear and disappear, or it can merge into another cavity or split into two or more cavities.

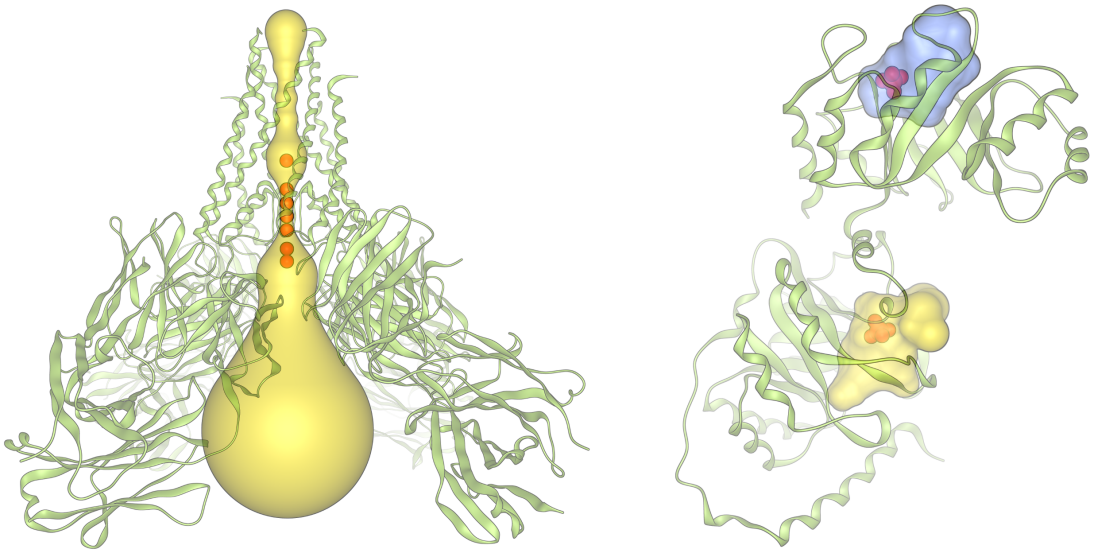


Figure 2.18: *Molecular cavities based on the geometric simplification showing the structure of the potassium channel pdb:1K4C (left) and two molecular pockets of the enzyme pdb:1PII (right).*

2.4.2 Simplification

Since the computation of all molecular paths is similar to an infinite number of physical simulations for all possible states of the small molecule, it is not practicable to directly apply this definition to the analysis of results of molecular simulations. In order to create a practical solution, the state of a molecule is often restrict to a space with pure geometrical properties. For the states of the large molecule, usually the hard sphere model is applied, that is only the atom positions and radii are used to create an imaginary hard boundary of the molecule. In addition, like for the SES, the small molecule is often approximated by a single probe sphere. Thus, for a state $\hat{X}$ of the large molecule, the set $S_{\hat{X}}$ includes all probe centers, where the probe does not intersect any atom sphere of the large molecule. With this restriction, a molecular path is a 3-dimensional continuous curve of probe centers; and a cavity is the union of all points inside all probe spheres that are connected by continuous curves in $S_{\hat{X}}$. The surface of cavities for this simplification is shown in Figure 2.18. Furthermore, a dynamic molecular path is a 3-dimensional continuous curve of the probe over time; and a dynamic cavity is the union of all cavities that are connected by dynamic molecular paths. For most grid-based algorithms, additionally, the shape of the molecules as well as their dynamics are discretized in $\mathbb{R}^3$. As boundary $b_{\hat{X}}$ for the large molecule, the convex hull of the atom positions or atom spheres is often used. Other approaches use a distance threshold to the atom spheres or simply the axis-aligned bounding box of the atom spheres. An ambient occlusion threshold as boundary indicator seems to be quite promising and is also used in several approaches.

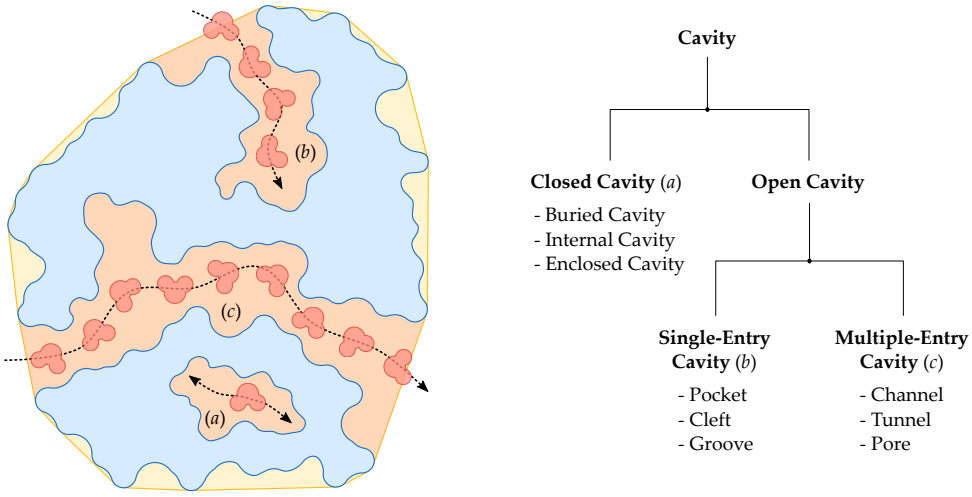


Figure 2.19: *Classification of cavities in a molecule (blue). Additionally, a molecular path through a tunnel, a path into a pocket, and a path inside a closed cavity are shown. The yellow region indicates the selected boundary which separates, for example, the tunnel from the pocket.*

2.4.3 Classification

In addition to the term cavity, many other denotations and classifications are used in the literature, like voids, pockets, tunnels, and channels. While void can be seen as a synonym for cavity, all other terms usually describe a specification of a class of cavities. To do so, one can distinguish between closed and open cavities. All molecular paths inside closed cavities do not reach the boundary, given by $b_{\hat{X}}$. Open cavities are further separated into cavities with a single entry or with multiple entries. An entry is a set of states of the small molecule where all states are connected by paths that lie completely in the boundary. Cavities with a single entry are often defined as pockets, grooves, or clefts and cavities with multiple entries as channels, tunnels, or pores. Figure 2.17 and 2.18 show an example of a channel and of two pockets and Figure 2.19 illustrates the classification, used in this thesis. However, since the methods proposed in this thesis are not restricted to a certain class of regions, usually only the term cavity is used to summarize all types of regions.

2.5 Molecular Grid Data Structures

Most algorithms for molecular visualization and analysis as well as path and cavity computation require neighborhood detections. This can be quite simple operations where all atoms close to a given point or another given atom need to be detected. But also more complex investigations are required, such as the detection of the atom sphere, which first intersects a given ray. For many algorithms, these neighborhood detections are crucial for the time complexity.

A naive implementation would compute and check the distance of each atom for a single neighborhood detection of a point. To reduce this amount of time, 3-dimensional data structures can be used such as grids, k-d trees, or octrees. For molecular data, often uniform rectangular grids are preferred. These grids are quite suitable to store objects, the distribution of which is almost uniform. Fortunately, the bonding forces but also the non-bonding forces between atoms lead to fix bounds for minimal atom distances in most real scenarios. This creates an almost uniform distribution of the atoms within the boundary of the molecular data.

In general, a 3-dimensional grid data structure decomposes a subspace of $\mathbb{R}^3$ into smaller regions, called *cells*. Usually, the cells do not overlap and their union is equal to the whole subspace. For each cell, the grid holds a list of objects that correspond to the cell from a spatial point of view. If all objects close to a certain point are requested, first all cells close to the point are detected and then only the objects stored in these cells need to be considered. This transfers the neighborhood detection problem to two other problems. First, the construction of the grid, which means the decomposition of the space and the assignment of the objects to the cells. And second, the detection of close cells. Both problems can be quite complex for arbitrary grids. Fortunately, specific types of grids such as regular grids can solve both problems. The most often used type of regular grids are uniform rectangular grids. A uniform rectangular grid G can be formally defined as 5-tupel (I, d, C, O, ψ), where $I = [b_{min}, b_{max}) \subset \mathbb{R}^3$ is an interval that encloses the domain of interest. This interval is uniformly partitioned into d_i pieces in each dimension $i = 1, ..., 3$ with $d \in \mathbb{N}^3$. Thus, n equal-sized box-shaped cells are generated with $n = \prod d_i$. Each cell corresponds to a list of unique object identifiers $C_g \in C$ with $C_g \subset \mathbb{N}$ and $g \in \mathbb{N}^3$, $g_i \leq d_i$ for $i = 1, ..., 3$. All objects are stored in the set O. Furthermore, $\psi : \mathbb{N} \to O$ is a bijection which maps the object identifiers to the objects. With each uniform rectangular grid automatically a mapping $m_G : I \to C$ is induced which maps a 3-dimensional point within the interval to its corresponding cell, with

$$m_G : p \mapsto C_{\left\lfloor \frac{p - b_{min}}{b_{max} - b_{min}} \cdot d \right\rfloor}$$

Since this mapping has a constant complexity, the detection of all cells close to a point becomes also constant. Additionally, the assignment of the objects to the cells is usually simple and mainly depends on the complexity of the objects. Typically an object is assigned to a cell if the intersection of the cell and the object is non-empty. The intersection test of an object with an axis aligned box is quite simple for points and spheres [6, 130]. However, for arbitrary objects it can be complex. In such cases often heuristics are used that may assign an object to more cells than necessary but still lead to much faster neighborhood detections. In this thesis, grids are mainly used to store atom spheres or spheres that represent cavities. Since the variation of the atom radii is not very large, it is in several cases reasonable to store only

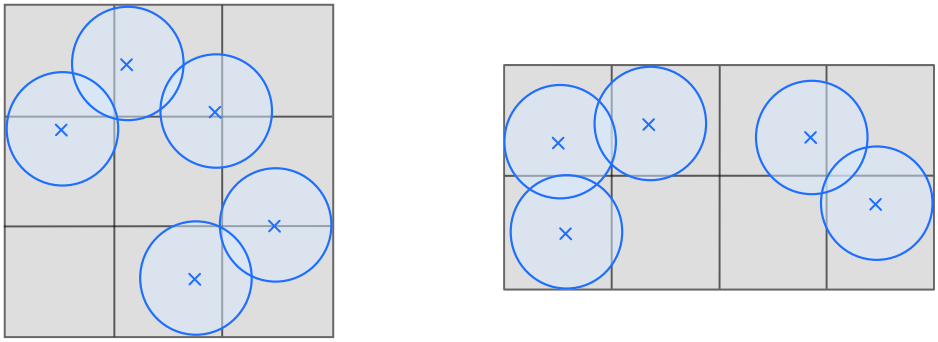


Figure 2.20: *A 2-dimensional illustration of a flexible grid for dynamic molecular trajectories. The maximal number of cells is twice the number of atoms, thus $M = 10$. The dimensions of the grid are computed in a way that the cells approach squares. Therefore, the left atom setting requires $m = 9$ and in case of the right setting $m = 8$ is used.*

the center points of the spheres into the grid. The assignment of a point to a cell is a single mapping operation using m_G and each point is assigned to exactly one cell. Thus the construction of the grid is very fast and the memory requirements are lower than for spheres. On the other hand, it is necessary to extend the search radius with the maximal sphere radius for neighborhood detections. As mentioned before, in practice atoms cannot be arbitrarily dense. Hence, the number of atoms within a fixed search radius is bounded by a constant. Using a grid data structure the complexity of the detection of these atoms becomes also constant. In the following, some details about implementations of uniform rectangular grids as used in this thesis are given.

2.5.1 Implementations

At the beginning of the construction of a grid, one has to define the region of interest as interval I and the number of partitions d. Both parameters together define the size of the cells, which have a major impact on the performance of neighborhood detection operations. While too large cells can contain many objects that lead to many potential neighbors, too small cells can create a lot of overhead, because many empty or redundant cells need to be processed. Often the minimal interval for I is given by the axis aligned bounding box of the input spheres or their centers. For most applications, I and d are selected based on a fixed cubical cell size, which is often in the range of a few Angstrom. For dynamic data it can be better to create a flexible grid with at most $M \in \mathbb{N}$ cells, where M is fix over time and should be chosen proportionally to the number of input spheres. Based on interval I, the partitions d are

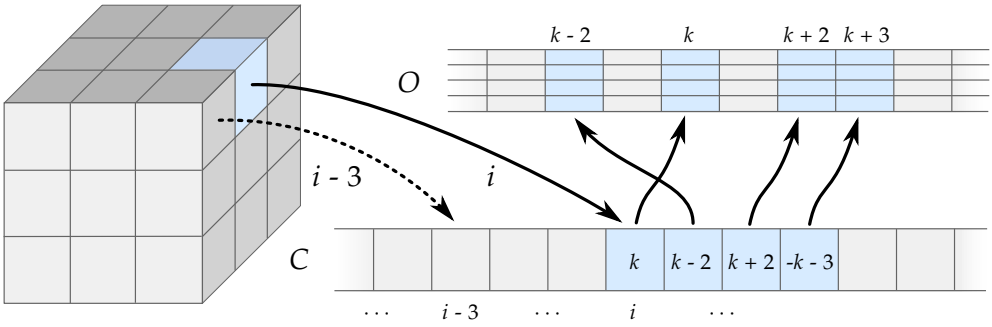


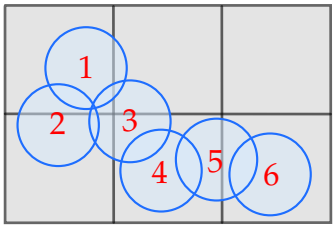
Figure 2.21: *Illustration of a grid data structure for OpenGL. Left: the 3-dimensional grid texture, which stores for each cell the position of the beginning of the list of object identifiers in the texture buffer object C. The object identifiers represent position in the texture buffer object O, which stores the spheres.*

computed for each time step such that the overall number of cells is smaller or equal than M and the cells become almost cubical, hence

$$d_i = \left\lfloor v_i \cdot \sqrt[3]{\frac{M}{v_1 \cdot v_2 \cdot v_3}} \right\rfloor \quad \text{with} \quad v_i = \frac{b_{max_i} - b_{min_i}}{\sum_{j=1}^{3} b_{max_j} - b_{min_j}} \quad \text{for} \quad i = 1, ..., 3.$$

Since the atom density within a molecule is not arbitrarily high, the maximal number of references in each cell is restricted by a constant. The fixed maximal number of grid cells M and the fixed number of references per cell allow the algorithm to quickly adjust the grid to the dynamic changes of the molecule without re-allocating memory (Figure 2.20). This can be of particular interest for algorithms running on the GPU.

On the CPU, grids are often implemented in an object oriented manner, where the grid is a container class of cells and each cell itself is a container of object identifiers, usually represented by integer numbers. Both classes provide functionality to easily fill, access and change the containers. On the GPU, the grid structure needs to be organized either in texture data for OpenGL or in simple arrays for OpenCL or CUDA. For OpenGL, it is quite suitable to represent the grid as 3-dimensional integer texture. Thus, the mapping m_g is almost automatically given by the *texelfetch* functionality in shaders. Each value corresponds to a cell and describes a position in a 1-dimensional integer array which is realized by a texture buffer object. The buffer implements C and ψ, which means it stores for all cells the lists of object identifiers. The integer value of the 3-dimensional grid texture points to the first object identifier of the list. All further identifiers are stored in the consecutive array elements. The end of the list is marked by a negative identifier. The object identifiers directly represent positions in a 1-dimensional floating point texture buffer object with four components per element, that implements O. The four components represent the position and radius of a single sphere. An illustration of this storage is shown in Figure 2.21. The implementation for OpenCL is quite similar. Both texture buffer objects are realized by classi-



(a)

0	3	-1
5	7	10

(b)

0	-1	-1
1	2	5

(c)

Texture buffer objects

0	1	2	3	4	5	6	7	8	9	10	11
p_1	p_2	p_3	p_1	p_3	p_2	p_3	p_3	p_4	p_5	p_5	p_6
r_1	r_2	$-r_3$	r_1	$-r_3$	r_2	$-r_3$	r_3	r_4	$-r_5$	r_5	$-r_6$

0	1	2	3	4	5
p_1	p_2	p_3	p_4	p_5	p_6
$-r_1$	$-r_2$	r_3	r_4	$-r_5$	$-r_6$

Figure 2.22: *The top row shows the grid and the atoms (a) together with the entries of the grid texture for two strategies (b) and (c). In (b) an atom is stored in a grid cell if it intersects the cell. The corresponding atom data texture buffer object is shown below (left). The grid in (c) stores an atom inside a cell if the center lies in the cell. The texture buffer object for this strategy is shown below (right).*

cal 1-dimensional arrays. And instead of a 3-dimensional texture, the grid is stored in a linearized 1-dimensional integer array. This requires the typical implementation of the bijection $c : \mathbb{N}^3 \to \mathbb{N}$ from the cell coordinates to the 1-dimensional array and back to the cell coordinates

$$c(p) = p_3 \cdot d_2 \cdot d_1 + p_2 \cdot d_1 + p_1 \qquad c^{-1}(i) = \begin{bmatrix} (i \,\%\, (d_1 \cdot d_2)) \,\%\, d_1 \\ (i \,\%\, (d_1 \cdot d_2)) \;/\; d_1 \\ i \;/\; (d_1 \cdot d_2) \end{bmatrix}$$

Depending on the requirements, for several implementations, especially for these running on the GPU, it can be reasonable to avoid ψ and O and directly store the objects into C. This reduces the number of array accesses, which can increase the performance a lot on the GPU. On the flip side, usually the memory requirements are higher because objects that belong to multiple cells are duplicated in C. However, if only the sphere centers are stored it is particularly recommended to avoid ψ since each sphere belongs to exactly one cell. Thus the memory requirements are even lower. A 2-dimensional illustration of the storage of a grid without ψ for the full spheres and only for the sphere centers is shown in Figure 2.22 for OpenGL. Note, that the end of the list of spheres for a cell is here indicated by a negative sphere radius.

3 Smooth Molecular Surfaces

Visualizing the dynamic behavior of molecules is particularly interesting to gain insight into a molecular system. For more than two decades, several types of molecular surfaces have been used to study interactions between proteins and ligands (Section 2.3.3). The most widely used type of molecular surface is the solvent excluded surface (SES). The molecular skin surface (MSS) is not yet used that often, but has a lot of potential and might become more important in the future. The interactive visualization of both dynamic molecular surfaces requires a balanced combination of an efficient computation of the surface description and its rendering. Additionally, the data transfer between CPU and GPU needs to be considered. Until recently, surfaces of both types had to be triangulated to visualize them. With modern GPUs, fast visualization has become possible using ray casting. In 2008 Chavent et al. [36] presented the first GPU-based ray casting for the interactive rendering of the MSS. But the time required to construct the MSS prevented its use for dynamic trajectories of proteins and other macromolecules. One year later, Krone et al. [120] reported an interactive SES visualization of molecular dynamics trajectories for a few thousand atoms.

In this chapter, it will be shown how to further accelerate the construction and rendering of the SES and MSS. These accelerations were described in "*Accelerated Visualization of Dynamic Molecular Surfaces*" [152] in 2010. The contributions can be summarized as follows. First, the approximate Voronoi diagram algorithm by Varshney et al. [232] is adopted for the computation of the MSS. The original algorithm was designed to compute the analytical description of the SES in linear time. By changing the distance function of the Voronoi diagram and a corresponding definition of neighboring atoms, the computation of the MSS will be accelerated by more than one order of mag-

nitude compared to Chavent et al. [36]. Second, it is demonstrated that the contour-buildup algorithm by Totrov [229] is ideally suited for computing the SES due to its inherently parallel structure. For both parallel algorithms, good scalability can be observed up to 8 cores. Thus, interactive frame rates will be obtained for molecular dynamics trajectories of up to twenty thousand atoms for the SES and up to a few thousand atoms for the MSS. Third, the rendering time for the SES is reduced by using tight-fitting bounding quadrangles as rasterization primitives. These primitives also accelerate the rendering of the MSS. With these improvements, the interactive visualization of the MSS of dynamic trajectories of a few thousand atoms became possible for the first time. Nevertheless, the SES remains a few times faster than the MSS such that the SES can be applied for several thousand atoms which comprises the majority of simulated proteins. For the following descriptions, consider a molecule that consists of n atoms with positions $p_i \in \mathbb{R}^3$ and radii $r_i \in \mathbb{R}$, with $i \in I = \{1, ..., n\}$.

3.1 Surface Computation

The algorithms for computing molecular surfaces can be divided into two categories. The first category comprises all methods that approximate the surface by discretizing the space of $\mathbb{R}^3$ [185, 35, 228, 248]. The second category contains all methods that compute a correct analytical representation of the surface. While algorithms of the first category are usually easier to implement, they can only achieve a fixed amount of detail. Furthermore, their running time and memory requirements scale typically cubically with the resolution of the underlying grid for the discretization. Hence, this section deals with algorithms of the second category.

In 1983, Connolly [45] presented the first algorithm for the computation of an analytical description of the SES, which was improved by Perrot et al. [186]. Varshney et al. [232] proposed an even faster parallelized version based on the computation of an approximate Voronoi diagram. Two years later a very efficient and elegant algorithm, called reduced surface algorithm, was presented by Sanner et al. [209]. Shortly after, it was extended to partial updates for dynamic data [208]. At the same time, Totrov and Abagyan proposed the contour-buildup algorithm [229]. For the MSS, the algorithm for the surface computation was implicitly given by its definition by Edelsbrunner [54].

In this section, two algorithms for the computation of molecular surfaces are described. The first is the contour-buildup algorithm [229] for the SES and the second is the approximate Voronoi diagram algorithm for weighted points of the MSS. The latter algorithm was previously developed for the computation of the SES [232]. By modifying the neighborhood definition of atoms with respect to the definition of the skin surface, it can be used for the computation of the MSS.

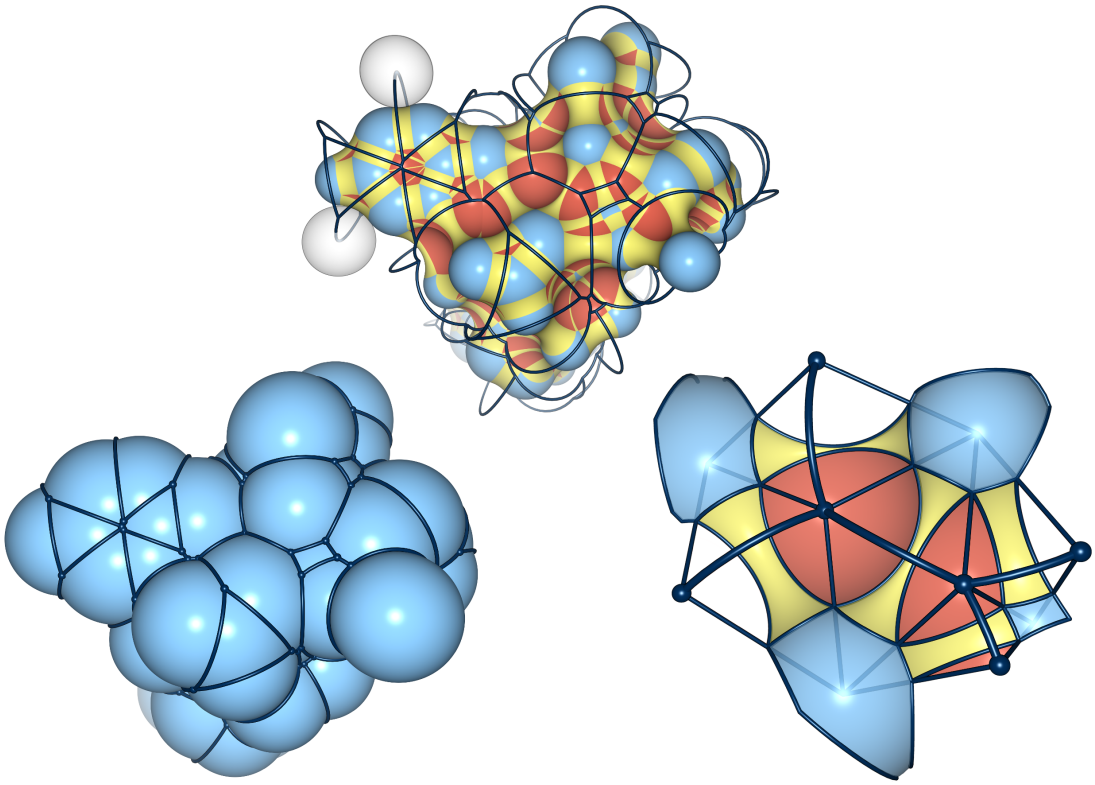


Figure 3.1: *Complete contour of the SAS (left) of deamino-oxytocin (pdb:1XY2). Each contour arc represents a toroidal patch (yellow), and each vertex, creates a concave spherical patch (red), which is shown in the middle image. The gray spheres show two possible positions of the probe. The contour of the SAS will be projected onto the van der Waals spheres to get the patch boundaries (right).*

3.1.1 SES: Contour-Buildup Algorithm

Recall that the SES is defined as the track of a probe sphere with radius r_p which rolls over the van der Waals spheres of the atoms of a molecule (Section 2.3.3). The surface can be decomposed into 3 types of patches: convex spherical, toroidal, and concave spherical. The first is defined by all points where the probe touches exactly one van der Waals sphere, the second is defined as the track where the probe touches exactly two spheres, and the third where the probe touches three and more spheres, respectively. Remember that the SAS shows the center of the probe when it rolls over the van der Waals atoms. The contour elements of the SAS directly correspond to the surface patches of the SES. Each arc corresponds to exactly one toroidal patch, and each vertex where three or more arcs meet corresponds to a concave spherical patch. The convex spherical patches are given by all minimal cycles of arcs (Figure 3.1). Note that all algorithms that compute the analytical description of the SES can only handle non-degenerate atom settings. This means there must not exist a concave spherical patch where the probe touches more than three atoms. This condition can be achieved by a small random per-

turbation of the atom positions. Alternatively, one could use simulation of simplicity [58].

The approximate Voronoi diagram algorithm [232] computes the contour of the SAS by detecting the Voronoi faces through neighboring atoms. Note that each contour arc of the SAS lies in such a face. Afterwards the contours can be quickly computed by the intersections of the Voronoi faces and the intersections with the extended van der Waals spheres. In contrast, the reduced surface algorithm computes some kind of dual structure of the SAS contour [209]. It consists of vertices, edges, and triangles. Each vertex corresponds to exactly one convex spherical patch, each edge to a toroidal patch and each triangle to a concave spherical patch. The third algorithm was presented by Totrov and Abagyan [229]. In contrast to the approximate Voronoi diagram algorithm, it directly computes the contour of the SAS. Furthermore, it is trivial to parallelize, which is not the case for the reduced surface algorithm. Based on these superior properties, the contour-buildup algorithm is most suitable to compute the SES. The algorithm consists of three parts:

1. Computation of the contour of the SAS.
2. Determination of the implicit surface description of each surface patch.
3. Detection of all singularities.

Contour Computation of the SAS

This part of the algorithm is the most involved one. It is illustrated in Figure 3.2. For each of the extended van der Waals spheres $(p_i, r_i + r_p)$, the contour describing the part of the boundary that contributes to the SAS has to be computed. These contours consist of circular arcs and full circles (Figure 3.1). The arcs and circles describing the contour of the ith atom are created by intersecting all neighboring atom spheres, where the neighborhood $N_{r_p}(i)$ is defined as

$$N_{r_p}(i) = \{j \mid ||p_j - p_i|| < r_j + r_i + 2r_p, \forall j, j \in I, j \neq i\}.$$

To quickly determine $N_{r_p}(i)$, a flexible 3-dimensional grid as described in Section 2.5 can be used. In the first step, for each extended van der Waals sphere $(p_i, r_i + r_p)$, all intersection circles c_{ij} are computed with $j \in N_{r_p}(i)$. These circles are sorted according to their distance to p_i. Then the circles are pairwise analyzed starting with the closest pair. During the analysis of a pair c_{ij} and c_{ik}, three important settings are detected. Consider the half-space H_{ij} given by the plane through c_{ij} and all points on the side of p_j. If c_{ik} lies completely in H_{ij}, then c_{ik} does not contribute to the contour and it can be removed. On the other hand if c_{ij} lies completely in H_{ik}, then c_{ij} can be removed. Furthermore, if both lie completely inside the half-space of the other one, atom i does not contribute to the SAS contour and all circles can be removed.

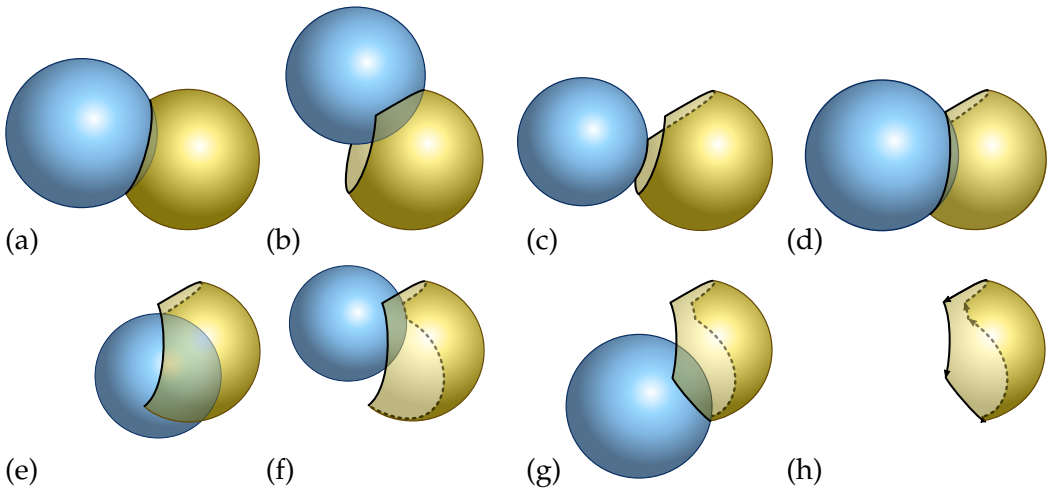


Figure 3.2: *Stepwise creation of the contour of the yellow atom. (a) Start contour (circle) created by the first neighboring atom. (b) The second circle splits the first circle, thereby creating two arcs. (c) Both arcs are contracted by the third circle. (d) The fourth circle deletes two arcs and contracts the remaining arc at both ends. (e, f, g) Analogously to (c). (h) Complete contour of the yellow atom.*

In the second step, the contour of each atom is constructed by an iterative approach. In each step, one of the remaining circles is taken and the current contour is updated. A circle can create a new contour circle or one or more arcs. In the latter case, existing contour elements might disappear or will be possibly cut. During a cut, the start or end point of the corresponding arc can change or it can be completely split. For more details, the reader is referred to the original publication by Totrov and Abagyan [229].

In protein-ligand docking simulations [31], the number of flexible atoms is sometimes reduced to the amino acids in the active site of the molecule to reduce the computational cost. In this case, only the contours of the flexible atoms in the active site together with their neighborhoods need to be updated.

Implicit Surface Patch Descriptions

As mentioned before, the analytical description of the SAS contains all information to compute the SES. The SAS contours simply need to be projected onto the respective van der Waals spheres. These projections generate the SES contours (Figure 3.1).

From each convex spherical patch of the SAS, a convex spherical patch of the SES is generated. Such a patch can be geometrically described as the surface of the corresponding van der Waals sphere that is bounded by planes through the projected contour arcs (Figure 3.1).

Each contour arc of the SAS generates a toroidal patch. The small radius of the torus is the probe radius r_p and the big radius is the distance of the contour to the line through the positions of the atoms which created the contour arc. This line also represents the axis of the torus. A toroidal patch is

bounded by at least two planes perpendicular to the torus axis where the patch connects to convex spherical patches. These are the same planes that bound the convex spherical patches. Additionally, in case of a real arc and not a complete contour circle, the torus is bounded by two more planes in the direction of the two neighboring concave spherical patches. These planes include the start and end point, respectively, as well as the two positions of the atoms that created the contour arc (Figure 3.1).

In the non-degenerate case, each concave spherical patch is a spherical triangle given by a vertex in the SAS contour. The corresponding sphere of the triangle is the probe with the center at the SAS contour vertex. The triangle vertices are the intersection points of the probe surface with lines through the probe center and the positions of the atoms which created the contour vertex. The arcs of the spherical triangles lie in the same planes like the boundaries of the neighboring toroidal patches (Figure 3.1).

Singularities

The generated SES description might contain self-intersections, which arise in two cases. First, if the small circle of the torus is larger than its big circle, the torus intersects itself. When this self-intersection is removed, two cusps remain. This intersection can be easily identified and removed. On the other hand, the intersection of two concave spherical patches is more costly to identify. Here, all concave spherical patches that are close enough to each other need to be tested for intersections. Similarly to the neighborhood search for the contour computation, these patches can be found using a 3-dimensional grid. In the original algorithm, the singularity contours were computed again with the contour-buildup method. However, since the surface will be ray cast and not triangulated, for each concave spherical patch, only the clipping planes describing these self-intersections are required and stored for each patch.

3.1.2 MSS: Approximate Voronoi Diagram

In this section, the idea of the approximate Voronoi diagram algorithm to compute the SES [232] is transferred to the computation of the MSS. The MSS was described in Section 2.1.4 and 2.3.3. Briefly, it is a C^1 continuous surface that can be decomposed into patches of quadrics. Its shape depends only on a single parameter called shrink factor $s \in (0,1) \subset \mathbb{R}$. The surface patches and their boundaries are implicitly given by the Voronoi diagram of the weighted points (p_i, w_i) with $w_i = {r_i}^2/s$ and the distance function $d_i(p) = \|p - p_i\|^2 - w_i$.

In case of the SES, Varshney et al. [232] found that it is not necessary to compute the correct Voronoi diagram of the SAS. They observed that only specific elements in the Voronoi diagram correspond to surface patches of the SES and that the other elements can be ignored. For this reason they

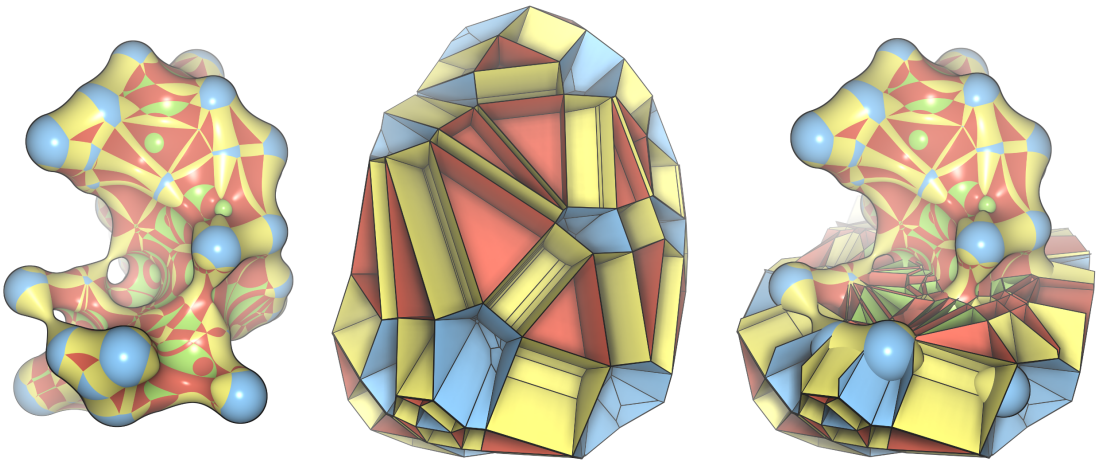


Figure 3.3: *MSS of deamino-oxytocin (pdb: 1XY2) colored by the surface patch types (left). The convex spherical patches are depicted in blue, the hyperbolic patches of type I in yellow, the hyperbolic patches of type II in red, and the concave spherical patches in green. The corresponding mixed cells for the patches are illustrated in the middle and a combination with the MSS is shown right.*

defined feasible Voronoi regions that only consider the atoms in the local neighborhood. A feasible Voronoi region V_i is a correct Voronoi region of a subset of the input atoms given by the neighborhood $N_{r_p}(i)$. Due to this restriction, the computation of the SES scales linearly with the number of atoms.

A similar observation can be made for the MSS. Consider the correct Voronoi diagram and its corresponding mixed complex depending on s. Based on the Voronoi and Delaunay elements, one can distinguish between four different patch types and their bounding mixed cells (Figure 3.3):

1. Convex spherical patches. Each patch is given by the surface of the van der Waals sphere of the corresponding atom. The mixed cell is the scaled Voronoi region which is a convex polyhedron.

2. Hyperbolic patches of type I. These patches are given by the Voronoi faces. Each patch is bounded by a prism, where the base area is equal to the scaled Voronoi face.

3. Hyperbolic patches of type II. This type comprises all patches defined by Voronoi edges. Their corresponding mixed cells are also prisms but with triangular base area. The triangles are given by the scaled dual Delaunay faces.

4. Concave spherical patches. These patches are defined by the Voronoi vertices and their corresponding mixed cells are tetrahedra. The tetrahedra are equal to the scaled dual Delaunay regions.

The convex spherical patches are connected by hyperbolic patches of type I along the clipping planes of the polyhedra. The interesting observation is,

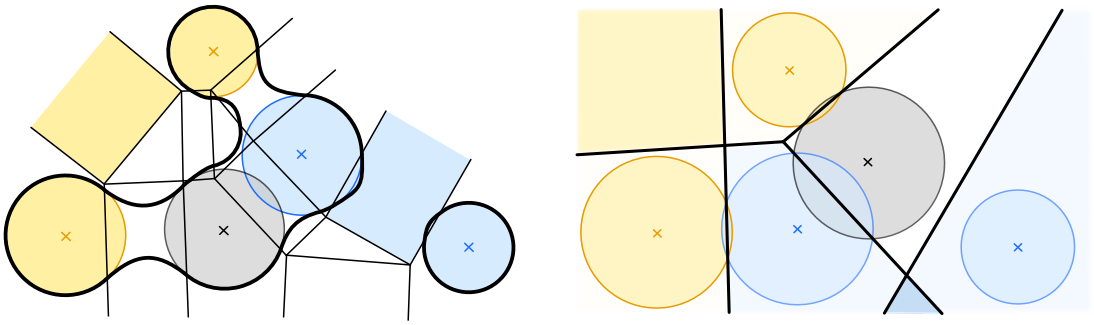


Figure 3.4: *Left: Illustration of the MSS with the corresponding mixed complex. The blue cell shows an example of a mixed cell that intersects the atom spheres and creates a patch between the two atoms. Hence, these atoms are neighbors. In contrast, the yellow cell does not intersect the atom spheres. Thus, the cell is empty and does not create a connecting patch. The corresponding atoms are not neighbors. Right: Illustration of the feasible regions based on the neighborhood definition (left). Two pairs of feasible regions intersect each other (yellow and blue).*

that if the plane in which a face of the polyhedron lies does not intersect the van der Waals sphere, there cannot exist a connecting hyperbolic patch for the corresponding Voronoi face. Furthermore, the intersection of the corresponding mixed complex cell with the implicit surface is empty. Hence it is not necessary to compute these faces, and the Voronoi region can be restricted to feasible regions based on the neighboring atoms (Figure 3.4).

To properly define a feasible region for a weighted point of the MSS, the definition of the neighborhood $N_s(i)$ of an atom i with shrink factor s is required. Let $V_{\{i,j\}}$ be the Voronoi face between atoms i and j. Furthermore, let $H_{i,j}$ be the corresponding plane, which is defined by all points p with $||p - p_i||^2 - w_i = ||p - q_j||^2 - w_j$. The orthogonal distance $d_i(H_{i,j})$ from p_i to $H_{i,j}$ is given by

$$d_i(H_{i,j}) = \left| \frac{(p_j - p_i)^2 + w_i - w_j}{2 \cdot \|p_j - p_i\|} \right| .$$

This distance is scaled by s to get the distance of the plane through the corresponding face of the bounding polyhedron of the convex spherical patch. If the distance is larger than r_i, then there cannot exist a surface patch between atom i and j. Hence the neighborhood $N_s(i)$ is defined as

$$N_s(i) := \{ j \mid s \cdot d_i(H_{i,j}) < r_i, 1 \leq j \leq n, j \neq i \} .$$

A feasible region for the weighted point i is then given by

$$F_i := \left\{ x \in \mathbb{R}^3 \mid d_i(x) \leq d_j(x) : \forall j \in N_s(i) \right\} .$$

F_i is generally larger than V_i because fewer neighbors are considered for the computation of F_i and thus fewer planes restrict the region. This can lead to overlapping feasible regions, (Figure 3.4). Due to computing feasible regions

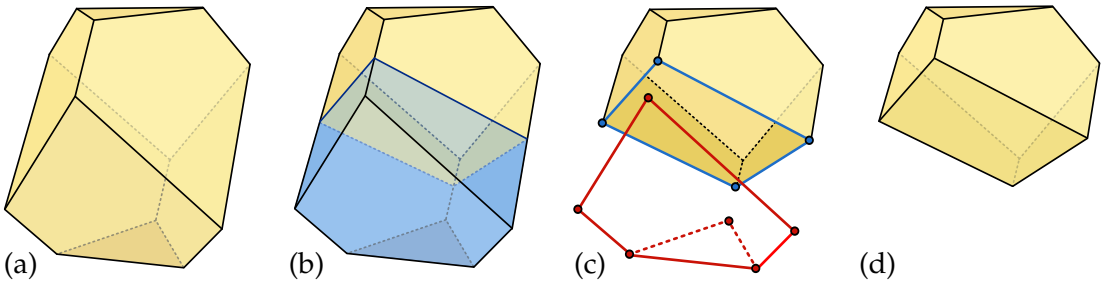


Figure 3.5: *One step of the creation of a feasible Voronoi region by cutting the region (a) with the separating plane of another atom (b). First, all elements of the cell are detected, that will be removed (red) and that need to be modified (blue)(c). Second, along the modified edges a new face is created to close the region (d).*

instead of the correct Voronoi regions, the algorithm scales linearly with the number of weighted points, because for physically correct molecules, the size of the neighborhood of the weighted points is bounded by a constant. Each feasible region can be computed independently. Hence, the algorithm can be easily parallelized as was described by Varshney et al. [232]. In contrast to their approach, here, the feasible regions are computed by cutting polyhedrons. Each region is initialized with a tetrahedron large enough to enclose the whole molecule. Then the cell is iteratively cut and closed according to the planes $H_{i,j}$ of all neighbored atoms $N_s(i)$ (Figure 3.5). Finally, the algorithm returns a closed feasible cell. All cell elements that are part of the initial tetrahedron are subsequently ignored.

To obtain the analytical description of the MSS, for each of the 0- to 3-dimensional feasible Voronoi elements, the implicit function is generated as described in Section 2.1.4. Additionally, the computation of the corresponding bounding feasible mixed cells is straightforward.

3.2 Rendering

Usually, surfaces are triangulated in order to visualize them in real-time computer graphics. However, this can be time consuming depending on the complexity of the surface and the quality of the resulting triangular mesh. Furthermore, with triangulations, one can only achieve a fixed amount of detail, based on memory constraints and the requested interactive rendering performance. For many years, numerous publications have dealt with the triangulation of molecular surfaces. In 1985, Connolly presented the first method for the SES [46]. Most approaches first triangulate the surface patches separately and then combine them into a single surface mesh [1, 209, 132, 37, 127]. Additionally, Bajaj et al. approximate the patches by spline surfaces to simplify the triangulation [10, 11, 12, 250]. One of the fastest methods was proposed by Ryu et al., which uses subdivision surfaces [206, 205]. However, their approach is not able to handle all possible singularities. They focus only on the ones that occur most often. A high quality triangulation of the MSS was

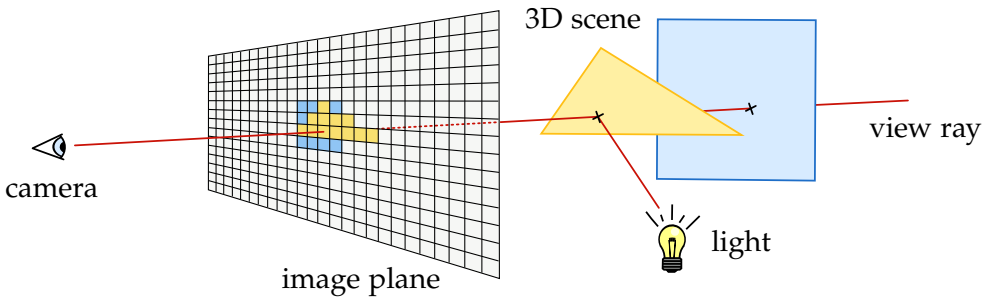


Figure 3.6: *The general concept of ray casting. Through each pixel in the image plane, a ray from the camera is traced. If the ray intersects an object in the 3-dimensional scene, the closest intersection point to the camera is detected. A second ray from the intersection point to the light source is used to shade the pixel with a simple illumination model.*

proposed by Cheng et al. [38, 39, 40]. Nevertheless, all currently available methods to triangulate the SES or MSS are either not fast enough to use them for dynamic protein data or their quality is insufficient.

An alternative to the triangulation-based rendering is *ray casting*. Classical ray casting algorithms compute for each pixel of the final image the view ray from the camera through this pixel. If this ray intersects one or more objects, the closest intersection point is detected and a simple illumination model is used to compute the color of the pixel (Figure 3.6). The advantage of this approach is, that the degree of detail is determined by the number of pixels in the final image and not by the 3-dimensional scene. However, this requires the computation of the intersection of the ray with all objects in the scene. Depending on the complexity of the objects, this can be very time-consuming and is sometimes not even analytically possible. But due to the continuous advances in computer graphics hardware and the possibility to program the graphic pipeline with shader languages, for several applications, ray casting has become more efficient than triangulation.

One application is the rendering of simple algebraic surfaces. Recall, that algebraic surfaces are implicit surfaces that can be described by polynomials (Section 2.1.2). Consider, for example, the surface of a sphere, which can be described as algebraic surface of degree two, called quadric. A high quality triangulation of a sphere requires several hundreds or thousands of triangles. On the other hand, the computation of the intersection of a ray with a sphere results in the detection of the root for a polynomial of degree two, which is quite simple. For this reason, the classical ray casting was transferred to the GPU to interactively render simple algebraic surfaces with high quality.

In contrast to the classical ray casting, where the whole scene is rendered at once, on the GPU each surface instance is rendered independently. To do so, one or more simple polygons or polyhedrons are generated for each surface patch such that their projection to the image plane encloses the projection of the corresponding surface patch. These objects will be called *rasterization primitives* here. After the projection, in the fragment shader, for each fragment of the rasterization primitive, the view ray from the camera is computed and

the intersection test with the corresponding surface patch is performed. In case of an intersection, the illumination model is applied to color the fragment. Furthermore, the correct depth of the intersection point instead of the rasterization primitive is computed and stored in the depth buffer. Because the GPU stores automatically only the closest fragment, all occluded intersections of the ray with other patches are ignored.

In 2003, one of the first GPU-based ray casting approaches was presented by Gumhold to render ellipsoids in tensor fields [82]. The data transfer of his technique was optimized by Klein and Ertl to illustrate magnetic field lines [114]. In 2006, Sigg et al. presented an efficient and general ray casting approach for arbitrary quadrics [216]. In contrast to the previous approaches that were developed only for orthogonal projection, they are able to use perspective projection. Additionally, they used point sprites to further reduce the data transfer to the GPU. They demonstrated the applicability of their technique on the ball-and-stick representation and the van der Waals surface. Loop and Blinn developed the first ray casting technique to render piecewise algebraic surfaces up to degree four [153]. They use the analytical Ferrari-Lagrange method presented by Herbison-Evans [90] to compute the intersections of the ray with the surface patches. Toledo and Lévy also investigated the ray casting of algebraic surfaces up to degree four with the focus on tori [227, 226]. They analyzed the performance depending on different rasterization primitives and different iterative approaches for the computation of the intersection points. Based on their results, they recommend the iterative Newton-Raphson algorithm, which they found to be superior to Hart's sphere tracing [86] and the analytical solution used by Loop and Blinn [153]. Ray casting of algebraic surfaces of even higher degree than four was presented by Singh and Narayanan [217]. However, the focus of their approach lies on the rendering of a single surface patch.

Both, MSS and SES are composed of piecewise algebraic surface patches. While the MSS consists only of patches of degree two, the patches of the SES are of degree two and four. The latter patches are also known as quartics. Since the intersections of a ray and these algebraic surfaces can be computed very efficiently, the MSS and SES are ideally suited for ray casting. Chavent et al. presented the first ray casting algorithm to visualize the MSS directly on the GPU [36]. A similar but more efficient approach was presented by Krone et al. for the SES [120].

The performance of GPU-based ray casting is mainly determined by two operations. First, the complexity of the computation of the intersections of rays with the algebraic surface. Thus, the faster the intersections can be computed, the faster the surface can be rendered. Second, the number of rays that need to be tested for intersections is determined by the size of the selected rasterization primitives, because for each rasterized fragment, an intersection test will be done. The smaller these primitives are, the less rays need to be tested. However, a small number of rays will only be worthwhile if not too much time is spent for computing the primitives. In the following, the ray

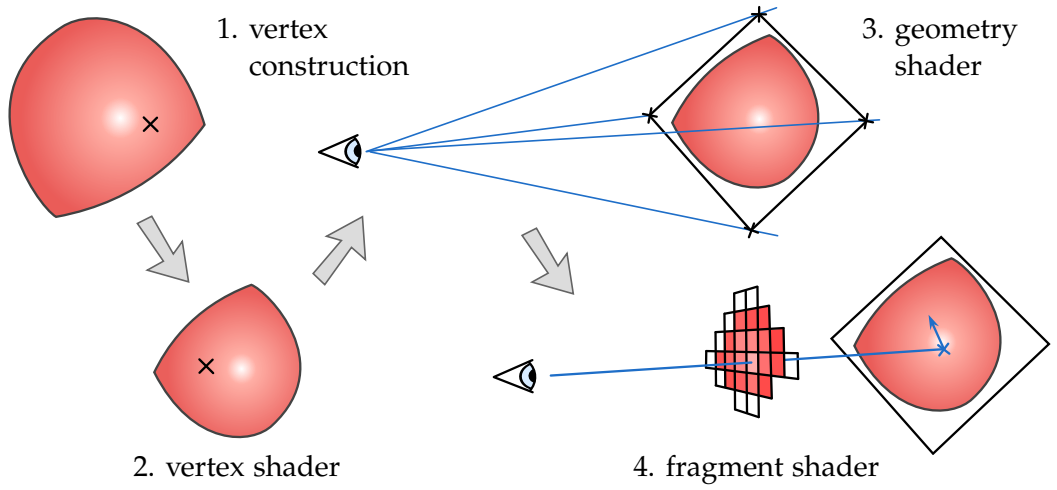


Figure 3.7: *Concept of ray casting on the GPU for a surface patch of the SES. First for each patch a vertex is created which contains the implicit surface equation and the patch boundary. Second, in the vertex shader, the patch is transformed into modelview space. Third, a tight fitting quadrangle enclosing the complete patch after projection is generated by the geometry shader. And fourth, the ray casting is performed for each fragment of the quadrangle.*

casting of the SES and MSS is presented, where these two operations are optimized to achieve interactive rendering.

3.2.1 Pipeline

Before the two important operations are described in detail, an outline of the optimized ray casting pipeline on the GPU is given. The pipeline is illustrated in Figure 3.7. In case of the SES, for each surface patch a single vertex is created which contains all information about the patch. For example, the vertex for a concave spherical patch stores the position and radius of the sphere and the three boundaries of the spherical triangle. The vertices are then transferred to the GPU, where each patch is transformed by the modelview matrix in the vertex shader. Afterwards, a rasterization primitive for each patch is created in the geometry shader. For the SES, a tight fitting quadrangle is created out of the single vertex. Additionally, for each vertex of the quadrangle, the view ray from the camera is computed. During the rasterization of the quadrangle, the view rays will be automatically interpolated such that each fragment gets the correct ray. Then, in the fragment shader, the actual ray casting is performed. First, the intersection with the surface patch is computed. In case of an intersection, the classical Blinn-Phong [17] illumination model is applied to color the fragment. Finally, the correct depth of the fragment is required to enable depth tests with neighboring patches. Using the geometry shader to create the rasterization primitives has two important advantages. First, the data upload to the GPU is minimized, similar to the work of Sigg et al. [216]. And second, it is possible to create individual tight fitting raster-

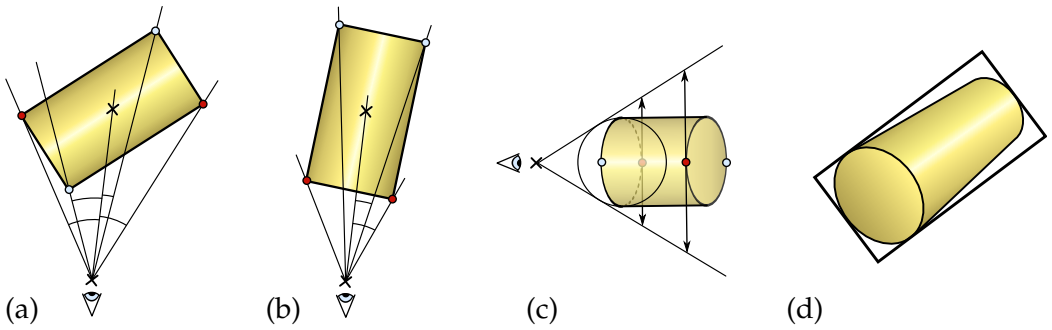


Figure 3.8: *Computation of a tight planar object that includes a cylinder after rasterization (d). In (a) and (b), two possibilities for the outer points of the cylinder are depicted. (c) displays how the outer points are used to span the object.*

ization primitives as it was done by Gumhold [82] or Toledo and Lévy [226], which is, for example, not possible with point sprites.

3.2.2 Rasterization Primitives

In the following, some details are described about the rasterization primitives that will be suitable for the patches of the SES and MSS.

SES

Recall that a convex spherical patch describes a part of the van der Waals sphere of an atom which is bounded by arcs. The boundary of each arc can be implemented by a clipping plane. However, the clipped parts of the van der Waals sphere lie completely inside the SES and never penetrate it. Hence, for an opaque visualization of the SES, it is faster and easier to render the whole van der Waals spheres. For each van der Waals sphere, a single vertex is created, which stores only the center of the sphere and the radius. In the geometry shader, a square is computed as rasterization primitive. This square is placed orthogonally to the line from the camera to the sphere center. The size of the square depends on the distance to the camera and the radius of the sphere. It is chosen in a way that it tightly surrounds the silhouette of the sphere, from the view of the camera.

A toroidal patch is a piece of an inner part of a torus which is bounded by at most four circular arcs. Two arcs connect the toroidal patch with two convex spherical patches. These two boundaries can be represented by clipping planes that are orthogonal to the torus axis. The other two arcs occur in case the patch is neighbored to concave spherical patches Figure 3.1. However, similar to the convex spherical patches along these directions, the toroidal patch lies always inside the SES. Thus, again it is faster and easier to ignore these boundaries. For this reason, the vertex of a toroidal patch stores the midpoint and axis of the torus as well as the small and big radii and two distances from the midpoint for the two clipping planes in direction of the convex spherical patches. For the computation of a rasterization primitive,

the patch is first bounded by a tight cylinder. While the direction of the axis of the cylinder is equal to the axis of the torus, it can be shifted to minimize its radius. Therefore, the vertex stores also the shifting and the radius of the cylinder.

Ignoring the self-intersections, a concave spherical patch is a spherical triangle with vertices v_1, v_2, and v_3. Similar to the toroidal patches, a bounding cylinder is constructed. The cylinder is chosen such that the radius is equal to the radius of the circumcircle c of v_1, v_2, and v_3. The cylinder axis starts in the midpoint of c and is orthogonal to the plane H_s spanned by v_1, v_2, and v_3. The cylinder length is equal to the probe radius r_p minus the orthogonal distance from the probe center p_p to H_s. As mentioned before, each vertex representing a concave spherical patch stores the position and radius of the sphere as well as the three vertex positions of the spherical triangle.

For the bounding cylinder of the toroidal patches and the concave spherical patches, a bounding quadrangle can be computed as follows (Figure 3.8). Let H_c be the plane spanned by the cylinder axis and the vector from the camera position to the cylinder center. Then, the four intersection points of H_c with the cylinder caps are determined. The right-most and left-most points are detected, which are depicted by the red points in Figure 3.8 (a) and (b). Through these two extremal points, a trapezoid is spanned the parallel sides of which are orthogonal to H_c. The lengths of these sides need to be chosen such that the trapezoid completely encloses the cylinder (Figure 3.8, c). In Figure 3.8 (d) the projection of the cylinder and the corresponding quadrangle is shown.

MSS

For the convex spherical patches of the MSS, the same bounding quadrangles are used as for the SES. For the hyperbolic patches corresponding to Voronoi faces, truncated pyramids are used. And for the other two types of patches, directly the mixed cells are used. These cells are prisms with triangular base area and tetrahedra, respectively. Note that here is room for further optimization, but optimizing the rasterization primitives for the two hyperbolic and the concave spherical patch types is more complex.

3.2.3 Ray Intersection

After computing the tight-fitting rasterization primitives, for each fragment of a primitive, the ray casting is performed in the fragment shader. In order to compute the intersection of the view ray with the surface patch, the parametric description of the ray is inserted into the implicit surface equation of the patch. The intersection points are the roots of the resulting polynomial. In case of quadric patches, the polynomials have degree 2. The analytical solution for solving roots of quadrics is well studied, and stable formulas are available [195]. These formulas will be directly used in the fragment shader

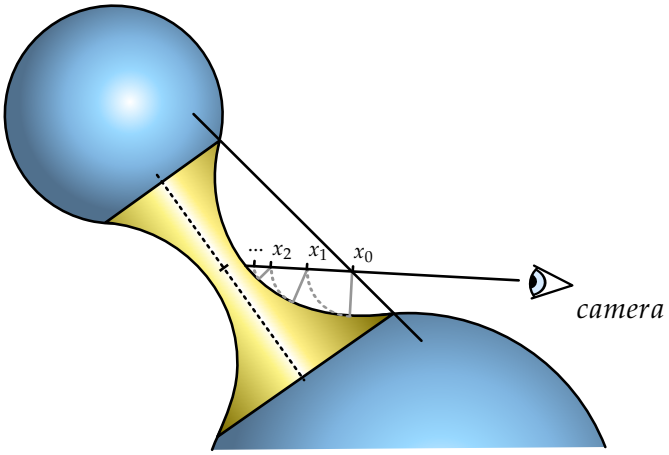


Figure 3.9: *Ray casting of a toroidal patch. First the intersection point x_0 with the bounding cone is computed. Then the sphere tracing is started from this point. The distance to the surface determines the step size.*

to compute the intersection points for all spherical patches as well as the hyperbolic patches of the MSS.

For intersecting a ray with the toroidal patch, the roots of a polynomial of degree four need to be detected. Although there are analytical solutions to this problem [90] that have been successfully used for these patches [120], here sphere tracing [86] is applied. In order to get a good start position for the algorithm, first the intersection $x_0 \in \mathbb{R}^3$ of the ray with the minimal bounding cone of the patch is computed (Figure 3.9). If the ray does not intersect the cone or if it intersects the conic object at the flat ends, the ray does not intersect the toroidal patch. Otherwise, the sphere tracing is started at the intersection point x_0. In each iteration step i of the algorithm, the minimal distance $l_i \in \mathbb{R}$ from the current position $x_i \in \mathbb{R}^3$ to the patch surface is computed. Because l_i is the minimal distance, it is guaranteed that the next point x_{i+1} along the ray with distance l_i to x_i still lies in front or exactly on the surface of the patch. From x_{i+1} again the minimal distance l_{i+1} is computed. The algorithm proceeds until the minimal distance reaches a fixed threshold or a maximal number of steps is reached. If the last point still lies too far away from the surface, it will be ignored. Otherwise, the last point is taken as intersection point. Based on visual experience, a good choice for the maximal number of iteration steps is 30, whereby the actual maximal number can be computed depending on the distance of the patch to the camera. Thus for close patches maximal 30 iterations should be used while for far patches often 10 or less iterations are enough to achieve a high visual quality.

Most of the patches of the SES and MSS are bounded by clipping planes. While in most cases the number of clipping planes is small and fixed, this is not the case for the singularities of the concave spherical patches of the SES. To remove the self-intersections, the corresponding clipping planes describing the self-intersection are stored in a texture as done by Krone et al. [120]. The

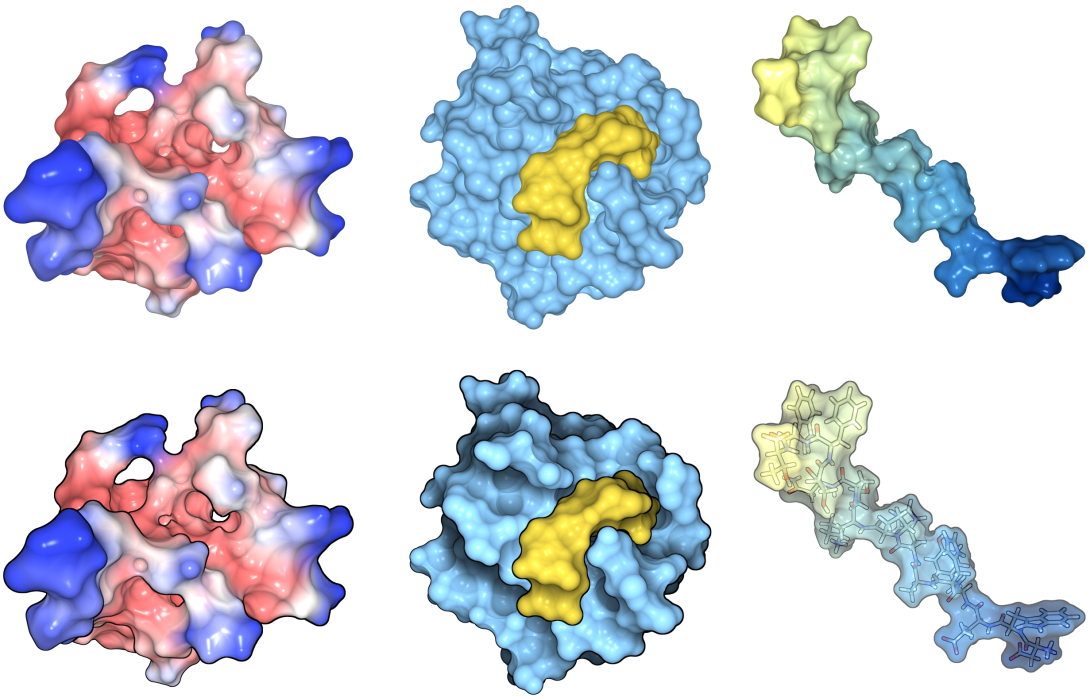


Figure 3.10: *Comparison between pure surface rendering (first row) and surface rendering with post-processing (second row) of pdb: 1X5V (left), pdb: 2RNT (middle), and pdb: 2JQC (right). For 1X5V and 2JQC, the MSS is shown and for 2RNT the SES is visualized. In all three examples, silhouettes are used and for 2RNT and 2JQC also depth darkening is applied. Additionally, the surface of 2JQC is blended with the ball-and-stick representation.*

fragment shader then tests each intersection point whether it is clipped by any clipping plane. In case the point is clipped, it will be discarded.

3.2.4 Post-Processing

Although the surfaces of the SES and MSS are smooth, it can still be difficult to focus on specific regions, especially for large molecules. The most critical point is the depth perception due to the simple direct illumination model. Many techniques have been proposed to interactively enhance the depth perception. Under these techniques especially the visualization of silhouettes and the interactive approximation of a global illumination model play important roles. However, even very simple techniques, like depth cueing, can be very effective. During this work, all three techniques were implemented for the visualization of the SES and MSS. Therefore, the surface is first rendered to a frame buffer object, which stores two textures for the shaded colors and the depth values. In a second pass, a screen space filling rectangle is rendered with both textures. During this rendering a shader implements one ore more of these techniques. For the depth cueing, simply the color is brightened or darkened with increasing distance. For silhouettes, the gradient of the depth is approximated using the local neighborhood. The result is then

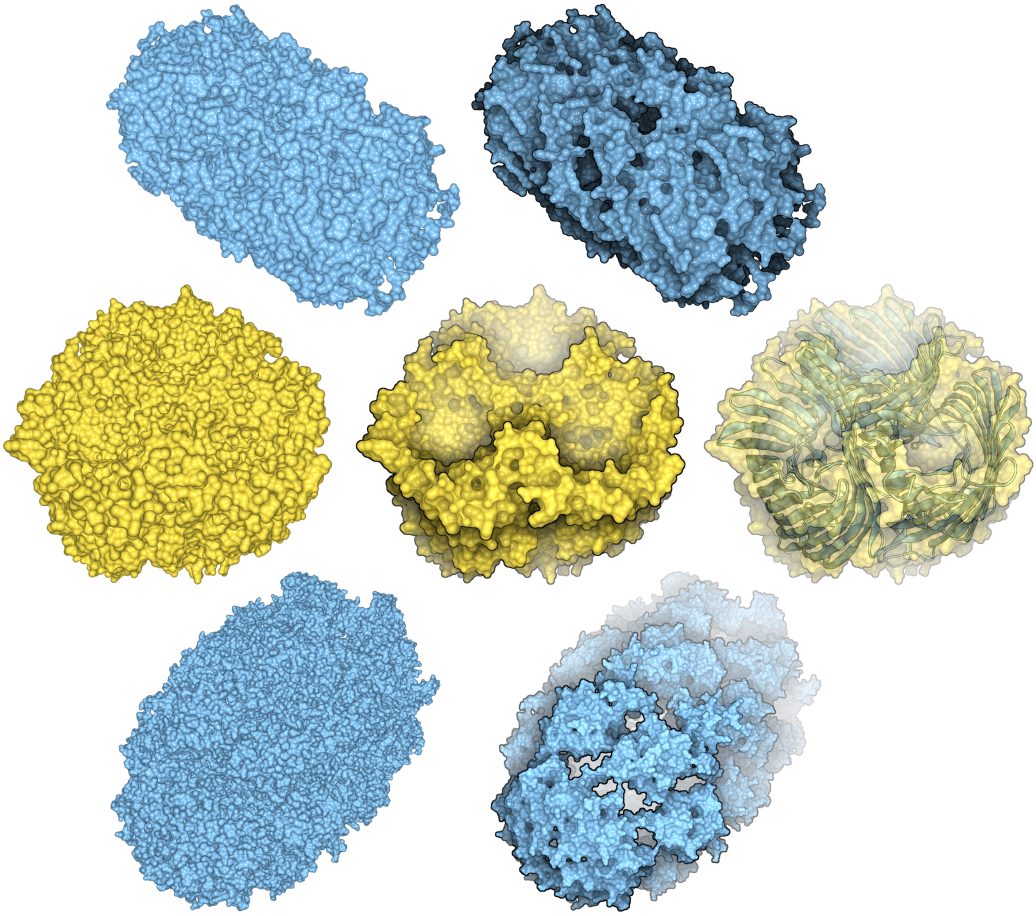


Figure 3.11: *Comparison between pure SES rendering (left) and surface rendering with post-processing of pdb: 1J4N (top), pdb: 1AF6 (middle), and pdb: 1AON (bottom). In all three examples, the post processing applies silhouette detection and depth darkening as well as depth cueing for 1AF6 and 1AON. Additionally, the surface of 1AF6 was blended with the secondary structure representation.*

combined with the color. In order to approximate global illumination Luft et al. presented a simple but efficient technique, called depth darkening [157]. For each pixel, the depth function is smoothed using a Gaussian kernel. Afterwards, the difference between the smoothed depth and the original depth is used to darken the color of the pixel. This creates an illusion of global illumination because at the boundaries of cavities the surface becomes darker. Several examples that show the effect of these techniques can be seen in Figure 3.10 and 3.11.

The rendering of the SES and MSS, as described above, cannot visualize the surface with transparency. Nevertheless, it is still possible to use a fast blending by rendering only the front face of the surface transparently in combination with other visualizations. This is for most applications sufficient and

PDB-ID	#Atoms	CB[1] (SES)		AVD[1] (MSS)		RS[2] (SES)
		1	8	1	8	1
1VIS	2 531	102	18	451	83	80
1AF6	10 517	451	73	2 113	369	360
1GKI	20 150	880	145	4 184	698	770
1AON	58 870	2 407	405	9 924	1 683	2 680
3G71	99 174	4 488	759	18 507	3 353	–

[1]System: 2 Intel Xeon E5540 2.53 GHz. [2]System: Intel Core 2 Duo 3 GHz.

Table 3.1: *Update times in milliseconds for the surface computations using OpenMP with* 1 *and* 8 *cores. For the SES, the contour-buildup algorithm (CB) was used and for the MSS, the approximate Voronoi diagram (AVD) algorithm. The last column shows the update times for the reduced surface (RS) algorithm, implemented by Krone et al. [120].*

sometimes even more useful than a correct transparency. Examples are shown in Figure 3.10 and 3.11.

3.3 Results

The implementations were tested on several molecules of different size. Static proteins were used from the protein data bank [184] as well as dynamic data sets from cooperation partners.

3.3.1 Surface Computation

In this section, the results for the parallelized versions of the contour-buildup algorithm and the approximate Voronoi diagram algorithm are presented. All tests were performed on an 8 core 2,53 GHz Intel system. Table 3.1 gives the timings for the molecular surface computations performed on 1 and 8 cores. The timings include the computation of all surface patches as well as their transfer to the GPU. With 8 cores, a speedup of approximately 6 was measured for the SES and 5 to 6 for the MSS. The MSS was computed with a shrink factor $s = 0.3$, because for this value, the surface seems to be most similar to the SES with a probe radius of 1.4 Å.

The plots (a) and (c) in Figure 3.12 show the speedup for the computation of the SES and MSS for the chaperonin molecule (*pdb:1AON*) with $\sim$ 60k atoms. The plots contain the speedup for the overall computation and for the individual parts. Note that for the main part of the computation (the blue curve), a speedup of more than 7 was measured on 8 cores. The plots (b) and (d) in Figure 3.12 show the proportionate time for each part of the algorithm.

Compared to the reduced surface algorithm [209] used by Krone et al. [120], the contour-buildup algorithm scales better with the number of atoms (Table 3.1). While for small molecules the running time is slightly higher for the

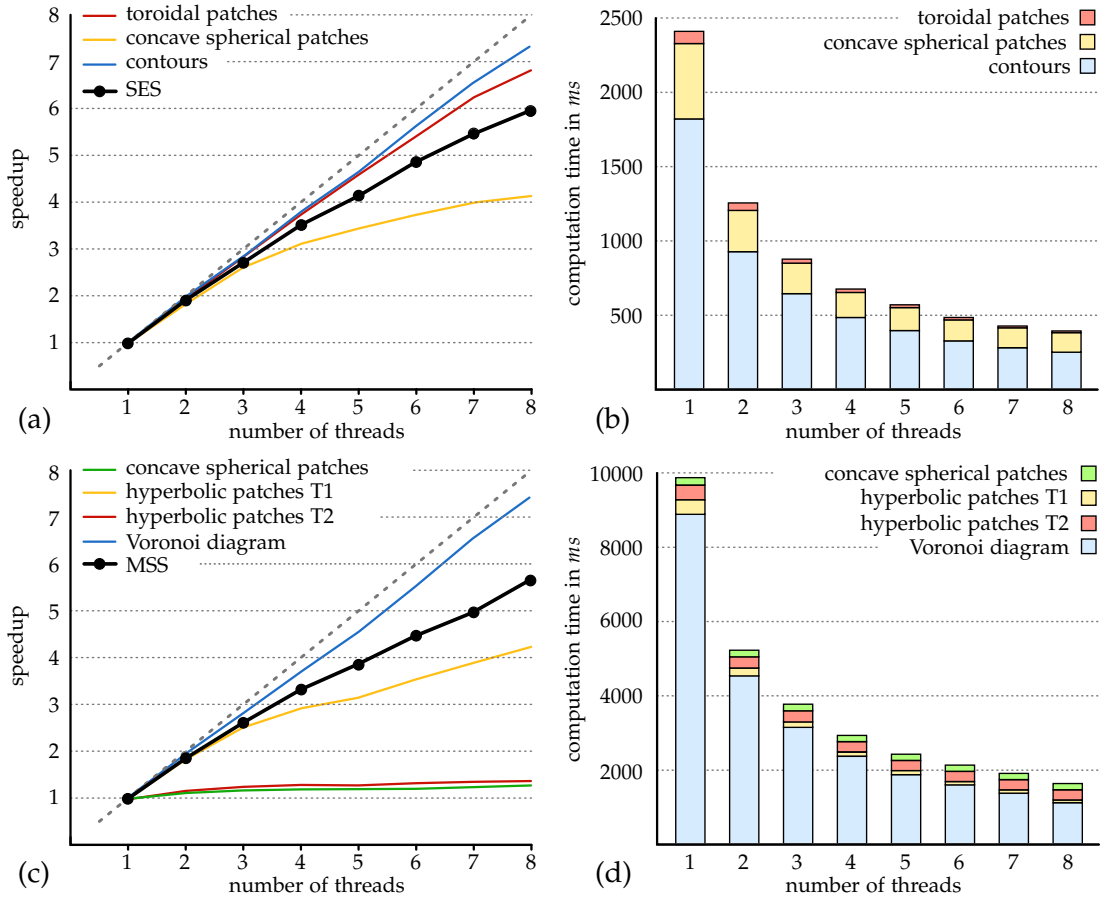


Figure 3.12: *Timings for SES (top row) and MSS (bottom row) of chaperonin molecule (pdb:1AON) on an 8 core 2,53 GHz Intel system. Diagrams (a), for SES, and (c), for MSS, show the speedup for a complete update of the surface depending on the number of threads. Diagrams (b) and (d) show the timings for the individual parts.*

contour-buildup algorithm, for the largest molecule, 1AON, it is faster than the reduced surface algorithm. Note that the computational times for the contour-buildup algorithm include patch generation and data transfer to the GPU. Next to the implementation of the approximate Voronoi diagram algorithm for the MSS, it was also implemented for the SES. However, the contour-buildup algorithm was approximately 1.3 times faster for all molecules in Table 3.1.

Chavent et al. [36] reported a computation time of 15 s for the MSS of a water channel molecule (*pdb:1J4N*). With the approximate Voronoi diagram algorithm the computation time was reduced to 320 ms for the same molecule with $s = 0.3$ on a comparable single CPU core. Hence, the speedup is approximately a factor of 40.

PDB-ID	#Atoms	FR in %	rendering performance (in fps)			
			SES	SES [120]	MSS 0.5	MSS 0.3
1VIS	2 531	60	74	60	39	25
1AF6	10 517	70	46	27	14	10
1GKI	20 150	70	26	19	8	6
1AON	58 870	55	18	13	4	3
3G71	99 174	60	14	–	–	–

Graphics card: NVIDIA GeForce GTX280.

Table 3.2: *Rendering performance of static molecular surfaces. The table shows results for the SES with $r_p = 1.4$ and MSS with $s = 0.5$ and $s = 0.3$ compared to the SES results of Krone et al. [120]. The fill rate (FR) and the resolution (1024×1024) are the same for all renderings.*

3.3.2 Rendering

In order to compare the rendering performances, the same GPU as well as the experimental setup was used as described by Krone et al. [120]. The frame rates are given in Table 3.2. With the tight bounding quadrangles, described in Section 3.2.2 a speed-up of at least 1.3 was measured for the SES. Furthermore, the frame rates for the SES were higher by a factor of 2 to 4 compared to the MSS with $s = 0.5$ and even higher for $s = 0.3$. Additionally, the MSS rendering performance was compared to MetaMol [36]. On an NVIDIA GeForce 8800 GTX, 30 frames per second (fps) were measured for the water channel (*pdb:1J4N*), compared to 7 fps in MetaMol [36] on the same GPU. Thus, the MSS rendering, presented here, is faster by a factor of approximately 4.

For both SES and MSS, typical colorizations and simple transparency using blending were implemented. Furthermore, advanced depth perception techniques like depth darkening [157] and silhouettes were implemented. The overhead introduced by these techniques is constant, because they use simple image filters. For example, for the more expensive depth darkening of *pdb:1AON*, the performance dropped from 18 to 16 fps.

3.3.3 Dynamic Molecular Surfaces

To evaluate the performance for real dynamic molecular data, the algorithms were tested on several molecular dynamics data sets from cooperation partners. The measurements comprise the multi-threaded surface computation using 8 cores as well as the GPU data upload and the rendering. The overall update rates are given in Table 3.3.

In the simulation of DynMol-2, 500 of 4 500 atoms were flexible [31]. Including the neighborhoods of these atoms, the contours of 1 500 atoms needs to be recomputed. With this partial update, a speedup of 2.5 compared to re-computing the whole SES was achieved.

Molecule	#Atoms	overall update rate (in fps)		
		SES	MSS (0.5)	MSS (0.3)
DynMol-1	1 200	110	25	20
DynMol-2	4 500	33	$7-8$	5
DynMol-3	6 500	20	$5-6$	$3-4$

System: 2 Intel Xeon E5540 2.53 GHz, NVIDIA GeForce GTX280

Table 3.3: *Overall update rates for dynamic molecules using OpenMP with 8 cores.*

3.4 Discussion

For the computation of the SES, the contour-buildup algorithm seems to be the method of choice. While it is trivial to parallelize, which is not obvious for the reduced surface algorithm [209], it performs better than the approximate Voronoi diagram algorithm. Furthermore, the approximate Voronoi diagram algorithm has the drawback that it needs more memory, because it also stores additional data structures for the feasible cells. The computation of these additional data structures might also be the reason for its worse running time. Nevertheless, the approximate Voronoi diagram algorithm has the nice property that it can be used for the computation of both MSS and SES. The plots in Figure 3.12 show that the patch computation limits scalability, while the contour-buildup and the approximate Voronoi diagram algorithms alone seem to scale well beyond 8 cores. The most likely reason for the limited scalability is the concurrent memory access from all threads during patch creation. Note also that the patch creation for the MSS has not been fully parallelized.

It seems that using tight-fitting bounding quadrangles as rasterization primitives is the main reason for the improved SES rendering performance (1.3 times faster than Krone et al.[120]). To confirm this, also the point sprite-based approach was tested which was used by Krone et al. [120] and measured frame rates similar to theirs, which supports the explanation. Note that

	construction	rendering
SES	1 core: 1x to [120] 8 cores: 6x to 1 core	>1.3x to [120]
MSS	1 core: 40x to [36] 8 cores: 5x to 1 core 200x to [36]	>3x to [36]
SES to MSS	SES 4x to MSS	SES >3x to MSS

Table 3.4: *Approximate speedups of SES and MSS compared to previous approaches, and speedup of SES over MSS.*

the bounding quadrangles are computed in the geometry shader, so the data transfer to the GPU is the same for the bounding quadrangles and basic point sprites. For the high computational load caused by sphere tracing in the fragment shader, tight-fitting bounding geometries seem to be superior to basic points, which confirms that "geometry shaders [offer] a viable option for the construction of bounding geometry" [80]. Additionally, the Newton-Raphson algorithm is 10-20% faster than the sphere tracing for the rendering of the toroidal patches. However, a few pixel errors occurred at the patch contours due to starting points that are too far from the intersection point. Hence, sphere tracing [86] or the stabilized Ferrari-Lagrange method [90, 120] seem to be the methods of choice.

There are mainly two reasons for the improvement in MSS rendering speed (4 times faster than MetaMol [36]). First, the usage of tight-fitting bounding quadrangles for the convex spherical patches. And second, for the hyperbolic patches corresponding to the Voronoi faces, 3-dimensional polyhedra are used, which are possibly smaller than the mixed cells used in MetaMol [36].

Compared to the SES rendering, the MSS rendering is clearly more costly. This mainly is due to the larger number of patches of the MSS. For shrink factors of $s = 0.5$ and $s = 0.3$, there are approximately 4 times as many patches as for the MSS. Moreover, the rasterization primitives are not fully optimized. Hence, further improvements in the rendering performance might be possible.

The results for dynamic data sets show that the overall update rates are limited by the surface construction. The data transfer to the GPU and rendering times are negligible on an 8 core system.

3.5 Further Developments

Since the publication of the methods described above in 2010, several further developments in this research area have been proposed. In the following, the most important ones are summarized.

In 2010, Krone et al. [121] presented a parallel version of the reduced surface algorithm for the GPU. The method computes in each frame only the visible part of the SES, which makes it in general suitable for dynamic and large molecular data. However, due to the complex parallelization of the reduced surface algorithm, they achieved interactive frame rates only for molecules with at most 2 000 atoms. One year later, they also implemented a parallel version of the contour-buildup algorithm, but for the GPU using CUDA [123]. Due to the massive parallel computation power of the GPU, they could further accelerate the computation. Additionally, Kauker et al. extended the ray casting to visualize the SES with transparency [103].

In 2012, Parulek and Viola presented the first ray casting of the SES which comes without a pre-computation of the analytical description of the surface [180]. Therefore, they use a modified sphere tracing and directly com-

pute the implicit description of the surface along the local neighborhood of the ray. This allows the user to directly visualize the SES of dynamic molecular data without changes in the frame rate. However, they achieved interactive frame rates only for small molecules. Furthermore, the visualization contains pixel artifacts, especially at singularities and patch boundaries. In order to solve these problems, Parulek and Brambilla proposed a new approach that approximates the SES by blending implicit functions [177]. With this approximation they achieved interactive frame rates for dynamic proteins with several thousand atoms. Additionally, the method produces less artifacts, but the patch boundaries are still visible.

Besides to these direct ray casting approaches, Decherchi and Rocchia, presented a new triangulation method based on ray casting [50]. Although they could accelerate the triangulation of the SES and MSS, the overall speed and visual quality for both surfaces cannot reach the quality of the methods presented here.

4 Van der Waals Surface

In the previous chapter, it was shown how to visualize smooth molecular surfaces to interactively analyze dynamic molecular data, especially proteins. While these surfaces are suitable for many molecular visualization tasks, their computation and rendering is too expensive for larger data sets. In order to visualize this kind of data that comprises many length scales and a huge number of atoms, often hierarchical geometrical representations are used. For the lower end of these length scales, the van der Waals surface is a good compromise between simplicity and physical correctness compared to more abstract representations like, for example, the secondary structure representation. However, the basic problem with hierarchical geometric representations is to achieve visually seamless transitions between them. Typically, either its underlying geometry is almost continuously coarsened, or only one or a few discrete changes are made that visually are glossed over using, for example, smooth blending. For both techniques, usually a compromise between performance and visual quality needs to be selected.

In this chapter, a new approach is presented that enables the user to interactively visualize the van der Waals surface, bridging five orders of magnitude in length scale. The method was described in "*Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale*" in 2012 [150]. It shifts the limit where atomic representations must be substituted by simpler geometric objects. Thus, the problem of a seamless visual transition almost vanishes. This is achieved by a simple yet efficient GPU rendering method, based on the idea of modern volume rendering. Furthermore, it will be exploited that biological structures often consist of recurring molecular substructures. The rendering is extended by a three-step surface normal computation in a deferred shading step. This avoids aliasing artifacts and creates an improved structure perception for far camera distances. Since the technique comes with

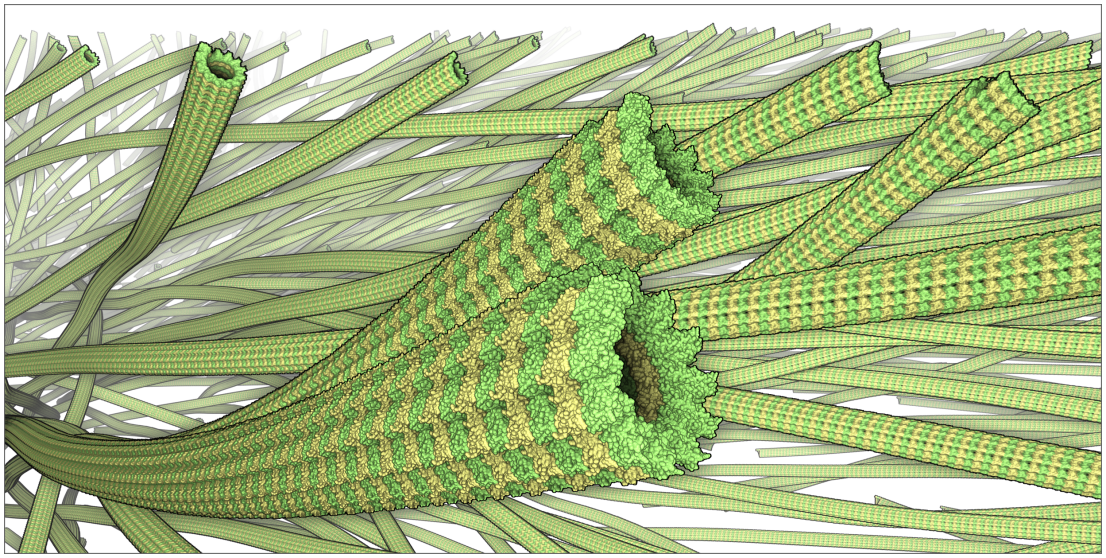


Figure 4.1: *A microtubule data set containing* 4025 *microtubules with approximately* 10 *billion atoms. The microtubules were reconstructed from electron tomography images. The ray casting approach of this chapter is able to render the data set with at least* 3 *fps.*

no explicit level of detail, it requires only a few fast precomputations and is easy to implement. For biological structures, like microtubules and actin filaments, interactive frame rates are still achieved for several billions of atoms. For non-recurring data, like inorganic materials from atom probe tomography, the number of atoms is mainly limited by the memory of the graphics card. The approach can be utilized not only for interactive and exploratory analysis, but also for the production of movies, e.g. for educational purposes in molecular biology and nanosciences. An example of a visualization of a microtubule data set is depicted in Figure 4.1.

4.1 Related Work

The most well known approaches for rendering large scenes use geometry simplification, which is often called level of detail. This is a vast research topic with rather diverse techniques depending on the geometric primitives and the specific goals. It comprises fundamental concepts, like progressive meshes by Hoppe [94] and highly specific approaches, like vessel visualizations by Wischgoll et al. [242]. A broad overview of the most important techniques for simplification of triangular meshes was given by Luebke et al. [156]. More recently, Laine and Karras [128] showed that sparse voxel octrees can be used for efficient occlusion detection during ray casting of large triangulated scenes. However, since the approach presented in this chapter does not use simplification nor triangular meshes, the focus of this section lies on methods dealing with the rendering of large atom or particle data sets.

Most closely related to this topic are probably the works by Sharma et al. [214] and Grottel et al. [79]. Both approaches utilize occlusion culling to speed up the rendering. Sharma et al. apply hierarchical view-frustum culling as well as probabilistic and depth-based occlusion culling. Similarly, Grottel et al. apply a two-level occlusion culling approach. On the coarse level, they employ a grid structure and on the fine level, hierarchical depth buffers [75] are used. To reduce rendering artifacts and to generate coherent impressions of large-scale structures, Grottel et al. make use of deferred shading.

While the method proposed by Grottel et al. as well as the method proposed here render all visible atoms, many other techniques, including the work by Sharma et al., employ hierarchical or level-of-detail representations. For example, Arndt et al. [5] presented the Genome3D viewer, which was specifically designed to visualize structural epigenomic data. To do so, the authors designed level of detail approximations for substructures. This allows them to visualize the whole human genome consisting of approximately 60 billion atoms. Frey et al. [68] visualized a large number of particles by creating representatives that capture the characteristics of the underlying particle density and exhibit coherency. The Millenium Run data set, a large-scale cosmological data set containing 10 billion particles, was visualized by Fraedrich et al. [66]. They achieved this by developing a visually continuous level of detail representation based on a hierarchical quantization scheme for particle coordinates and rules for generating coarse particle distributions. Their approach is well-suited for isotropic density-like particle distributions, but does not apply to molecular models like the van der Waals surface.

In contrast to modern GPU-based ray casting approaches for simple implicit surfaces, as proposed by Sigg et al. [216], the rendering technique that is presented here, exploits the idea of ray-casting-based volume rendering as described by Hadwiger et al. [83]. Instead of creating a primitive for each atom sphere, the molecular data will be stored in regular grids rendered as cuboids. By traversing the cells of the grids, the atoms that intersect the casted ray can be quickly identified.

4.2 Rendering

Biological structures often consist of a large number of recurring parts of a single or a few proteins. To make use of this self-similarity property, components will be created of which many instances can be rendered. Note that these components do not necessarily correspond to a single protein or macromolecule. Rather, it is possible to combine as many molecules as is sensible for a specific application into one such component. In this section, all rendering steps are described in detail. An overview of the different steps is shown in Figure 4.2. Since the rendering method is designed for the GPU, also information about the implementation for OpenGL and GLSL [171] will be given.

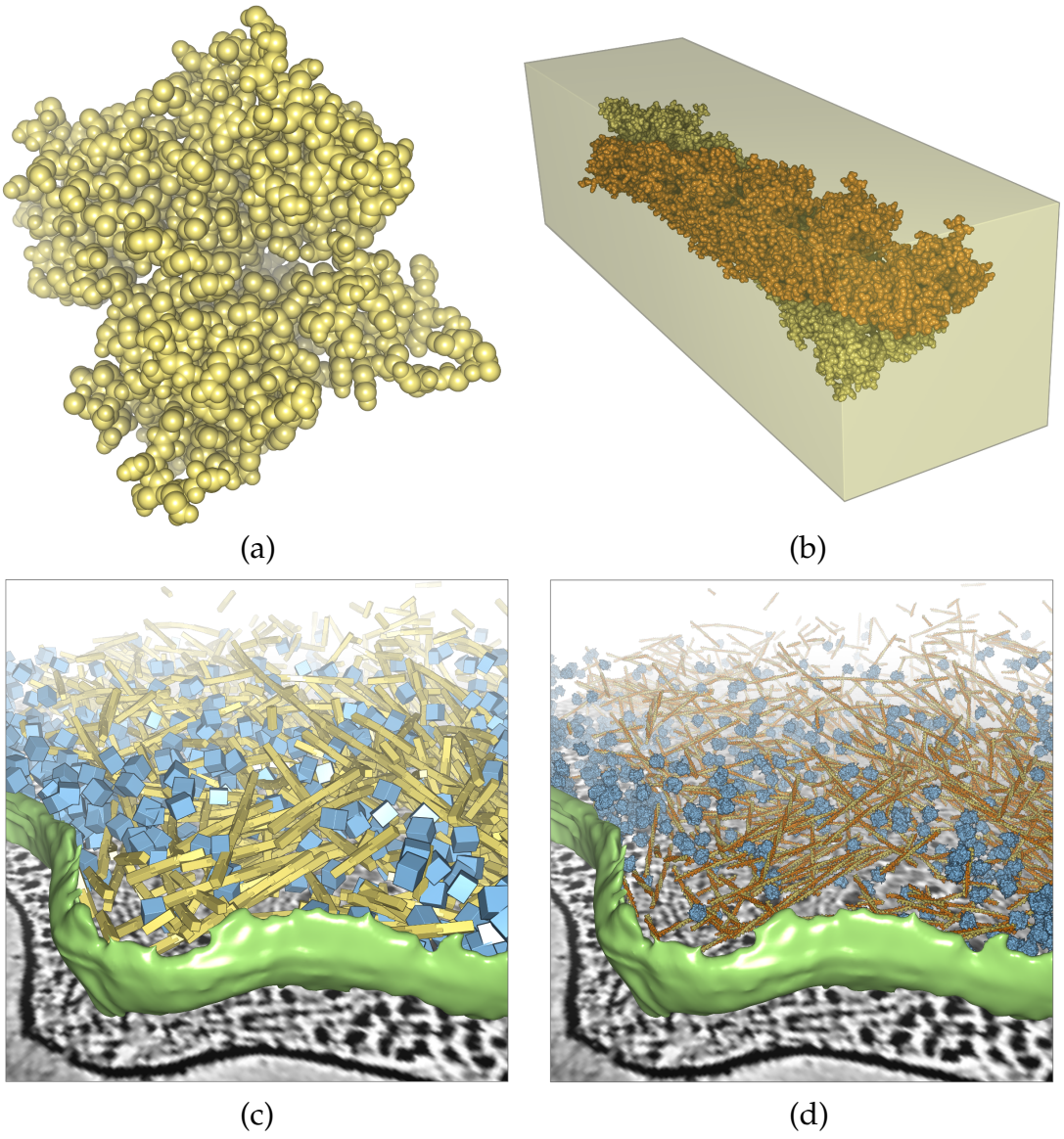


Figure 4.2: *Concept of the rendering pipeline using the example of actin filaments. First, ten actin proteins pdb: 3MFP (a) are used to to create a component (b) of two intertwined strands. Then many instances of this component are constructed with different transformations to form the complete filaments (c). Additionally, ribosomes are shown in blue. Finally, all instances are rendered in a single ray casting pass (d).*

4.2.1 Ray Casting

As described in the previous chapter, modern GPU-based sphere ray casting algorithms create for each sphere a simple primitive. Then, the ray from the camera through each fragment of the primitive is computed and the intersection test with the sphere is performed. While this works very well for up to a few million atoms, for larger data sets, mainly two problems arise. First, the overall number of primitives is too high such that they do not fit into the

GPU memory. Second, the rasterization of so many primitives becomes too expensive. Note that today's typical desktop monitors show images with up to 4 million pixels. Thus, rendering data sets with several billions of atoms implies that many atoms are not visible because they either lie outside the view frustum or they are occluded by other atoms or they are located so far from the camera that they are smaller than a pixel. Especially the two latter reasons are problematic for the current GPU-based ray casting approaches, because for all fragments, even occluded ones, the rasterization and ray casting is performed.

To solve these two problems, the self-similarity of biological structures is used as described above. By storing only a few components of which many instances can be rendered, the memory requirements are enormously reduced (Figure 4.2). Furthermore, a ray casting technique is proposed that renders the components similar to classical voxel rendering methods like volume or isosurface ray casting [83]. All atoms of a single component are stored in a grid data structure. The grid will be represented by a 3-dimensional texture, like voxel data for volume rendering. Afterwards, only the bounding box of the grid of the component is rendered, and the ray casting is performed in the fragment shader. To compute the intersection of the ray with the atoms stored in the grid, two different grid traversal approaches will be investigated that differ both in time and memory efficiency. Once the ray intersects an atom sphere, the traversal can be stopped. This avoids useless intersection tests for occluded atoms. Finally, the correct depth, normal, and color for the intersection point is computed in a postprocessing step

Grid Construction

The atoms of a component are stored in a flexible grid, as proposed in Section 2.5 with the storage strategies shown in Figure 2.22. As mentioned before, two grid traversal approaches will be investigated. For the first approach, an atom is stored into all grid cells that intersect with the atom sphere. For the second approach, an atom is inserted only into a single grid cell, that is, into the cell in which the atom's center lies. The texture buffer object that stores the atom spheres can use either 16- or 32-bit floating point values. Besides this buffer, a further 8-bit unsigned integer texture buffer object is used to store a color identification number per atom. With a 1-dimensional texture, theses identification numbers are mapped to colors during the ray casting.

Instancing and Ray Computation

In order to use the self-similarity property of biological structures, each recurring component is represented by a single grid. Then all instances of a component are rendered by applying different transformations to the same grid. So each instance consists of the bounding box of the component, a rigid transformation T into world space, and an object identification number

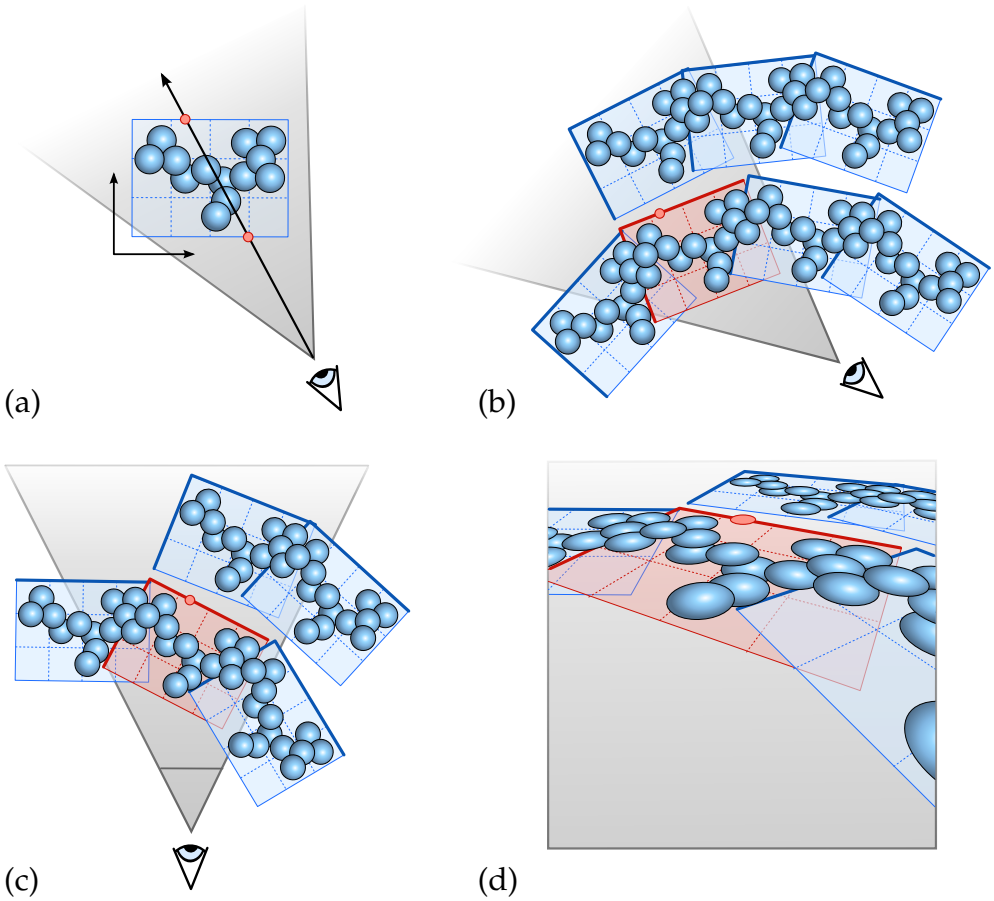


Figure 4.3: *Illustration of the different spaces of the rendering pipeline. Many instances of a component (a) can be rendered in world space (b) using different transformations. Afterwards, each instance is transformed into model view space (c) and projection space (d). Only the back faces (thick lines) of the instance boxes are rendered. The ray casting is performed in component space (a) by transforming the ray from projection space into component space. Finally, the intersection point is transformed again into projection space.*

(OIN). The OIN is used during the shading to distinguish different biological structures, for example, different microtubules.

In many volume-based visualization techniques, the start and end positions of the rays are determined using two separate rendering passes of the front and back faces of the bounding box of an instance. However, rendering n instances separately requires $2n$ rendering passes per frame, which becomes costly for large n. This problem is solved by performing the complete ray casting of n instances in one pass including the determination of the start and end points of the rays. To do so, the back faces of the bounding boxes of all instances are rendered. This creates all necessary rays. Note that the rendering of the front faces would create no rays for an instance when the camera lies inside its bounding box (Figure 4.3).

To perform the ray casting in the space of the component, the rays need to be transformed into this space, too. In more detail, the following is done.

During the vertex shader, all vertices of the bounding box of each instance are transformed from component space into the world space and then into the projection space. This creates the correct coordinates for the rasterization of the bounding boxes. Additionally, the vertex shader pass the vertex positions in component space to the fragment shader, which directly gives the end points of the rays. In the fragment shader, the position of the camera is transformed back from the projection space into the component space by using the inverse rigid transformation T^{-1}. Afterwards, the start position of the ray is determined by computing the intersections of the ray from the camera with the planes containing the bounding box. The intersection point closest to the end position in the direction to the camera is the start position of the ray. If this position is behind the camera, the position of the camera is used as ray start. The different spaces for the rendering pipeline are illustrated in Figure 4.3.

Ray Casting of Atom Spheres

During rasterization of the instances, ray casting of the atom spheres is performed in the fragment shader. For this purpose two different GPU ray casting approaches are investigated, that make use of the grid data structures to efficiently detect occlusion within an instance.

Ray Voxel Traversal Method. This approach was described by Amanatides and Woo [2]. The algorithm is summarized here and illustrated in Figure 4.4. It assumes a grid that stores an atom inside all cells that intersect the atom. Let $r_o + t \cdot \vec{r}_d$ be the ray for the current fragment, where $r_o, \vec{r}_d \in \mathbb{R}^3$ are the ray start and direction, and $t \in \mathbb{R}$ is the ray parameter. First, the cell $c \in \mathbb{N}^3$ is computed in which the ray starts. Furthermore, the ray parameters $t_1, t_2, t_3 \in \mathbb{R}$ are computed that represent the intersections of the ray with the boundary planes of cell c in x, y, and z directions. The final initialization step is the computation of the values $d_1, d_2, d_3 \in \mathbb{R}$, representing the ray parameters to cross a complete cell in direction x, y, and z, respectively.

Now, the following steps are repeated. First, all intersections of the ray with the atom spheres stored in cell c are computed. If the ray parameter $t_i \in \mathbb{R}$ for the closest intersection is smaller than $\min(t_1, t_2, t_3)$, the intersection lies inside c and the algorithm stops. Otherwise, the algorithm proceeds with the next cell. Consider $t_1 < t_2$ and $t_1 < t_3$, which means that the ray hits a neighboring cell in x direction. For a positive x component of $\vec{r}_d$, the x component of c is incremented, otherwise it is decremented. If the new cell c lies outside the grid, the algorithm stops, because the ray does not intersect any atom sphere. Finally, d_1 is added to t_1 and the algorithm jumps to the first step. If t_2 or t_3 is the minimum, c is moved in y or z direction, respectively. Note that this algorithm might compute intersection tests with an atom sphere several times, because an atom sphere can be stored in several cells. Amanatides and Woo suggest the solution to add a flag to each sphere that characterizes that

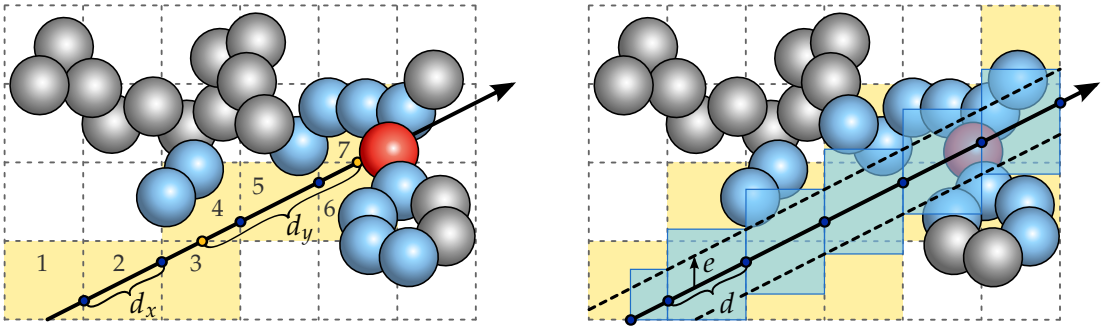


Figure 4.4: *Left: Ray Voxel Traversal in 2D. The yellow grid cells* $1,...,7$ *have to be considered in this order for the ray casting. Hence, intersection test with all blue atoms have to be performed. The red atom is the first atom intersected by the ray. The blue dots depict steps at which* t_x *is minimal and the next cell lies in* x *direction. Accordingly, the orange dots illustrate minima of* t_y*. Right: Ray Layer Traversal in 2D. The main direction in this example is the* x *direction. The cylinder for the ray is depicted by the dotted lines. For each layer,* p_{min} *and* p_{max} *are computed, illustrated by the blue boxes, and intersection tests with the blue atoms stored in the yellow grid cells are performed. The red atom is the first atom that is intersected. The blue dots show the points* p *and* p_n *for each layer.*

an intersection test was already done for the ray. However, this solution is not practicable on the GPU, due to the concurrent traversal of the grid with many rays. A flag for each possible ray would be too expensive.

Ray Layer Traversal Method. This approach is a modification of the ray voxel traversal method for grids that store an atom in a cell only if its center lies inside this cell. Thus the approach uses less memory than the ray voxel traversal method. Another advantage is that one can change the atom radii interactively without recreating the grid. This is interesting for the visualization of the solvent accessible surface (Section 2.3.3).

To perform a correct ray casting for this kind of sphere storage, the algorithm needs to consider all grid cells that intersect a cylinder whose axis is equal to the ray and whose radius is given by the maximal atom radius. Therefore, the algorithm iterates over the cell layers of the grid in the main direction of the ray that is given by the direction component with the largest absolute value. Let $d \in \mathbb{R}$ be the distance on the ray to go exactly the width of one cell into the main direction. Furthermore, let $p \in \mathbb{R}^3$ be the ray start that lies in layer $i \in \mathbb{N}$, and let $p_n \in \mathbb{R}^3$ be the intersection of the ray with the boundary plane of the layer in the main direction. Now, the expansion $e \in \mathbb{R}^3$ of the cylinder is computed in the remaining two directions. This expansion is given by the intersection of the cylinder with the plane spanned by the remaining directions. One can compute a maximal expansion using trigonometric functions. Note that e is 0 in the component of the main direction. In the following, the iteration over the cell layers of the grid is described (Figure 4.4).

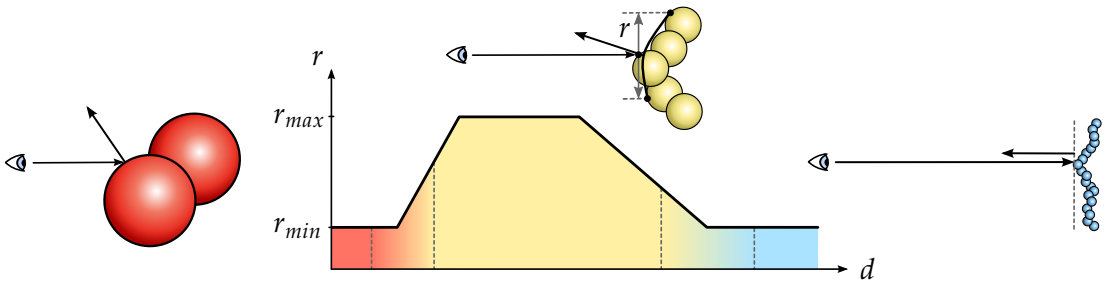


Figure 4.5: *Usage of three normal calculations, depending on the distance d of the fragment: analytically normal (red), approximated normal (yellow), and inverse view direction (blue). The curve shows a possible radius function for the approximated normal.*

First, $p_{min} \in \mathbb{R}^3$ and $p_{max} \in \mathbb{R}^3$ are computed, with $p_{min} = \min(p - e, p_n - e)$ and $p_{max} = \max(p + e, p_n + e)$. Note that min and max work here in a component-wise manner. Furthermore, p_{min} and p_{max} are clamped with the boundary of the grid. Afterwards, the corresponding grid cells $c_{min} \in \mathbb{N}^3$ and $c_{max} \in \mathbb{N}^3$ are detected. Note that c_{min} is equal to c_{max} in the component of the main direction. Now, the algorithm iterates over the rectangle of grid cells given by c_{min} and c_{max} and performs intersection tests with the atom spheres stored inside these cells. Finally p becomes p_n and p_n becomes $p_n + d \cdot \vec{r}_d$, where $\vec{r}_d \in \mathbb{R}^3$ is again the ray direction. This procedure is repeated until the closest intersection point is found or until p and p_n are outside the bounding box of the grid.

Final Calculations. Let $s \in \mathbb{R}^3$ be the closest intersection point of the ray with any atom sphere. To compute the correct depth of the fragment, s is transform into projection space by using again T. Additionally, the normal in s is computed in modelview space and the color is determined from the color identification number and a color map. Depth, normal and color are then written to a frame buffer object for later use. In the alpha channel of the color, the OIN is stored.

4.2.2 Deferred Shading

Deferred shading is often used for fast illumination and post-processing rendering techniques [207, 85]. However, similar to the work of Grottel et al. [79], here, a deferred shading step is used to avoid visual artifacts due to strongly varying normals of spheres that become small in image space. In addition, the shading is extended to emphasize different molecular structures according to the distances of the camera. Therefore, three different normal calculations are used. If a fragment is close to the camera, the correct analytically computed normal is used. With growing distance, this normal is linearly interpolated with an approximated normal given by the surrounding fragments. Finally, for large distances, the approximated normal is interpolated with a constant normal to generate a flat shading (Figure 4.5).

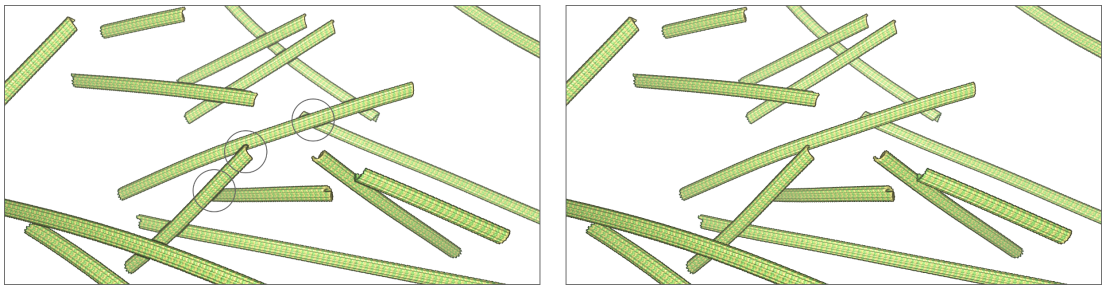
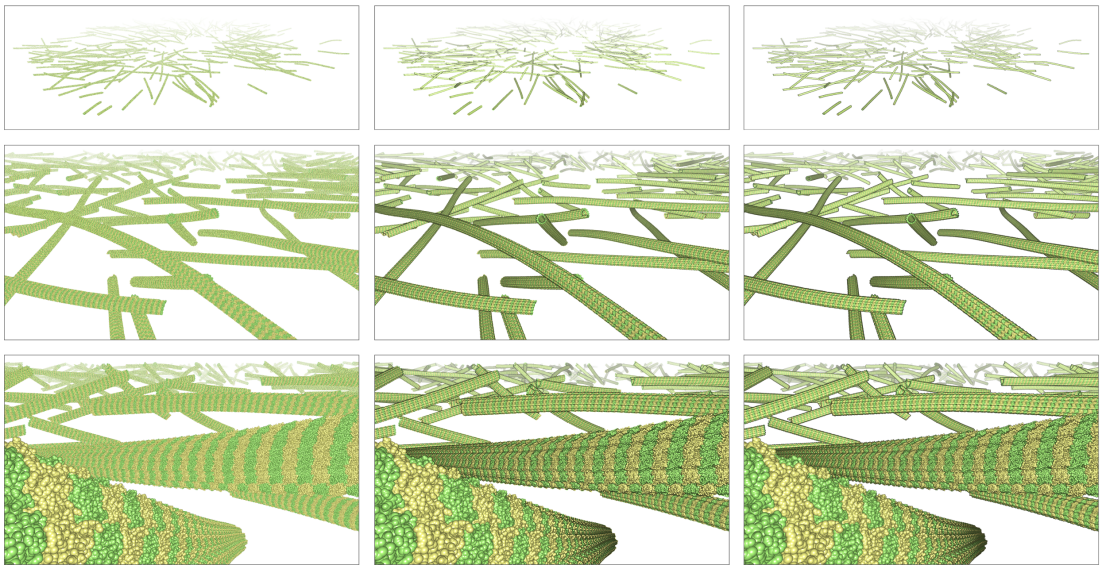


Figure 4.6: *On the left image one can see the problem of silhouettes twice as thick for partially occluded instances. The right image shows the result by computing only silhouettes for the closer instance to the camera.*

To approximate the normal, Grottel et al. use the positions of the eight surrounding fragments and the position of the current fragment as control points for a quadratic Bezier surface. The positions can be computed by the depth values and the inverse projection matrix. The final normal is then given by the normal of the midpoint of this surface.

This method is extended with a radius for the selection of the depth values around the current fragment. The radius is a floating point value and linear interpolation is allowed for depth texture fetches. To illustrate different molecular structures and to avoid hard image changes during camera moves, a function is proposed for this radius depending on the distance of the fragment to the camera (Figure 4.5). First, the radius increases with the distance until a given maximal radius is reached. Afterwards, the radius is constant until the instance becomes very small and finally, the radius is decreased again. Using the radius function, a smooth surface impression of the data is created with increasing camera distance, as the user would expect.

The main problem of the normal approximation occurs when neighboring fragments belong to the background or another structure. For background fragments, Grottel et al. suggest to replicate the data of the current fragment. However, this creates silhouettes when neighboring fragments belong to other structures, while the boundaries of the structure have no silhouettes (Figure 4.7). In contrast to this technique, here always all depth values of the neighboring fragments will be considered. This results in maximal depth values for background fragments. Hence, the approach generates silhouettes at structure boundaries, the thickness of which is given by the radius function. While the silhouettes are suitable for visualization of the molecular structure, they become disturbing if the instance becomes small in image space. For this reason, the radius is decreased after a certain distance. One can also observe that the normal approximation is not sensible anymore when the whole instance becomes only a few pixel wide in image space. Therefore the approximated normal is interpolated with a constant vector representing the normal, which reduces noisy artifacts. Overall, the normal of a fragment changes with increasing distance from the computed analytic normal over an approximated normal depending on a radius function to a constant vector (Figure 4.5).



(a) (b) (c)

Figure 4.7: *Comparison of different shading techniques using a microtubules data set: (a) rendered without deferred shading using the analytically computed normals, (b) implementation of the approach by Grottel et al. [79], and (c) proposed deferred shading using a radius function to increase the perception of molecular structures depending on the viewer distance.*

If one instance is in front of another one and does not occlude it completely, the silhouette between both instances is twice as thick (Figure 4.6, left). This problem can be handled using the OIN. If a surrounding fragment has another OIN and the depth value is smaller than the depth value of the current fragment, the depth of the current fragment is used to compute the position of the surrounding fragment. This avoids silhouettes for the occluded instances (Figure 4.6, right).

Finally, also the color of a fragment is interpolated with the colors of the surrounding fragments depending on the distance to the camera. For a better depth perception, one can optionally add depth cueing. The final results of this deferred shading in comparison to no deferred shading and an implementation of the deferred shading proposed by Grottel et al. [79] are shown in Figure 4.7.

4.3 Data and Applications

In order to test and demonstrate the proposed rendering approach, several data sets were investigated. These data sets and their construction are briefly described in the following.

4.3.1 Materials

With atom probe tomography (APT) it is possible to create a 3D reconstruction representing a field evaporated volume of atoms from a sharp needle-shaped specimen [105]. The reconstruction can consist of tens of millions of atoms, where the position and the element type of each individual atom are identified, offering a near atomic resolution analysis. Visualization of such data sets helps detecting crystallographic structures, lattice defects, as well as clustering of substitutional elements. Examples can be seen in Figure 4.10.

4.3.2 Biological Structures

Among others, two important cellular structures are microtubules and actin filaments, which together with the intermediate filaments form the cytoskeleton of cells. These filamentous structures play a pivotal role in a variety of cellular processes. With electron tomography, it is possible to obtain 3-dimensional images of cytoskeletal structures in cellular environments. However, the resolution of such tomographic maps is not high enough to reconstruct microtubules, actin filaments or intermediate filaments on an atomic level as described in the review by Leis et al. [140]. Hence, these structures are often only visualized using abstract geometric objects like tubes. With the approach presented here, it is possible to bridge the gap between the cellular and the atomic scale, thereby reminding the user of the fact that the phenomena of the data being dealt with are driven by interactions on a molecular or even atomic level.

Structures

- **Microtubules** are tube-like polymers of the protein tubulin [95] (Figure 4.1). The main building blocks are α- and β-tubulin. A third type of tubulin is γ-tubulin, which can be found at the minus ends of the microtubules, contributing to a cap. The α- and β-tubulin build dimers, many of which are arranged in a helical structure. Microtubules form the core of organelles like axonemes but most importantly they form bipolar spindles, structures involved in chromosome segregation mitosis. Depending on the species, the mitotic spindle apparatus can consist of several thousands of microtubules.

- **Filamentous (F) actin filaments** are formed by polymerization of globular actin proteins. A single F-actin filament appears as two intertwined strands forming a helical structure (Figure 4.8). The polymerization and depolymerization of actin into higher order structures, e.g. branched networks or bundled fibers, is driving cytoplasmic organization and cell motility [193].

- **Ribosomes** are cellular machines that assemble the aminoacids to form proteins [199]. Because of the importance of ribosomes, many of these

macro-molecular structures are present in each cell (Figure 4.8). Of high interest is the distribution of ribosomes, because it gives hints about regions of high protein biosynthesis.

Data Preparation

Both microtubules and actin filaments are reconstructed from electron tomograms [203, 238]. As result of the reconstructions, the structures are given by their piece-wise linear center lines. The recurring atomic data is taken from the protein data bank [184]. To create the atoms for the center lines, first the lines are smoothed and resampled such that the length of each line segment is equal to the length of one recurring component. Then, for each line segment, one instance of the component is created and placed along the segment to form the biological structure.

- For the microtubules, the tubulin protein *pdb: 1TUB* is used, which comprises both α- and β-tubulin. Thirteen of these building blocks were placed around a line segment with a radius of 12 nm to form one left-handed spiral cycle with an extend of approximately 12 nm of the helix. Ten consecutive cycles were combined to form one recurring component of a microtubule. Several tens of instances of this component were then placed along each line to form a complete microtubule.
- As building blocks for the actin filaments, the protein *pdb: 3MFP* is used, which comprises two actin monomers, one for each strand. Twelve of these building blocks were concatenated to form one component. The blocks were placed along a line segment each rotated by the same amount of degree. Several instances of this component were placed along the whole line forming a complete actin filament.
- For the ribosomes, only the positions were extracted from the electron tomograms. Hence, the orientations are arbitrarily chosen. To represent the ribosomes, *pdb: 2WDK* and *pdb: 2WDL* are combined into one component. At each position of a ribosome, a single instance is placed.

4.4 Results

For the data sets described above, the following results were achieved. These results are compared to the rendering performance of the fastest previous approaches. Before going into the details, the chosen parameters are given.

4.4.1 Parameter Choice

The number of maximal grid cells, see Section 2.5, is very important for balancing the performance and memory requirements. For the ray voxel traversal, a number of cells equal to the number of atoms performed best in most

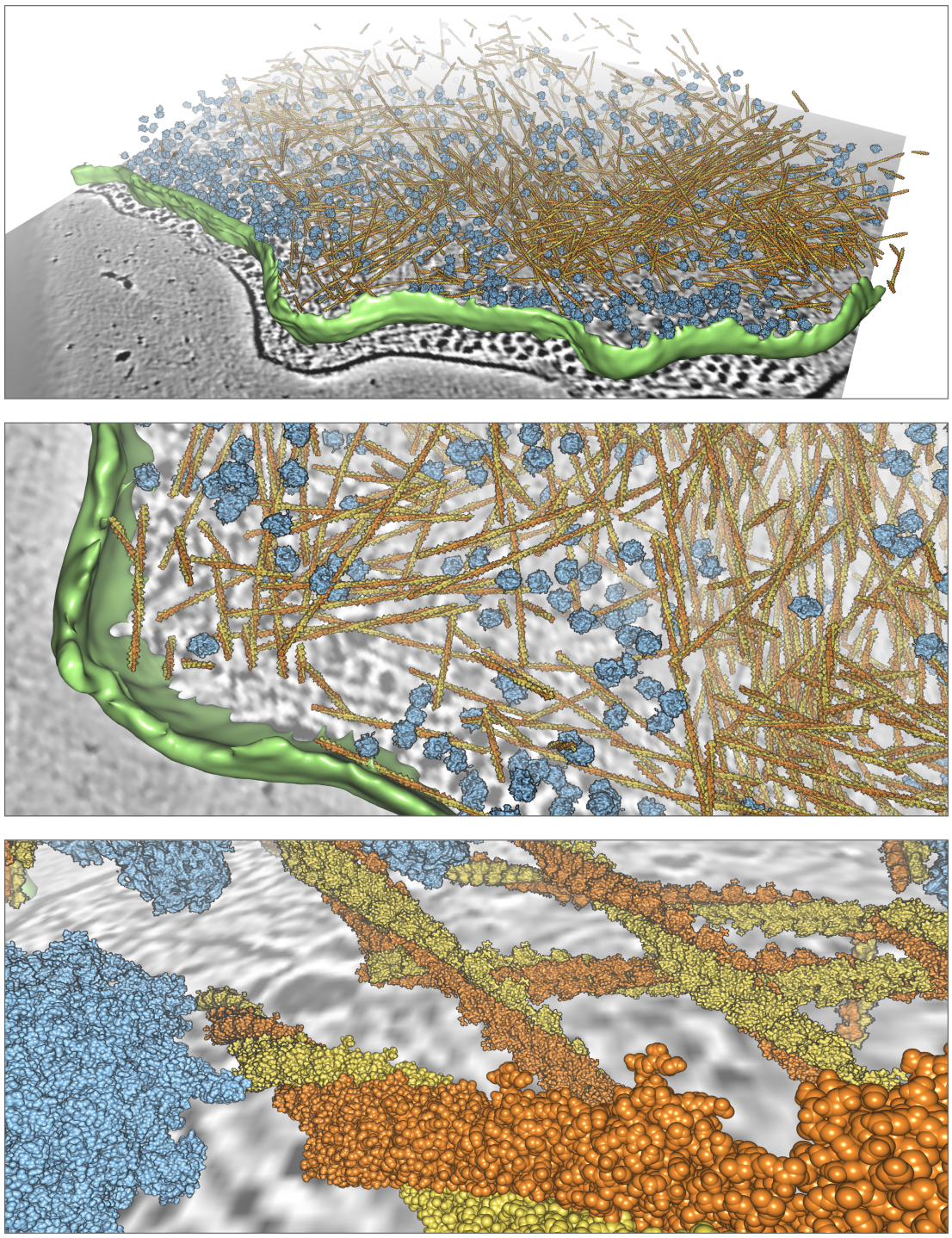


Figure 4.8: *Illustration of different perspectives of a reconstruction consisting of actin filaments (yellow/orange) and ribosomes (blue) from an electron tomography image. One image slice is shown as gray-value map in the background. Part of the membrane (green) was reconstructed as triangulated surface.*

Data	#Atoms	#API	FR	FPS (V)	FPS (L)	FPS (C)	MB (V)	MB (L)
1X9P (virus)	220 600	220 600	58	61	42	185	8.8	2.7
1OHG (virus)	904 200	904 200	76	41	25	105	29.7	9.9
1TUB (microtubules)	2 700 000	900 000	67	50	30	41	39.5	9.8
3MFP (actin)	7 850 000	35 500	57	29	18	15	1.3	0.5
APT (ion crystal)	36 000 000	36 000 000	77	26	15	1	1 092.6	362.3
APT (synthetic)	75 500 000	75 500 000	59	–	13	–	–	831.4
1TUB (microtubules)	148 500 000	900 000	63	15	8	–	39.5	9.8

Graphics card: NVIDIA GeForce GTX 470.

Table 4.1: *Performances in frames per seconds (FPS) for the ray voxel traversal (V), the ray layer traversal (L), and the classical sphere ray casting (C). The table also shows the number of atoms per instance (#API), the fill rate (FR), and the memory requirements for the grid and the atom data in MB, where the atom positions are stored as 16-bit floating point values.*

cases. In contrast, for the ray layer traversal, only one fourth of this number leads to the maximal performance for the test data sets. The remaining parameters are needed for tuning the deferred shading. For r_{min} and r_{max} of the radius function, see Figure 4.5, 0.25 and 2.0 pixels were used, respectively. The radius is increased at a distance of 35 nm and the maximal value is reached at 300 nm. As already mentioned, the distance for the constant maximal radius depends on the biological structure. For all structures, the radius is kept constant until the instance is larger than 5 pixels in image space and it reaches its minimum again when the instance becomes smaller than 2 pixels. Furthermore, the analytical normal is interpolated with the approximated normal between 20 nm and 50 nm. The interpolation of the approximated normal and the inverse camera direction is done between the instance sizes of 3 and 1 pixels.

4.4.2 Ray Casting Performance

The technique was tested for several biological data sets as well as materials from ATP. The results were compared to the typical GPU-based ray casting of spheres similar to the approach by Sigg et al. [216]. A direct comparison to the optimized approach of Grottel et al. [79] was not possible, because of the specific data sets. However, a comparison based on the order of the number of visualized atoms seems to be sufficient.

For the performance tests, an NVIDIA GeForce GTX 470 graphics card was used with a fixed resolution of 1024×1024. Since the performance of all approaches depends a lot on the fill rate of the image as well as the number of occluded fragments, the data sets were placed in a way, such that approximately the worst case frame rate with respect to the distance of the data set to the camera was achieved. Then, the data was rotated around its center of gravity and the average fill and frame rate was taken. The results are given in

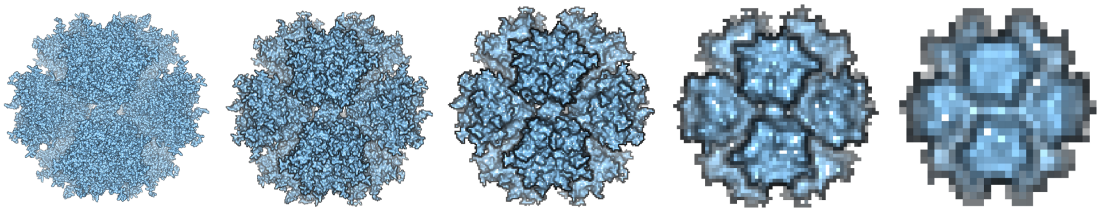


Figure 4.9: *Illustration of the virus capsid 1X9P for different distances to the camera. One can see that the smoothness of the surface increases with the distance.*

Table 4.1. The first two data sets are icosahedral virus capsid structures from the VIPERdb data base [233] (Figure 4.9). The third and the last data sets are small sections from microtubule data sets. A part of an actin filament data set was used as fourth example. The whole data set can be seen in Figure 4.8. The remaining two data sets are atom probe tomography examples (APT), where the smaller one is a metallic ion crystal (Figure 4.10) and the other one is a synthetic silicium data set created for performance tests. The table shows the results for both ray traversal methods and the classical GPU-based ray casting of spheres which is similar to the technique presented by Sigg et al. [216]. One can see that the ray voxel traversal method is nearly twice as fast as the ray layer traversal. Furthermore, the classical ray casting is much faster for small molecular data, but its performance decreases rapidly when the data sets reach 5 million atoms. The proposed rendering technique scales much better with the number of atoms and clearly outperforms classical ray casting of spheres for 10 million atoms and more. While the ray voxel traversal is faster than the ray layer traversal, it requires $2-4$ times more memory to store a component. For this reason it was not possible to render the large APT data set with the voxel traversal on this graphics card. So the size of a single component is mainly restricted by the memory requirements of the grid data structure for the corresponding ray traversal method. For the GeForce GTX 470 graphics card, the largest component can consist of approximately 50 million atoms for the ray voxel traversal and 100 million atoms for the ray layer traversal. Using many instances of a recurring component allows reducing the memory requirements a lot. Note that the frame rate does not directly profit from the instancing, but depends mainly on the overall number of atoms and the camera position. The frame rate changes approximately linearly with the number of rendered fragments. When zooming, the number of fragments changes mostly continuously. Thus, the frame rate also changes continuously.

The proposed technique has a similar performance as the approach by Grottel et al. [79] for up to 100 million atoms but scales better beyond this. This claim is based on performance measures for the APT data sets, which seem to have similar atom densities as their laser ablation simulations. For their data set with 48 million atoms, Grottel et al. achieved a frame rate of about 7 fps for the sphere ray casting on an NVIDIA GeForce GTX 285. On the same

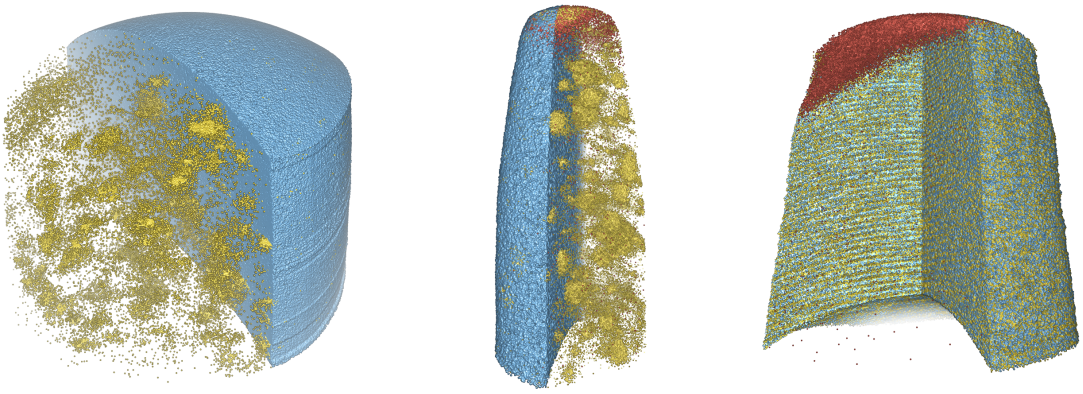


Figure 4.10: *Visualization of three different APT data sets using interactive cutting planes. This allows analyzing clusters (a), (b) or the lattice structure (c).*

graphics card, the smaller APT data set from Table 4.1 was rendered with 23 fps and 14 fps for the two traversal methods presented here. Besides, the performance does not depend on the previous frame.

The rendering approach was also applied to larger data sets with up to billions of atoms. The whole actin data set, illustrated in Figure 4.8 comprises about 700 million atoms and the large microtubules data set, shown in Figure 4.1, contains about 10 billion atoms. Therefore, the overall scale ranges from 0.2 nm for a single atom to 8 000 nm for the whole data set. For the actin data set, an average frame rate of 8 fps was achieved depending on the zoom factor and the view direction. For the large microtubules data set, the frame rate was at least 3 fps.

4.4.3 Deferred Shading

The deferred shading is relatively cheap in contrast to the ray casting of the spheres. The complete deferred shading step takes about 1.7 ms per frame. Hence the frame rate drops, for example, from 25 to 24 with deferred shading. However, the shading greatly improves the perception of the molecular structure, similar as the approach by Grottel et al. [79]. The differences due to the silhouette handling and the radius function are illustrated in Figure 4.6 and 4.7. With increasing distance, the visualization becomes more and more like a smooth surface and shows the important structures for the corresponding distance (Figure 4.9). Nevertheless, the deferred shading produces minor flickering artifacts at cavity boundaries as well as at the molecular boundary. These artifacts are created by great distance differences between neighbored pixels as well as their abrupt appearance and disappearance.

4.5 Discussion

Although the approach, presented in this chapter, is one of the fastest techniques to visualize large atomic data, it comes with some limitations.

The approach is currently only designed for static data and limited to rigid transformations for the recurring substructures. In addition it is not suited for all data sets consisting of spheres. Especially, large variations in the radii and very sparse or not well distributed data sets can decrease the performance a lot. However, typical molecular data sets do not fall under this category.

Since most molecular structures have a homogeneous atom density, a grid should perform best for the storage of the atoms. The initialization of a grid is very fast and its conversion into a 3-dimensional texture is straightforward. Furthermore, the implementation of the ray traversal methods using GLSL is quite easy. The main limitation is the memory requirement of the grid structure. Maybe other data structures, like octrees or k-d trees, are less memory demanding and can possibly better deal with more general data sets of spheres. On the other hand, the ray traversal methods are usually more complex for these spatial data structures.

Some flickering artifacts can be observed during camera changes. It needs to be investigated whether it is possible to handle the depth outliers by a further rendering pass before the deferred shading. The depth difference should be clamped depending on the distance to the camera for these pixels. The flickering at the data and cavity boundaries can possibly be handled with a motion blur effect or a larger kernel for the normal approximation.

So far, occlusion culling is not used explicitly but only implicitly on the atomic level of a single component by applying a grid-based approach, but not on the instance level. This means, that instances occluded by other instances will still be rendered. Thus occlusion culling on the instance level could further speed up the rendering. At least two approaches are possible. Similar to the cell level occlusion culling applied by Grottel et al. [79], occlusion culling could be done on the instance level. Another possibility is to use a hierarchical approach by inserting all instances of all components into another grid similar to the component grid. Then, instead of rendering the grids of all instances, only the grid containing the instances will be rendered. This could be done on a per-fragment basis.

In 2012 and 2013, Falk et al. [62, 63] already presented two improvements to accelerate the rendering. First, they subdivide the rendering of the instances into several passes. In each pass only a subset of the instances is rendered. The depth information are extended in the pass and the result of the previous pass is used to early detect occlusions. The second improvement they call hierarchical ray casting. This technique does not perform a ray casting for the atom spheres if a grid cell becomes smaller than a pixel in view space. It is only checked if the cell is empty or not. For the latter case an intersection with the first atom is assumed. A similar technique is used if the whole grid becomes smaller than a pixel in view space.

More recently, Le Muzic et al. [138, 136, 137] presented a series of works that follow the direction of explicit level of detail to render large biological scenes with thousands of proteins. For this purpose, atom clustering techniques are applied in combination with hierarchical Z-buffer (HZB) occlusion culling [75]. In addition, they present advanced clipping techniques that allow exploring dense data sets.

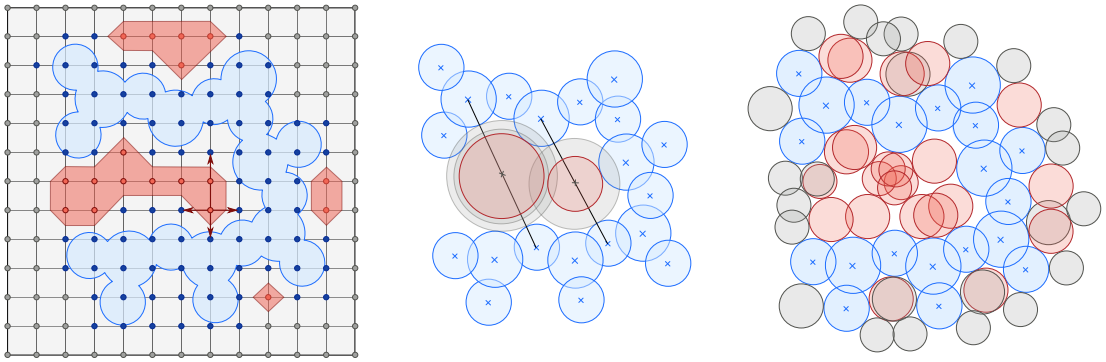
With the approach presented here, the limit where atomic representations of complex objects must be substituted by simpler geometric objects is shifted so far, that it should be easily possible to replace the atomic representations without creating visual artifacts. Thus, when rendering even larger scenes containing several cells, level of detail representations should and could clearly be applied. With these extensions and further advances in the graphics hardware, it could become possible within the next few years to render a whole cell with all its cellular structures at almost interactive speed.

5 Cavities: A Survey

In the previous two chapters, the computation and rendering of molecular surfaces was presented to visualize and analyze the results of molecular simulations or reconstructions from microscopy data. The following chapters deal with the complementary space of molecules, which represents the cavity structure. Cavities are the main places where molecular interactions take place and thus of particular interest for biophysicists. In order to investigate the cavity structure in molecules, a novel method to compute, analyze, and trace cavities over time, based on the molecule's geometry, will be proposed. However, first a survey of existing methods is presented, which is partially based on "*Visual Analysis of Biomolecular Cavities: State of the Art*" [125] from 2016.

The detection of molecular paths and cavities is a large research area with many different approaches. Over the past two decades, several methods to explore and visualize molecular paths and cavities have been proposed. However, most of them are restricted to a small subset of paths. Others concentrate on the detection of all tunnels in a molecule but do not find paths in general. And many methods do not compute path descriptions but only the surface of possible cavities. Despite the large number of publications in this field, in the following, an extensive overview of the most important works dealing with geometry-based extraction and identification of cavities and paths is given.

One can distinguish two groups of methods. The first group comprises all algorithms that directly compute the shape of the cavities in the molecule. This is often realized by analyzing a molecular surface or by filling the cavities with geometric structures. The second group on the other hand contains all methods that compute possible molecular paths. Often it is simpler to compute the shape of the cavities from these paths than the opposite. Of



(a) (b) (c)

Figure 5.1: *Illustrations of the algorithms in POCKET (a), SURFNET (b), and PASS (c). Pocket: the blue grid points indicate points close to the protein and the red are grid points that hit protein points in at least one main direction along positive and negative side. Surfnet: the detection of two gap spheres is shown. PASS: detection of different layers of tangent probe spheres. The grey spheres are filter either because they lie too close to other probe spheres or their burried value is two small.*

course, some algorithms cannot be clearly classified into one of these groups, so that some assignments are probably arguable.

5.1 Category I: Cavity Computation

One of the first approaches to compute and visualize cavities is POCKET [141] developed by Levitt and Banaszak in 1992. The algorithm creates a 3-dimensional cubical grid with a user-defined cell width, which is typically around 1 Å. For each grid point, the closest distance to an atom center is computed. If this distance is smaller than a predefined threshold (usually 3 Å), the grid point is marked as protein contact point. Then, the neighboring grid points in the three main directions of each unmarked grid point are investigated. If such a point is bounded by protein contact points along both sides of at least one direction, the density of the point is set to 1. Note, that the density is initialized with 0. The initial idea of this grid-based density approach already comes from Voorintholt et al. [235]. Finally a modified Marching cubes algorithm is used to compute the surface of the cavities (Figure 5.1). Because of the small number of directions, that are investigated for each grid point, the result depends a lot on the orientation of the molecule. Furthermore, and this holds for all following grid-based methods, the geometrical accuracy as well as the computation time and memory requirements depend greatly on the resolution of the grid.

In 1994, Kleywegt and Jones developed the tool VOIDOO [116] to detect closed cavities in molecules. The tool computes the solvent accessible surface (Section 2.3.3) for a given probe on a discrete grid. Afterwards, all grid points that can be reached from the boundary of the grid are removed. All remaining

grid points outside the SAS are points inside closed cavities. These points can be used to create a surface of the cavities or to measure their volumes. The procedure is repeated several times with increasing scaling values for the atomic radii. The scale factor that creates the most cavities is finally used for further analyses. However, the detection of this factor is not trivial, and small variations can change the results a lot. Due to the nature of the algorithm, only closed cavities can be detected but not channels or pockets.

One year later, Laskowski presented a tool which he called SURFNET [131]. It fills the cavities in a molecule with *gap spheres* that do not penetrate the atom spheres. In more detail, between each pair of atoms a gap sphere is placed in the middle, touching the two atom spheres. Afterwards, it is checked if no other atom sphere penetrates the gap sphere. In case of a penetration, the radius of the gap sphere is reduced (Figure 5.1). If the radius falls below a user-defined threshold, the sphere is completely rejected. Finally all gap spheres are sampled into a 3-dimensional grid using Gaussian density kernels. From this grid a surface of the cavities can be easily generated. The main shortcomings of this method are the time complexity, which is cubic, and the geometric accuracy, which is not optimal, due to the fix position of the gap sphere.

A similar approach to SURFNET is PASS [26] presented by Brady and Stouten. The method adds an initial layer of probe spheres where each probe is tangent to three atom spheres but does not penetrate any atom sphere. For each probe, a buried value is computed, which is the number of atoms whose distance to the probe is smaller than a given radius. If this value is smaller than a selected threshold, the probe is removed. Furthermore, probes are filtered such that no two probes are closer than 1Å. Afterwards, additional layers with smaller probes will be added tangent to the previous layers, and the probes are filtered in the same way. This is repeated until all probes of a new layer are filtered (Figure 5.1). Based on the buried values of the probes, the centers of the cavities are computed.

Another approach was presented by Ho and Gruswitz which is implemented in their tool HOLLOW [93]. But instead of placing a sphere between each pair of atoms or tangent to atoms, they place them directly on a grid with a fixed sphere size. Afterwards all spheres that penetrate the atom spheres or that lie outside the envelope of the molecule are removed from the grid. The remaining spheres of the grid are used as dummy atoms whose molecular surface represents the surface of the cavities (Figure 5.2).

In contrast to a sphere placement, Yu et al. presented an algorithm, called Roll [247], which is based on the solvent excluded surface (Section 2.3.3). The volume of the cavities is defined as the difference of the volume enclosed by the SES and the volume enclosed by the van der Waals surface. To compute the difference efficiently, they sample the van der Walls surface into a 3-dimensional grid. Then, the SES is sampled by rolling the probe sphere along the grid without intersecting the atom spheres. The grid points between the SES and the van der Waals surface lie inside cavities. All cavities

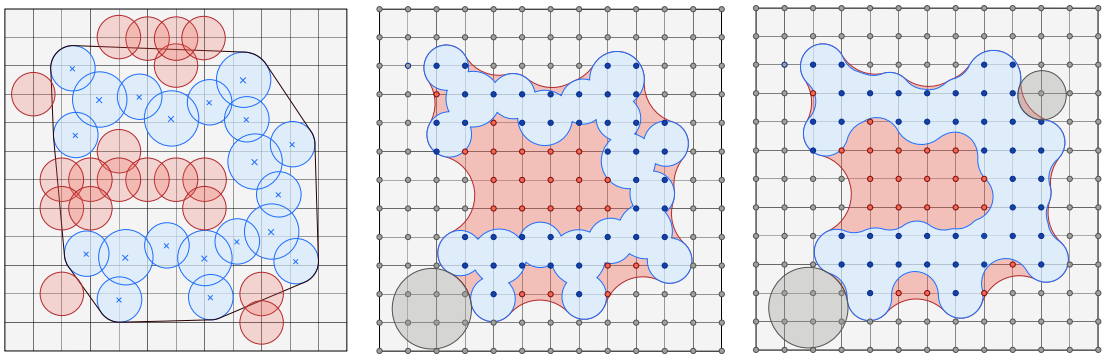


Figure 5.2: *Illustrations of the algorithms in HOLLOW (a), ROLL (b), and 3V(c). HOLLOW: probe placing on a grid, where probes that intersect the molecule or lie outside its envelope are rejected. ROLL: all grid points (red) between the van der Waals surface and the solvent excluded surface are defined as cavity points. 3V: all grid points (red) between two solvent excluded surfaces with different probe radii are defined as cavity points.*

that are completely surrounded by the van der Waals surface are specified as closed cavities. On the other hand, cavities that are partially surrounded by the SES are denoted as pockets (Figure 5.2). The tool, in which Roll is integrated, is called POCASA.

A generalization of Roll was developed by Voss and Gerstein, called 3V [237]. They compute the solvent excluded surface for two different probe spheres. The first probe approximates the solvent of interest and the second is larger and it is used to close all outer pockets of the molecule. For this reason they call the second surface *shell*. The volume of all cavities is defined as the difference of the volume enclosed by the shell and the volume enclosed by the SES of the solvent (Figure 5.2). In order to compute this in a robust way, they also use a discrete grid to compute the SES for both probe spheres. In 2014, nearly the same method was proposed again by Oliveira et al. in their tool KVFinder [173]. In addition, Desdouits et al. [52] extract cavities in the same manner to study their evolution throughout MD simulations.

In 2010, Kawabata presented the tool GHECOM [104], which is quite similar to 3V [104]. The approach computes the SES for different probe radii. The differences in the volumes of these surfaces provide the cavity information. In contrast to 3V, where only two probes are used, this approach applies multiple probes in order to rate the accessibility of the pockets. The definition of these "multiscale" pockets is based on morphological operators for the van Waals surface and the probe spheres. To achieve robust and efficient calculations, the algorithm is discretized on a 3-dimensional grid.

A different approach is LIGSITE [89], presented by Hendlich et al. The tool maps the SAS into a 3-dimensional grid. Afterwards, for each grid point outside the SAS, all neighboring grid points are investigated within 12 Å into the 3 main directions and the 4 cubic diagonal directions. If along both sides of a direction a grid point lies inside the SAS, the direction is marked as

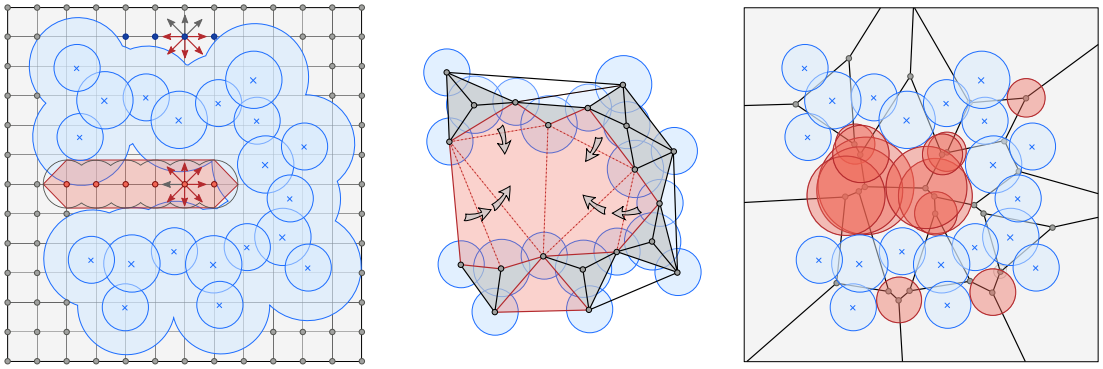
protein, solvent, protein (PSP). All grid points with at least 2 PSP directions are marked as cavity grid points and will be clustered. The surface of the cavities is obtained by sampling the solvent probe sphere at each cavity grid point (Figure 5.3).

A similar method by Exner et al. [61] maps the SES into a discrete grid representation. For each grid point outside the SES, the grid points in the three main directions are investigated within a given neighborhood radius. If at least two directions contain grid points that lie inside the molecular surface in positive and negative direction, the investigated grid point is marked as cavity grid point. All cavity grid points are combined in clusters on which contraction and expansion operations are performed. The final clusters represent the cavities. The main shortcomings are the limited detection directions. Depending on the neighborhood radius this can lead to missing cavities whose medial shape axis is aligned diagonal to the main directions.

To overcome the limitation of only investigating 3 or 7 directions, Weisel et al. developed the tool PocketPicker [239]. For each grid point that does not lie inside an atom sphere and whose minimal distance to the atom spheres is smaller than a user-defined threshold, a uniformly distributed set of 30 rays is cast. The rays are computed by subdividing an octahedron. For each ray the surrounding atom positions are orthogonally projected onto the ray. If the distance between an original atom position and its projected position is smaller than 0.9 Å, and the distance between the grid point and the projected position is smaller than 10 Å, the ray is marked as buried. For 16–26 buried rays, a grid point is defined as point inside a pocket. Grid points inside pockets are again clustered. Additionally, the shape of a pocket is described by evaluating the buried values and distances of all pairs of grid points in a single 420-dimensional vector.

In 2006, Huang and Schroeder extended the original LIGSITE algorithm by using the SES, like Exner et al. They called the new version LIGSITE^{CS} [97]. Additionally, the conservation of the neighboring residues of the three main pockets is analyzed to rate the availability of the pockets. With this additional feature, the algorithm is called LIGSITE^{CSC}. In a further work, Huang presented the tool MetaPocket [96], which combines the results of LIGSITE^{CS}, PASS [26], SURFNET [131], and Q-SiteFinder [133] to improve the identification of possible binding sites. Q-SiteFinder [133] generates a 3-dimensional grid and computes at each position the non-bonded interaction energy with the program Liggrid. Using a threshold, all grid points with a high binding energy are marked and clustered. Finally, volume calculations are performed. More recently, Huang and Schroeder presented MetaPocket 2.0 [249] which takes into account four further tools, namely Fpocket [135], GHECOM [104], ConCavity [33], and POCASA [247].

In contrast to the previous approaches, An et al. proposed a more physically-based technique, which is implemented in the tool PocketFinder [3]. Like many other approaches they use a grid for the cavity detection. But instead of geometrical properties, they compute at each grid position the Lennard-



(a) (b) (c)

Figure 5.3: *Illustrations of the algorithms in LIGSITE (a), CAST (b), and FPocket(c). LIGSITE: at each grid point outside the SAS a set of rays is cast along the main directions and the diagonals. If two rays hit a grid point inside the SAS along negative and positive direction, the point is marked as cavity point. CAST: the cavities (red) are defined as the Delaunay elements that do not belong the α-shape (gray). FPocket: a clustering of filtered α-spheres (red), placed at Voronoi vertices, describes the cavity structure.*

Jones potential of a carbon probe atom (Section 2.2.1). In the next steps, they smooth the discrete potential field and compute a threshold using the average field value and the root mean square distance of all values. With this threshold the cavities are given by the isosurface of the discrete potential field. Finally, the cavities the volume of which is smaller than 100 Å are filtered out.

A similar approach is SITEHOUND [91], proposed by Hernandez et al. The tool generates a grid and computes at each position the binding affinity of either a carbon or a phosphate atom. Therefore, the non-bonding interactions between the atoms are investigated. Subsequently, only grid points with a high binding affinity are considered and clustered to get potential cavities.

Capra et al. presented the tool ConCavity [33], which makes use of Ligsite, Surfnet, and PocketFinder for the geometrical cavity detection. These algorithms are extended by a 'voting' of the cavities, which is based on the sequence conservation of the surrounding residues. Therefore, the authors used the Jensen Shannon divergence [34].

Edelsbrunner, Liang and others presented a series of papers dealing with cavity detection and cavity analysis based on α-shapes. These works, finally, lead to the tool CAST [146]. The approach is based on the α-shape theory by Edelsbrunner and Mücke [59] as well as their preliminary works [53, 55, 56, 57, 145, 143, 144]. The α-complex is a subset of the Delaunay complex. Each Delaunay element whose dual Voronoi element has a closer minimal distance to the atom positions than $\alpha \in \mathbb{R}$ is also an element of the α-complex. Note that the probe radius is integrated in the α value. All tetrahedra of the Delaunay complex that are not part of the α-complex lie inside a cavity. Two tetrahedras inside a cavity are neighbored if they share a common triangle.

Thus neighbored tetrahedras can be clustered to analyze the volume and type of a complete cavity (Figure 5.3).

In their first publication, Edelsbrunner et al. [55] described the detection of internal cavities and the analytical computation of their volumes. These cavities can be easily extracted from the α-shape. In a subsequent work, they extended the cavity computation to the pocket detection [56]. The approach uses the discrete flow of the Delaunay tetrahedras to define and identify the pockets (Figure 5.3). Later, they presented the tool VOLBL to compute all measurements, such as volume and area, for internal cavities and pockets [143, 144]. Finally they integrated the detection and measurements into the tool CAST [146]. Although the intersections of the atom spheres with the cavity tetrahedra are considered during the volume and area computations, the shape of the cavities is only roughly approximated by the Delaunay tetrahedras. Thus, the approach has limited visualization possibilities. Additionally, shallow pockets can not be detected by the algorithm.

Furthermore, Sridharamurthy et al. [222] used the α-complex to identify robust voids and pockets that are stable according to noise in the atomic radii. In order to detect these cavities, they investigate the topological changes of the α-shape depending on changes in the atomic radii. Another Delaunay-based approach was used by Maeda and Kinoshita [160] to analyze molecular interfaces.

In 2013, a generalization of the CAST approach that considers the correct atom radii by the β-shape was proposed by Kim et al. [110]. Additionally, the potential molecular paths are extracted by the dual Voronoi diagram of the atom spheres. Note that the path extraction is quite similar to the technique that is presented in the following chapter. The algorithms are available via the web application BetaCavityWeb [113].

A similar approach to CAST is Fpocket [135], which was presented by Guilloux et al. The algorithm computes first the Voronoi diagram of the atom positions and assigns to each Voronoi vertex a maximal α-sphere that does not intersect the atom spheres. In a first step, all α-spheres the radius of which is smaller than a minimal threshold or larger than a maximal threshold are removed (Figure 5.3). Afterwards, the remaining spheres are labeled as apolar or polar depending on the neighboring atoms. Then, a 3-step clustering method is applied to the spheres. In the first step, α-spheres are clustered if they are connected by a Voronoi edge and if their distance is smaller than a threshold. In the second step, clusters are aggregated based on the distance of their centers of mass. Finally, the pairwise distances between α-spheres of clusters are investigated. If a certain number of distances is smaller than a threshold, the two clusters are aggregated. After clustering, small and hydrophobic cavities are removed and the remaining cavities are ranked.

In 2009, Bajaj et al. [9] presented an approach based on topology analysis of the isosurfaces of a discrete distance field of the molecule. For this, they compute the contour tree of the distance field. Then a clustering and classification algorithm is applied to identify cavities and to distinguish between

different types. Since the technique computes the extremal structures of the distance function, automatically potential molecular paths can be provided that correspond to the cavities.

A further approach was presented by Till and Ullmann, called McVol [225]. The approach computes first the SAS of the protein as a discrete set of points with the method proposed by Eisenhaber et al. [60] based on a user-selects probe radius $r_p \in \mathbb{R}$. Internal cavities are detected by connecting neighboring points of the SAS, followed by a connected components search on the resulting graph. Typically, the largest connected component represents the outer part of the SAS, while the other components represent the internal cavities. In addition, a second possibility to extract the internal cavities is proposed. To do so, further points are sampled inside the bounding box of the protein. If a point lies inside the SES it is marked as protein point otherwise it is marked as solvent point. Then, a grid is constructed, where each cell is marked as a solvent cell if at least one sample point in the cell is a solvent point, otherwise the cell is defined as a protein cell. Neighboring solvent cells are connected and again all connected components are detected, which results in the exterior of the protein as well as all internal cavities. Since this method does not detect pockets, the authors proposed a modification to extract them in a separate pass. For each solvent cell, all surrounding cells within a given cube are investigated. If the ratio of protein cells and solvent cells is larger than a user-defined threshold, the cell is marked as a pocket cell. Note that the accuracy of the algorithm depends on the number and quality of the point samplings. Furthermore, the definition of internal cavities and pockets is rather heuristic.

Borland proposed a method that uses ambient occlusion as an indicator for cavities [24]. For this, a triangulated surface of the molecule is required. Then ambient occlusion is pre-computed for the surface. Afterwards, the ambient occlusion values are used to setup the transparency of the surface. High occlusion leads to high opacity; low occlusion leads to high transparency. Thus the user can quickly identify the cavities of the molecular surface. Additionally, the cavities can be colored or extracted by an ambient occlusion threshold.

In 2011, Olechnovič et al. presented Voroprot [172], which is one of the first tools using the Voronoi diagram of the atom spheres instead of the atom positions. They compute the diagram in order to analyze interatomic contact surfaces but also to study cavities. For the latter, they investigate the Voronoi vertices. For each vertex, there exists an empty sphere which is tangent to four atom spheres. Such a sphere corresponds to an internal cavity if it is larger than a given probe sphere and if it is not accessible by the probe sphere from outside the molecule. However, the authors do not give a clear definition nor a visualization concept for molecular cavities.

In 2010, Raunest and Kandt presented one of the first tools, called dxTuber [200], that investigates the internal cavities based on the dynamics of the protein and water inside and around the protein. To achieve this, the protein dynamics are simulated inside a lipid membrane (in case of a membrane protein) and surrounded by water using the Gromacs simulation package. The

positions of the water molecules yield the cavities of the protein. The authors found that short simulations of only 100 ps are sufficient to detect all cavities reachable by water. In order to compute the shape of the cavities, two 3-dimensional discrete grids will be generated that store the number of water and protein atoms per grid cell. The number can either represent the average number of atoms over the simulation time or the minimal number. Similar to LIGSITE [89] and the method by Exner et al. [61], the cavities will be detected and characterized by investigating for each grid cell all cells along the 3 main directions. A grid cell is characterized as internal cavity if it is surrounded along all 3 axes in positive and negative direction by the protein. This means the values in the protein grid are at least as large as a user-defined threshold. If only two of the three directions are surrounded by the protein, the grid cell is characterized as tunnel and in case of only one direction the cell is defined to be inside a pocket. Finally, the grid cells are clustered and the result is filtered to get the description of the cavities. While the algorithm is suitable to detect cavities accessible by water, it cannot detect empty cavities. These cavities are often more flexible and their dynamics are often related to conformational changes in the protein. Furthermore, the algorithm results in a static representation of the cavities, which does not allow to study cavity dynamics. Hence, transport processes due to cavity changes that build, for example, a dynamic channel that is not an always open tunnel cannot be investigated.

One year later, Schmidtke et al. presented MDPocket [211]. The tool uses FPocket [135] to compute the Voronoi diagram of the atom positions for each time step of a molecular dynamics trajectory. Then a grid is created on which a discrete density is computed based on the size of the α-sphere at the Voronoi vertices. By selecting not the complete trajectory but different parts, the specific dynamic processes of the cavities can be analyzed. The cavities are visualized using an isosurface of the discrete density function. However, as for Raunest and Kandt, it is difficult to analyze the detailed dynamic behavior of the cavities.

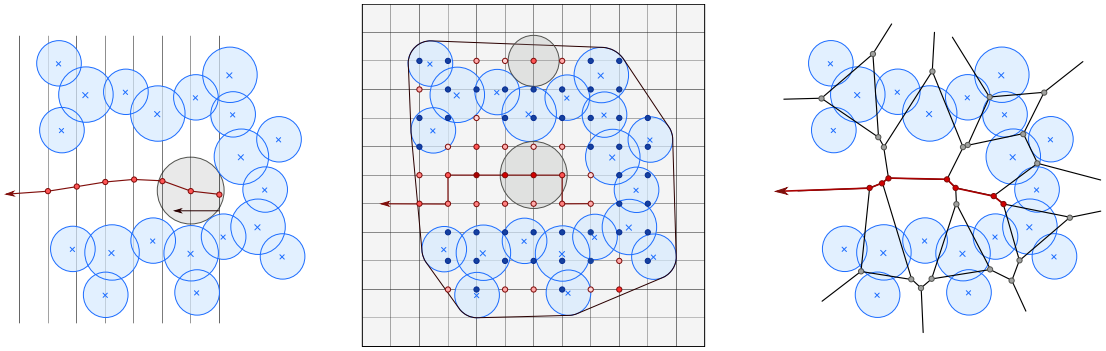
In the same year, Krone et al. [122] presented the first tool to interactively trace selected cavities over time from a molecular dynamics trajectory. They use a density kernel at each atom position, similar to the Gaussian kernel, to create a discrete scalar field of the protein. This allows them to quickly visualize an approximation of the molecular surface by GPU-based isosurface ray casting. Additionaly, the isosurface is used to separate the protein from its cavities. One can select a cavity in an arbitrary time step, which is facilitated by optionally showing a 2-dimensional cut of the surface. Once a cavity is selected a flood fill is performed on the scalar field. The flood fill marks all positions in the field that do not belong to the protein. At each of these positions, a small sphere is rendered to show the shape of the cavity. Afterwards, one can move to the next or previous time step and the cavity will be automatically traced. To do so, the intersection of the marked cavity positions and the new cavity positions of the updated scalar field is computed. This is followed by a new flood fill to get the complete shape of the cavity. Besides

the 3-dimensional tracing and visualization, a plot of the size of the selected cavity is shown. Although, the tool allows one to interactively investigate the dynamics of a cavity, there are still some limitations. The main limitation is, that only a single cavity can be investigated at once and that in case of a split, one part of the cavity will be ignored. Furthermore, the shape and size of the cavity is not restricted to a minimal size given by the shape of the ligand of interest. Thus, it becomes possible to trace the cavity into a region that is geometrically not accessible for the ligand. To achieve the interactivity the fast generation of a discrete density field is used. However, this cannot achieve the geometrical accuracy of, for example, Voronoi-based approaches.

In the more recent approaches, Krone et al. [126, 124] improved their method to overcome the limitation of tracing only a single cavity. First they changed the original density kernel to a modified Gaussian kernel, which results in a quantitative better approximation of the solvent excluded surface. Second, they replaced the ray casting by a GPU-based Marching tetrahedra method to generate a triangular mesh of the molecular surface. And third, the cavities are now defined by an ambient occlusion approach like it was done by Borland [24]. For this, the fast object-space ambient occlusion by Grottel et al. [78] is used. Based on an ambient occlusion threshold, the triangulated molecular surface is separated into parts that belong to the boundary of a cavity and parts that lie outside the molecule. The cavity parts are then labeled according to their connectivity, such that all triangles of the same connected component store the same label. This is used to trace topological events like splits and merges over time but also to visualize the different cavities. The tool is completed by graphs showing, for example, the diameter of a channel along its length or the topological evolution of the cavities over time. While the new approach allows one to keep track of the dynamics of all cavities over time, the cavity definition is purely based on the Gaussian molecular surface and the object-space ambient occlusion as boundary. This makes it difficult to rate the accuracy of the accessibility of the cavities for ligands, even in a purely geometric sense. While ambient occlusion in general provides a good boundary estimation, the result of the fast object-space version [78] depends a lot on the user-selected grid size. Thus, it is difficult to evaluate the quality and quantity of the result.

5.2 Category II: Path Computation

One of the earliest tools to compute a possible molecular path was HOLE by Smart et al. [219]. The tool computes a path from a user-defined start point inside a cavitiy to the outside of the molecule. The path direction is steered by a given direction vector $\vec{v}$ of the cavity. With a Monte Carlo simulating anealing approach the start point is moved to the position where the distance to the atom spheres becomes locally maximal. During this process the point stays in the plane which includes the original start point and is orthogonal



(a) (b) (c)

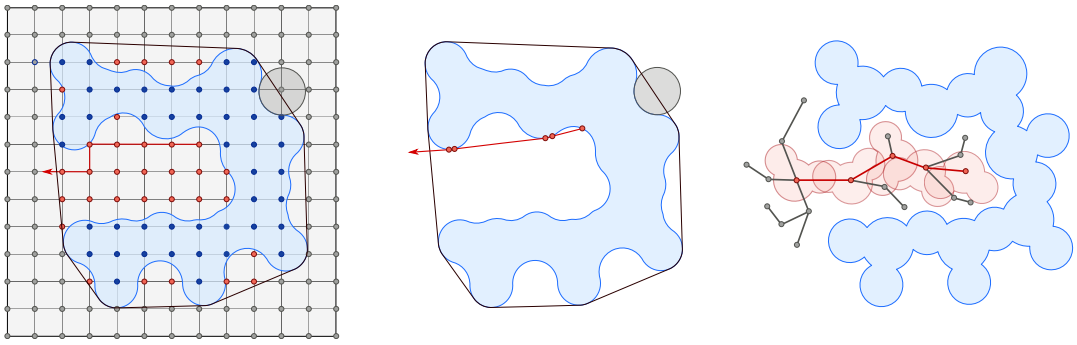
Figure 5.4: *Illustrations of the algorithms HOLE (a), CAVER (b), and MOLE (c). The algorithms detect a path from a user-defined internal position close to a binding site to the outside of the molecule. HOLE computes slices along a given direction and for each slice the point with the local maximal distance. Caver uses a distance-based grid for this purpose and Mole a Voronoi Diagram.*

to $\vec{v}$. Afterwards the point and the plane move a step into the direction of $\vec{v}$ and the simulated anealing approach starts again. This is repeated until the outside of the molecule is reached (Figure 5.4). Note that the approach cannot guarantee to detect the optimal point with the local maximal distance to the atom spheres. Furthermore, the algorithm fails to detect paths in cavities of which the medial axis is more complex, such that it cannot be described by a single direction.

To overcome these drawbacks, Petřek et al. developed a new tool, called CAVER [189], in 2006. CAVER creates a 3-dimensional grid whose grid points and edges are interpreted as a vertex-weighted graph. The weight at each grid point $x \in \mathbb{R}^3$ is given as $1/r(x)^2$, where $r(x)$ is the radius of a sphere with center x and maximal radius such that it does not penetrate the atom spheres. Note that all grid points with $r(x) \leq 0$ are removed from the graph. Afterwards, a modified Dijkstra shortest-path algorithm is applied to find a possible molecular path from a user-defined start point to the convex hull of the atom spheres (Figure 5.4). However, it is neither clear that the vertex-weighting is optimal nor that the shortest path algorithm finds the most probable path.

To reduce the memory requirements and enhance the geometrical accuracy Petřek et al. modified CAVER and called the new version MOLE [188]. The grid is replaced by the Voronoi diagram of the atom positions. The Voronoi vertices and edges now build the graph for the Dijkstra algorithm. They also modified the weighting to consider the path length. Instead of vertex weights, now each edge e is weighted by $l(e)/d(e)^2$, where $l(e)$ is the length of the edge and $d(e)$ is the minimal distance to the atom spheres. Edges whose minimal distance is smaller or equal 0 are removed from the graph (Figure 5.4).

A further extension of this technique was presented by Yaffe et al. in their tool MolAxis [246]. Since the Voronoi diagram of the atom positions does not



(a) (b) (c)

Figure 5.5: *The algorithms by Colmann and Sharp (a), Giard et al. (b), and Cortes et al. (c). The algorithm by Colmann and Sharp samples a molecular surface into a grid and computes for each grid point outside the surface the shortest path to the convex hull. Giard et al. compute the shortest path to the convex hull along the surface and rays between surface points or surface points and the convex hull. The algorithm by Cortes et al. uses rapidly-exploring random trees RRTs to consider the geometry and dynamics of a substrate.*

take into account the different atom radii, the paths computed by MOLE are geometricaly not optimal. To increase the accuracy of the paths in MolAxis, the atoms are approximated by sets of spheres with constant radii. While, for example, a hydrogen atom can be approximated by a single sphere, a carbon atom is approximated by a cluster of several spheres with the same size. Afterwards, again the Voronoi diagram of the centers of the placed spheres is computed. In the next step, all edges that correspond to spheres of the same cluster or that intersect the atom spheres are removed from the graph. The weighting and the path detection is equal to MOLE with the difference that not only a single path is computed but a tree rooted at the user-defined start position.

Parallel to MOLE, Medek et al. [163] developed a similar aproach. Like Yaffe et al., they also considered the different atomic radii in their theoretical investigations. For geometrical optimal paths of probe spheres, this requires the Voronoi diagram of the atom spheres, instead of the atom positions. However, due to the lack of implementations, they also use the classical Voronoi diagram of the atom positions. In contrast to MOLE, the Voronoi diagram is not directly computed, but it is derived from its dual Delaunay complex. Furthermore, a different weighting and Dijkstra modification is used. Each edge is weighted by its minimal distance to the atom spheres and the best path is defined as the path with the edge of maximal minimal weight. They also propose extensions to compute alternative paths.

At the same time when CAVER was developed, Colemann and Sharp presented a similar approach, called travel depth [43]. In this approach, first a triangulated surface of the molecule, like the SAS or SES (Section 2.3.3), is computed together with the corresponding convex hull. Then, a grid is created and for each grid point it is evaluated if the point lies outside or inside

the convex hull. For the latter case it is additionally checked if the point lies inside or outside the molecular surface. After this procedure, the minimal distance of all grid points outside the surface to the convex hull is computed, which approximates the important shortest paths for probe spheres from the convex hull to the reachable cavities (Figure 5.5). Note that paths inside closed cavities are ignored in this approach. To visualize the results, the surface is colored according to the travel depth distance to the convex hull.

In order to improve the time and space complexity, Giard et al. [71] presented more recently an approach which does not require a grid. In contrast, they detect for each point of the triangulated molecular surface the closest point on the convex hull. Then, they use a ray casting approach to evaluate if the surface point is visible from the convex hull or if it is hidden by the molecular surface. For all hidden points, minimal paths to visible points are computed. These paths either follow the triangular mesh or include other direct connections between surface points. For the direct connections it is again checked that the connecting line does not intersect the surface. This results in an approximation of the minimal travel depth for each surface point (Figure 5.5).

In 2009, Pellegrini-Calace et al. [32] presented PoreWalker, which detects the main channel in membrane proteins without user interaction. The method can be summarized in 3 steps. First the orientation of the main channel is aligned parallel to one of the coordinate axes. Therefore, the orientation of the channel is computed by analyzing the directions of long secondary structure elements. Second, the center of the channel in the middle of the protein is detected. This is realized by investigating the properties of the amino acids in the region of the channel. Finally, an iterative procedure slices along the channel direction. In each iteration, a position in the slice close to the computed channel direction is computed whose distance to the atom spheres becomes locally maximal. As with previous algorithms, the main drawbacks are the geometrical accuracy and the limitations for the channel structure, which is expected to be linear.

A technique to compute all channels in a protein, was developed by Coleman and Sharp in their tool CHUNNEL [44]. They compute a triangulation of the SES using a grid-based approach. Afterwards, all topological loops on the surface are detected as triangle strips. These strips characterize the channels in the molecule. In the final step the topological paths through the channels and the corresponding loops are computed such that their distance to the surface becomes maximal. While the approach is one of the first which detects automatically all channels, the algorithm is very slow and geometrically invalid channels can be detected. These invalid channels come from circular singularities of the SES.

In 2013, Brezovsky et al. [28] investigated and compared the tools CAVER, MOLE, and MOLAXIS for general path detection from an active site to the outside as well as HOLE, MOLAXIS, CHUNNEL, and POREWALKER for the

specialization of channel detections. Furthermore, they compared HOLLOW and 3V in case of general cavity detection.

In the same year, Sehnal et al. presented MOLE 2.0 [213], which comes with several improvements. The tool approximates automatically possible cavity centers based on the Voronoi diagram of the atom positions. Then, instead of only one path, several paths are computed from these centers to the outside of the molecule in a similar way as in the original version. Subsequently, the paths are filtered according to their bottleneck. Furthermore, if the centerlines of two paths are quite similar, the longer path is removed. Then, the main feature of the new method is applied. It computes along each remaining path physiochemical properties, like charge, hydropathy, hydrophobicity, mutability, and polarity based on the surrounding side chains of the amino acids. These properties can be used to filter the most probable paths.

One of the rare methods that take into account the geometry and dynamics of the substrate was developed by Cortes et al. [48]. They use rapidly-exploring random trees (RRTs) [134] to compute a possible molecular path to a binding site. RRTs were originally developed for fast path planning in robotics. A tree is incrementelly constructed by adding random valid roboter configurations as tree nodes until a node reaches a point or area of interest. For molecular path detection, the substrate is considered as the roboter and the protein is the labyrinth for which a path should be detected from a user-defined start position to the outside of the protein. The start position is the root of the tree and a valid configuration is a position and orientation of the substrate such that it does not penetrate the protein and that can be reached by the closest node in the current tree. The second condition means that the substrate must be close enough to an existing node in the tree such that it is guaranteed to move the substrate from the tree node to the new configuration without penetrating the protein (Figure 5.5). Depending on the number of free variables for the configuration of the substrate, the algorithm can be very slow. Furthermore, it is difficult to setup a stop criterion for the tree construction. In 2011, Cortes et al. [47] extended their previous approach to better visualize and analyze the results of the RRT. To do so, they generate a 3-dimensional voxel map, on which the RRT is mapped. Note that the voxel map is not restricted to the 3-dimensional position of the ligand. The three variables of the map can encode any user-selected ligand property of interest, such as 3 selected bond torsions. The algorithm for the mapping is straightforward and classical visualization techniques are applied.

Sethian and Haranczyk [84] also take into account the geometry of the substrate by proposing a discrete approach for a 7-dimensional space which includes translational, rotational, and internal degrees of freedom of the substrate. For each sample in this space, it is checked if the substrate is in a valid state according to the receptor molecule. After sampling the space, the shortest path for the substrate is computed based on the valid samples. However, due to the sampling of the 7-dimensional space, the approach is time-consuming and requires a lot of memory.

A method to analyze the cavities in molecular dynamic trajectories was presented by Parulek et al. [178, 179]. They compute the cavities for a certain time step by randomly sampling points inside and around the protein. All points inside the SES will be removed afterwards. Then, at each remaining point a ray is cast along the gradient of the distance field of the SES. If the ray does not hit the SES, the corresponding sample point is also removed. In the next step all remaining sample points are moved to the center of the line segment that connects the previously computed hit point of the ray with the SES and the hit point of the inverse ray with the SES. Finally, a modified minimal spanning tree method is applied to connect the sample points and to generate a skeleton graph representation for the cavities. Instead of tracing cavities over time, Parulek et al. propose brushing and linking to analyze the cavitiy structure. They compute for each skeleton graph, several attributes based on the degree of vertices or the maximal and average paths in the graph. Then, based on selected attributes, each graph is rendered as a point into a 2-dimensional scatter plot. By selecting points in the scatter plot, the 3-dimensional visualization is updated and shows the corresponding skeleton graphs. This allows one to detect and visualize, for example, cavities with similar properties. However, the coherency of cavities from time step to time step and their topological changes cannot be analyzed with this technique. Furthermore, the geometric accuracy of the definition of the cavity skeletons has not the quality of a Voronoi diagram of the atom spheres.

Recently, the tool CAVER 3.0 [42] was presented, which uses the same strategy as MolAxis [246] to approximate the different atom radii. Each atom is represented by a set of spheres with constant radius. Based on the Voronoi diagram of the centers of the spheres, paths are detected from a user-selected start point to the outside of the protein using again a modified Dijkstra algorithm. However, the main new feature of CAVER 3.0 is that it does not compute the paths for a single snapshot but for a molecular dynamics trajectory. Then a geometric heuristic is applied to cluster paths that correlate over time and to cluster quite similar paths in the same snapshot. Finally the paths, clusters and the corresponding cavities are visualized. The most critical part of the algorithm is the heuristic for the clustering of paths.

In 2016, Kim et al. [107] proposed a GPU-accelerated algorithm that extracts cavities using a grid-based Voronoi diagram of spheres. Their method first computes a voxelized approximate convex hull. Next, each convex hull voxel that is not within an atom is classified whether it belongs to a Voronoi diagram edge. This results in a discretized grid representation of the edges of the Voronoi diagram, which are then clustered and subsequently used to find paths.

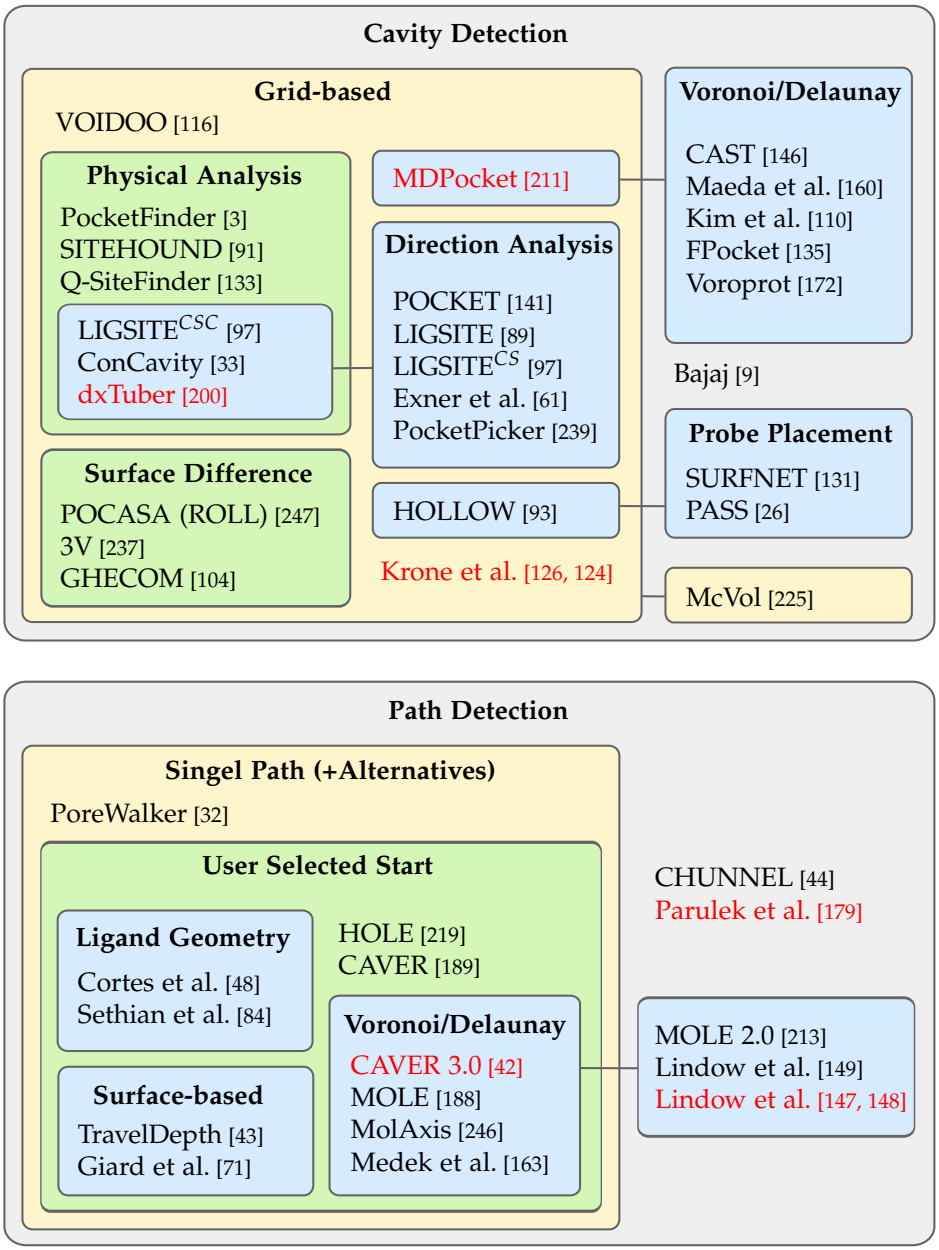


Figure 5.6: *A grouping of algorithms and tools to compute molecular paths and cavities. The tools highlighted by a red color deal with molecular dynamics trajectories.*

5.3 Summary

In the previous sections a lot of algorithms and tools have been outlined to compute molecular paths and cavities. To get a better overview, in this section a proposal for a grouping of the different techniques is proposed. This grouping is shown in Figure 5.6. As already mentioned in the previous section, two main groups of cavity computation algorithms and path detection methods can be distinguished.

Most of the cavity computation algorithms are based on a 3-dimensional grid to discretize the space around the molecule. Note that the time and memory complexity of these grid-based methods grows cubically with decreasing lattice spacing. However, on the other hand these algorithms are often easier to implement and provide numerical stable solutions. Most of the grid-based algorithms analyze at each grid point if a molecular surface representation can be reached in a fixed number of directions. There are several algorithms that compute simple physical properties at the grid points to detect valid cavities. In addition, some of these techniques also use a geometry-based direction analysis. Apart from the grid-based algorithms, there are several approaches based on Voronoi diagrams or Delaunay complexes. Usually, these methods have a higher geometrical precision but their implementation is often more difficult.

In the field of path detection methods, there is a large group of algorithms that compute only a single path, sometimes with alternative solutions. Moreover, most of these algorithms require a user-selected start position. Again, there is a group of algorithms based on Voronoi diagrams or Delaunay complexes. Since these structures provide a full path network with high geometrical accuracy, recently a few works, including the approach presented in this thesis, exploit this for a complete path analysis without user-selected start points. Other approaches that deal with the full path network computation either analyze the topology of a molecular surface or use geometric heuristics to compute in the end a graph with similar properties as a Voronoi diagram.

6 Cavity Computation

In this chapter, the method presented in "*Voronoi-Based Extraction and Visualization of Molecular Paths*" [149] from 2011 is described. It enables the user to compute the whole cavity structure inside a molecule as well as the corresponding significant possible molecular paths. A formal definition of molecular paths and cavities was given in Section 2.4. To achieve a good tradeoff between computation time and accuracy, the algorithms are solely based on the geometry of the atom spheres of the receptor molecule and the geometry of a probe sphere to approximate a ligand, solvent, or ion. This means, a molecular path is a continuous curve in $\mathbb{R}^3$ whose distance to any atom sphere is greater than the probe radius. Furthermore, a molecular cavity is the union of all points inside all probe spheres that are interconnected by static molecular paths. The boundary will be defined by an ambient-occlusion approach.

Since the computation of all molecular paths is not possible, only a finite subset is computed, which in this thesis is called the set of *maximal molecular paths*. These paths are defined as the ones with the local maximal distance to the atom spheres. With this definition, it is guaranteed that for each possible path inside a cavity, there exists at least one representative maximal path. Note that the edges of a Voronoi diagram have a local maximal distance to the corresponding input geometry (Section 2.1.3). Thus the maximal molecular paths are a subset of the vertices and edges of the Voronoi diagram of the atom spheres. In other words, the network of Voronoi vertices and edges contains all geometrical optimal paths for spheres. The approach presented in this chapter is one of the first that used the Voronoi diagram of spheres for the computation of possible molecular paths. Except for some rare and com-

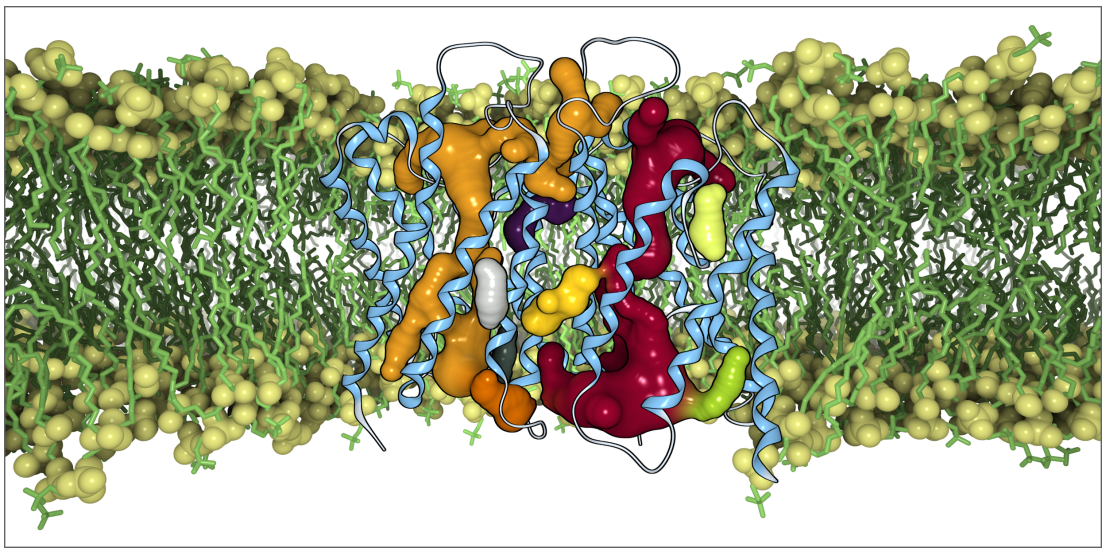


Figure 6.1: *Two of four chains of the Water Channel protein pdb: 2F2B (blue) inside a lipid bi-layer membrane (green). The internal cavities are shown as colored surfaces in which two main channels can be identified.*

putational expensive techniques that take into account the ligand geometry, such as the approaches by Sethian and Haranczyk [84] or by Cortes et al. [47], this method has the highest geometrical accuracy in the field of geometry-based molecular path computation for macromolecules. Furthermore, the approach is able to compute the full path network and the corresponding cavities, instead of a single path. Nevertheless, the speed of the algorithm can still compete with most other geometry-based molecular path computation algorithms. A preview of the resulting cavities in a water channel protein can be seen in Figure 6.1.

6.1 Voronoi Diagram of Spheres

The Voronoi diagram of spheres is an extension of the classical Voronoi diagram. While its computation is more complex than that of the Voronoi diagram of points, the separating faces of the decomposition can still be computed analytically. In contrast to the Voronoi diagram of points, only a few publications deal with the computation of the Voronoi diagram of spheres. Aurenhammer [7] outlined a method where the Voronoi diagram can be obtained from the power diagram of the transformed spheres in one dimension higher. A lower envelope algorithm for the construction of the Voronoi regions was presented by Will [241]. Boissonnat and Delage [21] proposed a method to compute the regions of the diagram by convex hulls of spheres. Kim et al. [108] presented an approach that starts with the Voronoi diagram of points using the sphere centers, that is, with all radii set to zero; subsequently the radii are incrementally increased sphere by sphere. However,

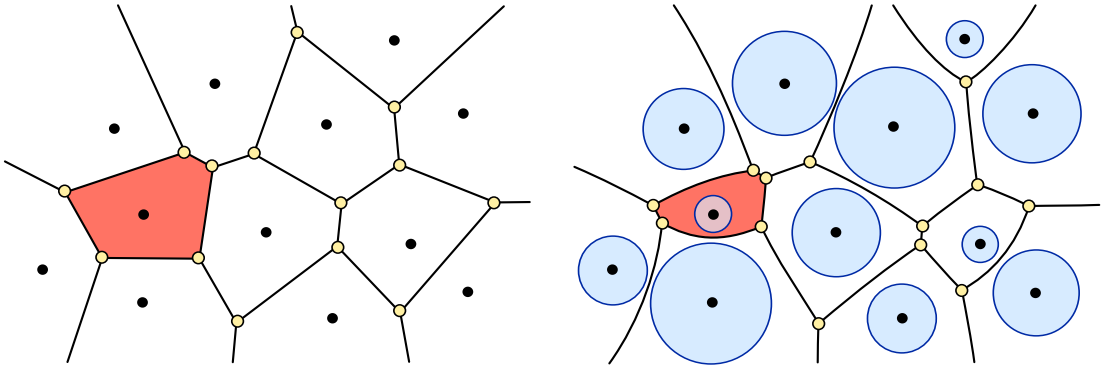


Figure 6.2: *Classical Voronoi diagram of points (left) compared to the Voronoi diagram of sphere (right) for the 2-dimensional case. The Voronoi vertices are shown in yellow, the edges in black and one closed Voronoi cell is highlighted in red in both diagrams.*

the implementation of the algorithms mentioned above is quite complicated. Hence, the simple yet efficient edge-tracing algorithms are more often used in practice [164, 161, 149].

The idea of tracing edges for computing Voronoi diagrams was already proposed by Luchnikov et al. [155] and later described for spheres in detail by Kim et al. [109] and Medvedev et al. [164]. One important step in both algorithms is to determine the tracing direction when computing the end vertex of an edge from a start vertex. While this step is missing in the description of Kim et al. [109], Medvedev et al. [164] use a procedure that works only for a specific kind of input data. Since an incorrect computation of this direction can lead to errors in the Voronoi diagram, this is an essential step of the algorithm. In this section, a criterion is derived that allows the correct computation of the edge tracing direction for the general case.

In addition, the complete computation of the 3D Voronoi diagram of spheres is described. Instead of a set of input points, as in the Voronoi diagram of points, input spheres define the Voronoi regions. A 2-dimensional illustration of both types is shown in Figure 6.2 and an example of a 3-dimensional diagram of spheres is visualized in Figure 6.3. Each Voronoi region is defined by the set of points whose distance to the corresponding input sphere is smaller than or equal to any other input sphere.

> **Voronoi Region of a Sphere.** *Let $S \subset \mathbb{R}^3 \times \mathbb{R}$ be a finite set of m spheres (p_i, r_i), with $i = 1, ..., m$. The Voronoi Region V_i corresponding to the ith sphere is*
>
> $$V_i = \left\{ p \in \mathbb{R}^3 \;\middle|\; \|p - p_i\| - r_i \leq \|p - p_j\| - r_j,\ \forall j \in \{1, \ldots, m\}, j \neq i \right\}.$$

The Voronoi diagram of spheres is the set of all nonempty intersections of Voronoi regions of spheres.

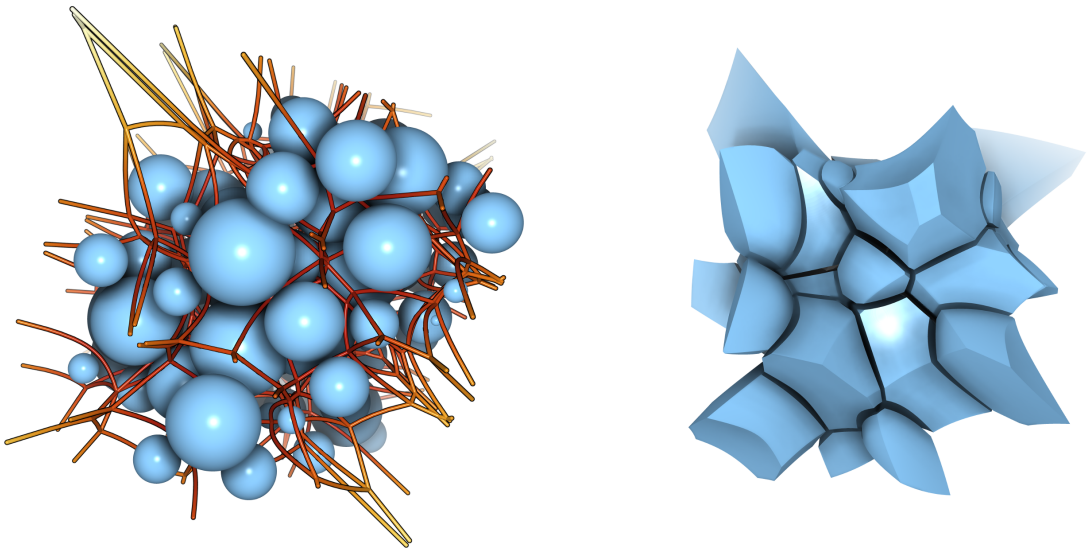


Figure 6.3: *Voronoi diagram of a set of 3-dimensional spheres. Left: Voronoi edges colored according to their distance to the input spheres (blue). Right: All closed Voronoi cells.*

Voronoi Diagram of Spheres. *Let $S \subset \mathbb{R}^3 \times \mathbb{R}$ be a finite set of m spheres (p_i, r_i), with $i = 1, ..., m$ and let V_i be the corresponding Voronoi regions. The Voronoi diagram V_S of S is*

$$V_S = \left\{ V_X \;\middle|\; V_X \neq \emptyset, V_X = \bigcap_{i \in X} V_i, X \subseteq \{1, \ldots, m\} \right\}.$$

Similar to the Voronoi diagram of points, the 3-dimensional Voronoi diagram of spheres consists of four element types called regions, faces, edges and vertices. These elements are generated by the intersection of Voronoi regions. For this reason, the corresponding input spheres are called *generator spheres* or simply generators of the respective Voronoi element. In the non-degenerate case, each face is the intersection of exactly two regions, each edge of exactly three regions, and each vertex of exactly four regions. Note that the algorithm presented here assumes that no degenerate case occurs. As for the classical Voronoi diagram, this assumption can be satisfied by either perturbing the input data by a small amount that is irrelevant for the corresponding application or by 'simulation of simplicity', as described by Edelsbrunner and Mücke [58].

For the following assume a set of n spheres $S \subset \mathbb{R}^3 \times \mathbb{R}$ like the atom spheres of a molecule with positions $p_i \in \mathbb{R}^3$ and radii $r_i \in \mathbb{R}$, with $i \in I = \{1, \ldots, n\}$.

6.1.1 Voronoi Elements

Before the description of the computation of the 3-dimensional Voronoi diagram of spheres is given, in this section, first the properties, analytical repre-

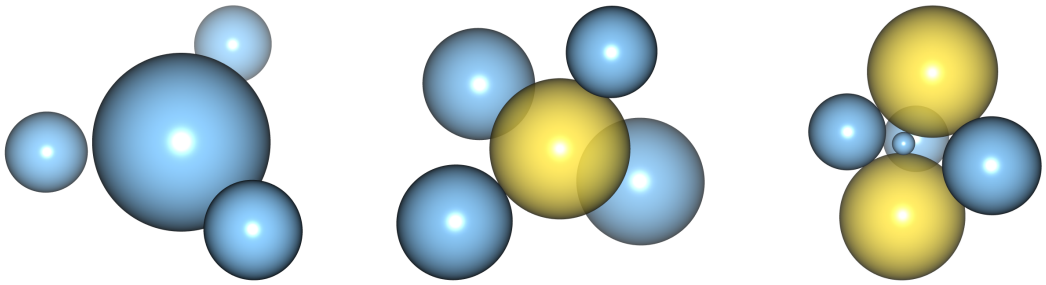


Figure 6.4: *Four input spheres (blue) can generate no Voronoi vertex (left), one vertex (middle) or two vertices (right). The Voronoi vertices are represented by the yellow spheres.*

sentations and derivations of all Voronoi elements are described. In contrast to the Voronoi diagram of points, there is no one-to-one correspondence between an element of V_S and a single face, a single edge or a single vertex. For example, in the Voronoi diagram of spheres, it is possible that two Voronoi vertices are generated by the same four generator spheres (Figure 6.4, right).

Vertices

For non-degenerated Voronoi diagrams, each Voronoi vertex is given by the intersection of exactly 4 Voronoi regions. Consider a Voronoi vertex v, which is an element of the intersection of four distinct regions V_{i_k}, with $i_k \in I$, $k = 1, ..., 4$. This means that v has the same distance to the four generator spheres (p_{i_k}, r_{i_k}). Another interpretation is that there exists a sphere $s = (p_v, r_v)$, with $p_v = v$ that is tangent to the four generator spheres. Note that except for the four generator spheres, no other input sphere exists that penetrates or touches sphere s. For this reason, s is called empty.

Given the generator spheres of a Voronoi vertex, its position can be determined by computing the spheres tangent to the generator spheres. An elegant algorithm to compute the tangent spheres for a set of spheres in an arbitrary dimensional space was presented by Gavriola et al. [70] and is summarized here for the 3-dimensional case. The solution to this problem can be formulated by a system of equations. In each of these equations, the distance between the position of the tangent sphere and the position of a generator sphere is equal to the sum of their radii.

$$\left\| p_v - p_{i_k} \right\| = r_v + r_{i_k}, \quad k = 1, ..., 4. \tag{6.1}$$

Because the left side of the equation is always positive, r_v must be greater than or equal to $-r_{i_k}$. Under this condition both sides can be squared which yields

$$(p_v - p_{i_k})^2 = (r_v + r_{i_k})^2, \quad k = 1, ..., 4. \tag{6.2}$$

One can see that it is possible to add a constant value $r_c \in \mathbb{R}$ to the radii of the generator spheres without changing the position of the tangent sphere.

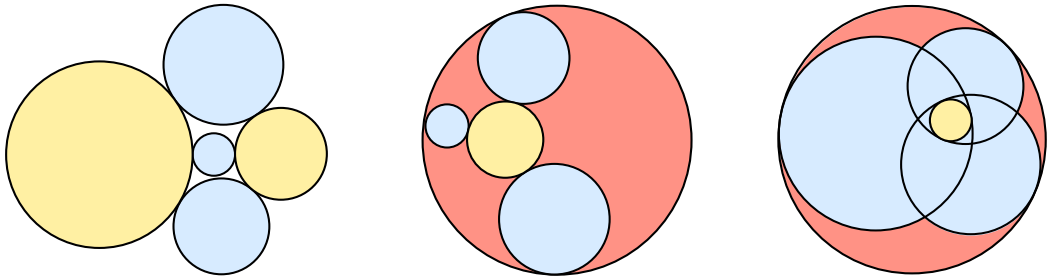


Figure 6.5: *Possible solutions of tangent spheres for input spheres (blue) from the system given by* (6.3) *and* (6.4) *transformed back and illustrated in 2D. Valid vertex spheres are dipicted in yellow and invalid spheres in red. Both radii of the tangent spheres are positive (left), negative (right) or one is positive and the other negative (middle).*

But the same value is subtracted from its radius. Furthermore it is possible to move all spheres by a constant vector $p_c \in \mathbb{R}^3$ without changing the radii or their relative positions to each other. So the problem is translation invariant

$$((p_v - p_c) - (p_{i_k} - p_c))^2 = (r_v - r_c + r_{i_k} + r_c)^2, \quad k = 1, ..., 4.$$

The main trick to solve the system of equations (6.2) is to shrink one of the input spheres to a point and move it to the origin of the coordinate system. Without loss of generality, the sphere with index i_1 is selected. So the transformed positions of the input spheres are $\tilde{p}_{i_k} = p_{i_k} - p_{i_1}$, and their radii are $\tilde{r}_{i_k} = r_{i_k} - r_{i_1}$. Hence, the new system of equations is

$$\tilde{p}_v^2 = \tilde{r}_v^2 \tag{6.3}$$

$$(\tilde{p}_v - \tilde{p}_{i_k})^2 = (\tilde{r}_v + \tilde{r}_{i_k})^2, \quad k = 2, ..., 4. \tag{6.4}$$

By subtraction of the first equation from the others, the system is decomposed into a linear part, consisting of three equations, and a quadratic part given by the first equation. The linear part results in the following under-determined system of equations.

$$2 \begin{bmatrix} \tilde{p}_{i_2}^T & \tilde{r}_{i_2} \\ \tilde{p}_{i_3}^T & \tilde{r}_{i_3} \\ \tilde{p}_{i_4}^T & \tilde{r}_{i_4} \end{bmatrix} \begin{bmatrix} \tilde{p}_v \\ \tilde{r}_v \end{bmatrix} = \begin{bmatrix} \tilde{w}_{i_2} \\ \tilde{w}_{i_3} \\ \tilde{w}_{i_4} \end{bmatrix} \quad \text{with} \quad \tilde{w}_{i_k} = \tilde{p}_{i_k}^2 - \tilde{r}_{i_k}^2 \quad , \quad k = 2, ..., 4. \tag{6.5}$$

The system can be solved, for example, using Gaussian elimination with pivoting. Since the system is under-determined, one has to choose a free variable, for example $\tilde{r}_v$. The linear solution of (6.5) can then be inserted into (6.3). This gives two, one or no solution for $\tilde{r}_v$. For the typical case of two solutions, two tangent spheres transformed back into the original space have to be considered. Depending on the signs of the radii r_{v_1} and r_{v_2}, there are four possible solution combinations, illustrated in Figure 6.5. However, these tangent spheres do not necessarily represent valid Voronoi vertices. Therefore one

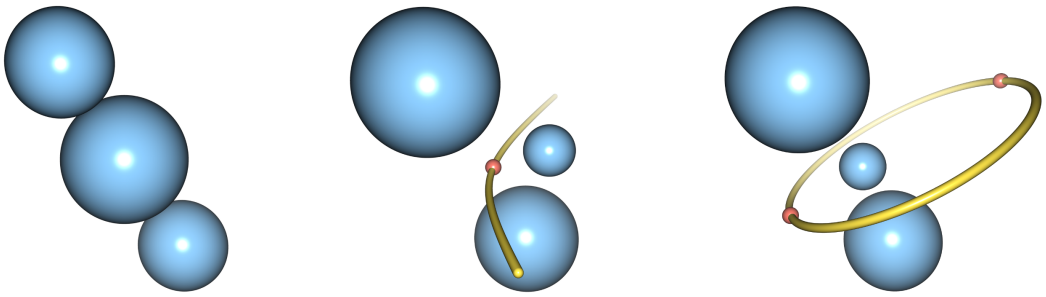


Figure 6.6: *Three input spheres (blue) can create no Voronoi edge curve (left), a non-closed edge curve (middle), or a closed edge curve (right). The points on the edge curves with the minimal and maximal distances to the input spheres are illustrated by the red spheres.*

has to check if the spheres violate the conditions $r_{v_{1/2}} \geq -r_{i_k}$, which means that the four generator spheres lie completely inside the tangent sphere with radius $\left|r_{v_{1/2}}\right|$. In summary, four input spheres can generate zero, one or two Voronoi vertices (Figure 6.4).

Edges

A Voronoi edge e is given by the intersection of exactly 3 different Voronoi regions V_{i_k}, $i_k \in I$, $k = 1, ..., 3$, that is $e \subset V_{\{i_1,i_2,i_3\}} \in V_S$. For each point on the edge, the distance to the three generator spheres (p_{i_k}, r_{i_k}) is equal, and the distance to any other input sphere is at least as large as the distance to the generator spheres. Similarly to a Voronoi vertex, each point on the edge corresponds to an empty sphere that is tangent to the three generator spheres.

Consider first the complete curve, which includes the edge. A point $p_e \in \mathbb{R}^3$ on this curve has the same distance $r_e \in \mathbb{R}$ to the three generator spheres. Hence, the curve is given by the solution of a system of equations, equivalent to the system for the computation of a Voronoi vertex (6.1). But it consists only of three equations with four variables. After applying the same transformation, it results into two linear equations (6.4) and one quadratic equation (6.3). A rearrangement of these equations leads to

$$\begin{aligned} \tilde{p}_e^2 &= \tilde{r}_e^2 \\ \langle \tilde{p}_e, \tilde{p}_{i_2} \rangle + 1/2(\tilde{r}_{i_2}^2 - \tilde{p}_{i_2}^2) + \tilde{r}_e \tilde{r}_{i_2} &= 0 \\ \langle \tilde{p}_e, \tilde{p}_{i_3} \rangle + 1/2(\tilde{r}_{i_3}^2 - \tilde{p}_{i_3}^2) + \tilde{r}_e \tilde{r}_{i_3} &= 0. \end{aligned}$$

The first equation describes a sphere that lies in the origin of the coordinate system and has the radius $\tilde{r}_e$. The second equation describes a plane with normal $\tilde{p}_{i_2}$ and a distance to the origin that depends linearly on $\tilde{r}_e$. The same is true for the third equation with normal $\tilde{p}_{i_3}$. The solution of this system of equations is the intersection of these three geometric representations. The intersection of the two planes can be empty, a line, or a plane.

1. If $\tilde{p}_{i_2}$ and $\tilde{p}_{i_3}$ are linearly independent, the intersection of the two planes is a line. Since the distance of the planes to the origin change linearly

with $\tilde{r}_e$, all resulting lines also lie in a plane. This means that the whole edge curve lies in a plane. The intersections of the lines and the sphere depending on $\tilde{r}_e$ is equivalent to the intersection of the plane, containing the lines, and a cone. Thus, the edge curve can be described as a conic section. Note that the intersection points of a line with a sphere are symmetric with respect to the plane that contains the center of the sphere and that is orthogonal to the line. For all $\tilde{r}_e$, this is exactly the plane containing $\tilde{p}_{i_2}$, $\tilde{p}_{i_3}$, and the origin. Hence, the edge curve is symmetric to the plane that contains the positions of the 3 generator spheres. For $r_{i_1} = r_{i_2} = r_{i_3}$, the planes have a constant distance to the origin. Thus, the edge curve becomes a line.

2. For the special case that $\tilde{p}_{i_2}$ and $\tilde{p}_{i_3}$ are linearly dependent, their intersection is empty or a plane. In this case, the 3 generator spheres lie on a line. Since $\tilde{p}_{i_2} \neq \tilde{p}_{i_3}$, there exists only one $\tilde{r}_e$ for which the intersection can be a plane. The intersection of this plane with the sphere can be empty, a point, or a circle. If the intersection is a point the edge and thus the whole Voronoi diagram is degenerate. For the case, that the intersection is a circle, the edge curve is also a circle, the center of which lies on the line through the position of the input spheres.

To summarize, the edge curve lies in a plane and is symmetric to the plane spanned by the positions of the three generator spheres. Additionally, the edge curve is a conic section and can be closed or non-closed. While a non-closed curve intersects the symmetry plane once, a closed curve has two intersection points. These intersection points correspond to the tangent spheres with minimal and maximal radius. Note that a non-closed curve has only a minimum. The computation of the intersection points with the symmetry plane is equivalent to the computation of the Voronoi vertices in one dimension lower. The 2-dimensional Voronoi vertices correspond to the circles tangent to the three circles given by the intersection of the symmetry plane with the generator spheres. For a solution with two vertices, the edge curve is closed, otherwise it is non-closed. The Voronoi edge of a closed curve can be unbounded or bounded by two Voronoi vertices; a Voronoi edge of a non-closed curve can be unbounded, bounded by one Voronoi vertex or bounded by two Voronoi vertices.

For analysis and visualization purposes, it is often necessary to compute a parametric description of the edges. As mentioned before, an edge curve can be described by a conic section. Hence a fully bounded edge can be parameterized by a rational quadratice Bézier curve [64]

$$e_B(t) = \frac{(1-t)^2 v_s + 2w(1-t)t v_c + t^2 v_e}{(1-t)^2 + 2w(1-t)t + t^2}, \quad t \in [0,1] \subset \mathbb{R}, \tag{6.6}$$

where $v_s \in \mathbb{R}^3$ ist the start vertex and $v_e \in \mathbb{R}^3$ is the end vertex. Together with $v_c \in \mathbb{R}^3$, they create the control triangle of the curve. Note that v_c is the

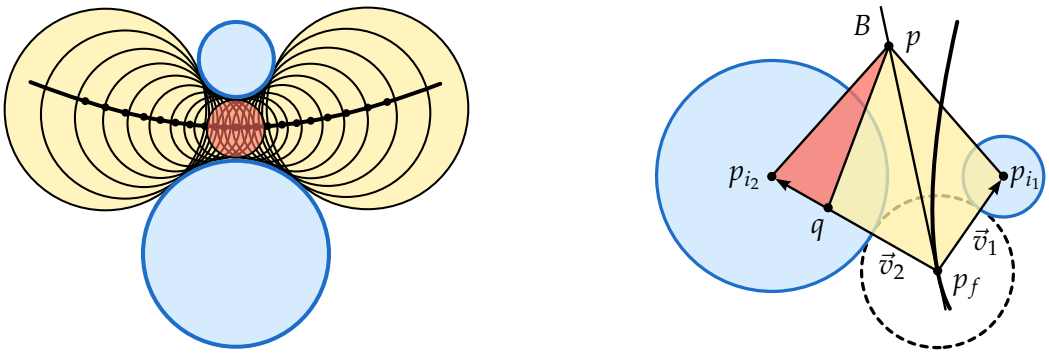


Figure 6.7: *Left: Illustration of the empty tangent spheres on the edge curve. The sphere with minimal distance is depicted in red. Right: Geometrical proof that a tangent plane in a point p_f on a Voronoi face is an angle bisector of the corresponding vectors $\vec{v}_1$ and $\vec{v}_2$.*

intersection point of the tangents through v_s and v_e. The weight $w \in \mathbb{R}$ can be computed by a third point on the edge curve, for example, the point with the minimal distance to the three generator spheres.

In the following, the proof by Kim et al. [109] is summarized, which shows that the direction $\vec{t}_e$ of a tangent in a point p_e on the edge curve is equi-angular to the three vectors $p_{i_k} - p_e$, $k = 1, ..., 3$. Thus, it can be computed by solving the system of linear equations

$$\left\langle \frac{p_{i_k} - p_e}{\|p_{i_k} - p_e\|}, \vec{t}_e \right\rangle = c, \quad k = 1, ..., 3, \tag{6.7}$$

where $c \in \mathbb{R}$ is a constant that is set to a value unequal to 0. For the special case where p_e lies in the symmetry plane, all vectors $p_{i_k} - p_e$ lie in this plane. So the equi-angular vector is normal to this plane. Consider an arbitrary point $p_f \in V_{\{i_1,i_2\}}$, which means p_f lies on the face given by the intersection of the Voronoi regions V_{i_1} and V_{i_2}. Let $\vec{v}_1 = p_{i_1} - p_f$ and $\vec{v}_2 = p_{i_2} - p_f$ and without loss of generality, assume $|\vec{v}_1| \leq |\vec{v}_2|$. Furthermore, let B be the angle bisector plane of $\vec{v}_1$ and $\vec{v}_2$. For an arbitrary point $p \in B$, with $p \neq p_f$, one can construct a kite with the vertices p_f, p_{i_1}, p, and a point $q \in \mathbb{R}^3$ on $\vec{v}_2$, (Figure 6.7). Next to the kite, a triangle remains with vertices p, q, and p_{i_2}. This triangle leads to the inequality

$$\begin{aligned} &\|p - p_{i_2}\| \leq (r_{i_2} - r_{i_1}) + \|p - p_{i_1}\| \quad \Longleftrightarrow \\ &\|p - p_{i_2}\| - r_{i_2} \leq \|p - p_{i_1}\| - r_{i_1}. \end{aligned}$$

This means that the distance of any point p of B to the sphere (p_{i_2}, r_{i_2}) is always less than or equal to the distance of p to the sphere (p_{i_1}, r_{i_1}). Thus, B is the tangent plane in p_f, because it never penetrates the Voronoi face. Consider now an arbitrary point p_e on a Voronoi edge. The edge is part of three Voronoi faces, and the tangent in p_e lies inside the tangent planes of the three faces and thus it has the same angle to the three vectors $p_{i_k} - p_e$, $k = 1, ..., 3$.

The last missing parameter to describe the edge is the weight w in (6.6). It was shown [64], that the weight can be computed by

$$w = \frac{\lambda_2}{2\sqrt{\lambda_1 \lambda_3}},$$

where $\lambda_i, i = 1, ..., 3$, are the barycentric coordinates of a point on the edge curve according to the triangle $\Delta v_s v_c v_e$. Note that the sign of w has to be reversed if the point on the edge curve with the minimal distance to the 3 input spheres is outside the bounds of the edge.

Faces

A Voronoi face f is part of the intersection of two distinct Voronoi regions V_i and V_j, thus $f \subset V_{\{i,j\}} \in V_S$. It was shown by Goede et al. [73] that a face is part of a hyperbolic surface $H_{\{i,j\}}$. To show this, Goede et al. used a rigid transformation such that p_i lies in the origin and p_j on the x-axis. Without loss of generality, assume $r_i \leq r_j$. For each point $p_f \in \mathbb{R}^3$ on $H_{\{i,j\}}$

$$\|p_f - p_i\| - r_i = \|p_f - p_j\| - r_j.$$

After applying the transformation, a point $\tilde{p}_f$ on $\tilde{H}_{\{i,j\}}$ is given by

$$\|\tilde{p}_f\| - r_i = \left\| \tilde{p}_f - \begin{bmatrix} d_{ij} \\ 0 \\ 0 \end{bmatrix} \right\| - r_j \qquad \Leftrightarrow$$

$$\sqrt{\tilde{p}_{f_1}^2 + \tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2} - r_i = \sqrt{\left(\tilde{p}_{f_1} - d_{ij}\right)^2 + \tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2} - r_j \qquad \Leftrightarrow$$

$$\sqrt{\tilde{p}_{f_1}^2 + \tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2} - r_i + r_j = \sqrt{\left(\tilde{p}_{f_1} - d_{ij}\right)^2 + \tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2},$$

where $d_{ij} = \|p_i - p_j\|$. Squaring this equation leads to

$$\tilde{p}_f^2 + r_{ij}^2 - 2r_{ij}\sqrt{\tilde{p}_{f_1}^2 + \tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2} = d_{ij}^2 - 2d_{ij}\tilde{p}_{f_1}^2 + \tilde{p}_f^2 \qquad \Leftrightarrow$$

$$r_{ij}^2 + 2d_{ij}\tilde{p}_{f_1}^2 - d_{ij}^2 = 2r_{ij}\sqrt{\tilde{p}_{f_1}^2 + \tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2} \geq 0,$$

where $r_{ij} = r_j - r_i$. Note that d_{ij} is always larger than r_{ij}. Otherwise one sphere would lie completely inside the other, and there would not exist a Voronoi face. Under this condition, the equation can be squared again and divided by $4(d_{ij}^2 - r_{ij}^2)$, which leads to

$$\left(\tilde{p}_{f_1} - \frac{d_{ij}}{2}\right)^2 - \frac{r_{ij}^2}{d_{ij}^2 - r_{ij}^2}\left(\tilde{p}_{f_2}^2 + \tilde{p}_{f_3}^2\right) = \frac{r_{ij}^2}{4}, \tag{6.8}$$

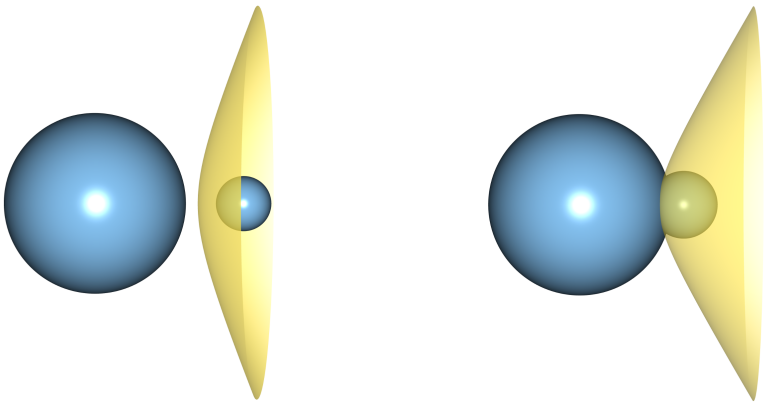


Figure 6.8: *Two examples for a Voronoi face between two input spheres. A Voronoi face is part of one sheet of a hyperboloid of two sheets.*

with $\tilde{p}_{f_1} \geq d_{ij}/2 - r_{ij}^2/2d_{ij}$. Equation (6.8) describes one sheet of a circular hyperboloid of two sheets whose rotational axis is equal to the x-axis (Figure 6.8). In general such a hyperboloid is given by the equation

$$a\left(\tilde{p}_{f_1} - m_1\right)^2 - b\left(\tilde{p}_{f_2} - m_2\right)^2 - c\left(\tilde{p}_{f_3} - m_3\right)^2 = w,$$

where $m \in \mathbb{R}^3$ is the center and $a, b, c, w \in \mathbb{R}$ are nonnegativ parameters describing the shape. Additionally, the circular property satisfies the condition $b = c$. For the hyperboloid in (6.8) this means

$$m = \begin{bmatrix} d_{ij}/2 \\ 0 \\ 0 \end{bmatrix}, \quad a = 1, \quad b = c = \frac{r_{ij}^2}{d_{ij}^2 - r_{ij}^2}, \quad w = \frac{r_{ij}^2}{4}.$$

Voronoi faces are bounded by Voronoi edges. Since a Voronoi edge lies in a plane, a face can be completely described as the intersection of the half-spaces, given by the planes of the Voronoi edges, and the hyperboloid. A similar description of Voronoi faces and their discretization into triangular meshes was given by Kim et al. [109].

6.1.2 Algorithm

In this section, a complete algorithm for the computation of the Voronoi diagram of spheres is presented. This algorithm is a combination and extension of the algorithms presented by Kim et al. [109] and Medvedev et al. [164]. Similarly to these algorithms, the Voronoi diagram is not directly computed, but a graph representing the Voronoi vertices and edges. The analytical description of all Voronoi elements can be quickly and easily computed from this graph. for this reason the graph is called *Voronoi graph*.

Overview

Before the algorithm is described in detail, first an overview is given. The Voronoi graph is constructed in an incremental manner from a single start vertex. Recall that each vertex is defined by exactly four and each edge by exactly three generator spheres. Let $i, j, k, l \in I$ be the indices of the four spheres generating the start vertex. Each combination of three of these generator spheres creates one Voronoi edge. The four possible combinations are (i, j, k), (j, k, l), (k, l, i), and (l, i, j). Thus, four edges are traced from the start vertex. These edges are stored on a stack. During the incremental construction of the Voronoi graph, in each iteration of the algorithm, an edge is taken from the stack. If the edge already exists in the Voronoi graph, it will be ignored. Otherwise, the corresponding end vertex is computed. If the end vertex is already stored in the graph, only the edge is added to the graph. In case, the vertex does not exist, the vertex is added to the graph, too. Furthermore, the three possible new edges from the end vertex are pushed onto the edge stack. If the edge has no end vertex, which means it is unbounded, a synthetic end vertex on the edge curve is computed. This vertex is marked as *infinity vertex* and added to the graph. The procedure will be repeated until the stack is empty. The following steps summarize the algorithm.

```
initialize empty Voronoi graph G = (V, E)
compute start vertex, add it to V, and push the 4 edges on the stack
while stack is not empty do
    take edge e from stack
    if e ∉ E then
        compute end vertex v_e of e
        if e is unbounded then
            compute infinity vertex and add it to V
        else
            if v_e ∉ V then
                add v_e to V
                push the 3 new edges to the stack
            end if
        end if
        add e to E
    end if
end while
```

It is possible that the Voronoi graph contains more than one connected component. To get the complete graph, one has to run the described algorithm for each component. Furthermore, the algorithm does not compute completely unbounded edges, that is, edges which are not connected with any of the Voronoi vertices. In order to compute both, all components as well as completely unbounded edges, the method needs to check all input spheres. If a sphere is not a generator of a vertex or an edge in the current graph, the tracing algorithm starts again at this sphere. During the start vertex computation, it is possible that the algorithm fails, because the Voronoi region of the sphere

does not contain a vertex. Thus it is only bounded by a single face or by more than one face with unbounded edges.

Computation of a Start Vertex

The initial step of the algorithm comprises the computation of a valid start vertex. Two possibilities to achieve this are described in this section. The first possibility was presented by Medvedev et al. [164]. Their algorithm takes an arbitrary input sphere and detects its closest neighboring sphere. In the next step, the closest sphere to these two spheres is computed. Finally, the closest sphere to the three spheres is detected. To achieve this, the distance of k spheres to another input sphere is defined as the radius of the smallest sphere corresponding to a Voronoi vertex of the $k+1$ spheres in $k-$dimensional space. The computation for $k = 3$ was described in Section 6.1.1 and is equivalent in other dimensions. If a Voronoi vertex does not exist, the Voronoi region of the sphere is bounded either by a single unbounded face or by more than one face with unbounded edges. This is evaluated by analyzing the closest sphere for $k < 3$. Then, the algorithm proceeds with another input sphere.

The second possibility was presented by Kim et al. [109]. They suggested to add four synthetic input spheres. These four spheres are placed in a way that they definitely generate a Voronoi vertex. For example, the spheres can be arranged at the vertex positions of a tetrahedron residing outside the bounding box of the other spheres. Then, the graph computation is run until a Voronoi vertex is found that is generated by the original input spheres only. This vertex is then used as a new Voronoi start vertex, and the previous computed graph as well as the four synthetic input spheres are removed. Note that it is much more difficult to compute all components or unbound edges with this approach.

From the start vertex, four Voronoi edges need to be traced. Each edge is stored on the stack as a 5-tupel (g_1, g_2, g_3, v_s, s), where g_1, g_2, g_3 are the three generator spheres, v_s is the start vertex of the edge, and s is the fourth input sphere that together with g_1, g_2, g_3 generated v_s.

Computation of the End Vertex of an Edge

This is the core step of the algorithm. For each edge on the stack, represented by a 5-tupel (g_1, g_2, g_3, v_s, s), the end of the edge needs to be computed. An example of such an edge in two dimensions is illustrated in Figure 6.9. Recall that the curve containing the edge is symmetric with respect to the plane spanned by the positions of the generator spheres g_1, g_2, and g_3. Furthermore, the curve itself lies in a plane and each point on the curve corresponds to a sphere tangent to the three generator spheres. The start vertex v_s is the center of one of these tangent spheres. Recall that this sphere is always empty. The complete edge is defined by the positions of all empty spheres traced from the start vertex v_s until reaching the first non-empty sphere. The last traced empty

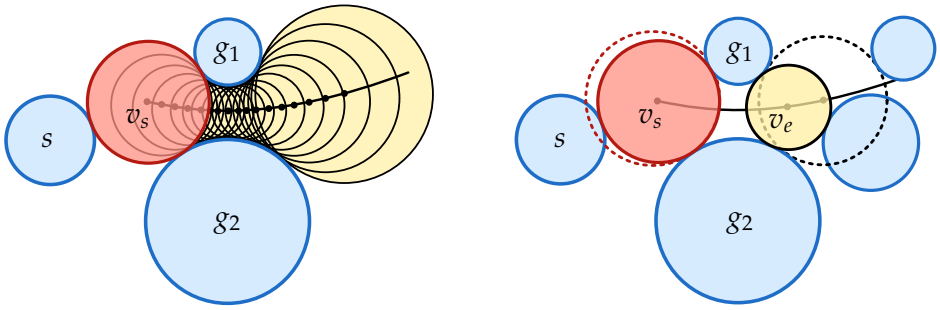


Figure 6.9: *A Voronoi edge tracing step in 2D. Left: the initial configuration with the start vertex v_s, the generator spheres g_1 and g_2 of the edge and the remaining tangent sphere s. Right: computation of the end vertex v_e, which is the closest empty sphere tangent to g_1 and g_2 and another input sphere.*

sphere defines the position of the edge's end vertex. Note that there are two directions into which the curve can be traced from the start vertex. However, only one of these directions is correct. When tracing the curve into the wrong direction, no empty tangent sphere can be found. Thus, the determination of the correct tracing direction is a crucial step of the algorithm. With the correct direction, the end vertex is the closest point to the start vertex along the curve such that the corresponding sphere is also tangent to one other input sphere from $I \setminus \{g_1, g_2, g_3\}$. The end vertex must be different to v_s, and the distance is measured along the edge. Thus, in order to determine the end vertex of an edge, two problems remain to be solved: (1) the determination of the correct direction, and (2) how to measure the distance of an arbirary point on the edge curve to the start vertex.

One possibility to measure the distance between two points on the edge curve is to use the angle between the vectors from a fixed point p_f to the two points (Figure 6.10). Note that p_f must not lie on the edge curve. Apart from other possibilities for p_f, it is a reasonable choice to select a point at the concave site of the curve on the symmetry plane, for example $p_f = 1/2(v_s + \tilde{v}_s)$, where $\tilde{v}_s$ is the point symmetric to v_s. For the special cases that $r_{g_1} = r_{g_2} = r_{g_3}$ or $v_s = \tilde{v}_s$, p_f lies on the edge curve and one has to select another arbitrary point on the symmetry plane, for example the center of a generator sphere. The distance between two points $a \in \mathbb{R}^3$ and $b \in \mathbb{R}^3$ on the edge curve can then be defined as the angle between $a - p_f$ and $b - p_f$. Here, the angle is based on the direction from a to b along the edge. Since the angle can be larger than π, it is not uniquely defined without a direction. This direction can be defined by the normal vector v_f of the plane in which the edge curve lies. Then the angular distance d_f between the points a and b is

$$d_f(a,b) = \begin{cases} \measuredangle(a - p_f, b - p_f), & \text{if } \langle v_f, (a - p_f) \times (b - p_f) \rangle > 0 \\ 2\pi - \measuredangle(a - p_f, b - p_f), & \text{else.} \end{cases}$$

The detection of the correct edge tracing direction defined by v_f can be based on the analysis of the Euclidean distance between sphere s and the tangent

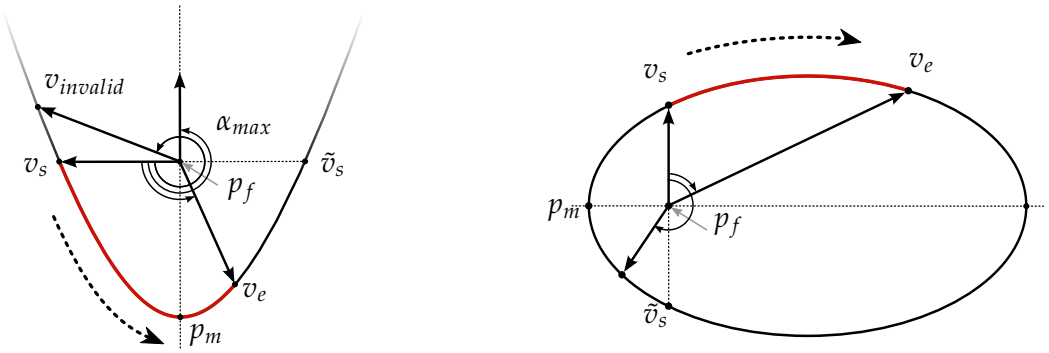


Figure 6.10: *Illustration of the edge-tracing in the plane of the edge for a non-closed edge curve (left) and a closed edge curve (right). The tracing direction is depicted by the dotted arrow. The start vertex is v_s and its symmetric point is $\tilde{v}_s$. The fixed point for the angular distance measurement is p_f and the point with the minimal distance to the 3 input spheres is p_m. A valid end vertex has an angular distance smaller than α_{max}. Furthermore, v_e illustrates a possible end vertex of the edge (red).*

spheres corresponding to the edge curve. When following the edge curve in one tracing direction, the corresponding spheres will intersect s while in the other direction they will not, (Figure 6.9). Consider the start vertex and its empty tangent sphere. Imagine that the edge is traced along one of the two directions by adding an $\epsilon \in \mathbb{R}$, with $|\epsilon|$ being very small, to the radius of the start vertex sphere and computing the position of the new sphere for this radius such that it is tangent to the spheres g_1, g_2, and g_3. Due to the symmetry of the curve, there are two possible positions for a sphere with this radius, but the one on the side of the start vertex is selected. If this sphere intersects with sphere s, the tracing direction is wrong, otherwise it is correct. This approach to detect the direction is correct for an arbitrarily small $|\epsilon|$ and was applied by Lindow et al. [149] for protein data. However, the choice of ϵ is difficult in practice and might result in a numerically unstable computation. To circumvent this problem, here a new analytical way to compute v_f is presented.

Let $e(t)$ be a parametric representation of the edge curve with $e(0) = v_s$.

> **Lemma**. *The tangent $e'(0)$ in v_s points into the correct tracing direction if*
>
> $$\left\langle e'(0), \frac{v_s - p_s}{\|v_s - p_s\|} - \frac{v_s - p_{g_1}}{\|v_s - p_{g_1}\|} \right\rangle > 0$$
>
> *Otherwise it points into the opposite direction.*

Proof. For each point $e(t)$, the radius of the corresponding tangent sphere is $r_e(t) = \|e(t) - p_{g_1}\| - r_{g_1}$. Note that g_2 or g_3 could also be used to define $r_e(t)$. The Euclidean distance $d_e(t)$ of the tangent spheres to the input sphere s is then given by

$$d_e(t) = \|e(t) - p_s\| - r_s - r_e(t) = \|e(t) - p_s\| - \|e(t) - p_{g_1}\| - r_s + r_{g_1}.$$

As described above, in one tracing direction the distance $d_e(t)$ decreases, because the tangent spheres intersect sphere s and in the correct tracing direction $d_e(t)$ increases. Hence, only the sign of the derivative of $d_e(t)$ for $t = 0$ has to be analyzed to get the correct tracing direction. Therefore, consider first the derivative of $\|e(t) - p\|$, where $p \in \mathbb{R}^3$ is an arbitary point.

$$\begin{aligned}\|e(t) - p\|' &= \left(\sqrt{e^2(t) - 2\langle e(t), p\rangle + p^2}\right)' \\ &= \frac{2\langle e(t), e'(t)\rangle - 2\langle e'(t), p\rangle}{2\|e(t) - p\|} \\ &= \frac{\langle e'(t), e(t) - p\rangle}{\|e(t) - p\|}\end{aligned}$$

With this auxiliary calculation, the derivative of $d_e(t)$ is

$$\begin{aligned}d_e'(t) &= \frac{\langle e'(t), e(t) - p_s\rangle}{\|e(t) - p_s\|} - \frac{\langle e'(t), e(t) - p_{g_1}\rangle}{\|e(t) - p_{g_1}\|} \\ &= \left\langle e'(t), \frac{e(t) - p_s}{\|e(t) - p_s\|} - \frac{e(t) - p_{g_1}}{\|e(t) - p_{g_1}\|}\right\rangle.\end{aligned}$$

Thus, the sign of the derivative depends only on two normalized vectors and $e'(t)$, which is a tangent vector in $e(t)$. A closer look at the case $t = 0$, reveals

$$d_e'(0) = \left\langle e'(0), \frac{v_s - p_s}{\|v_s - p_s\|} - \frac{v_s - p_{g_1}}{\|v_s - p_{g_1}\|}\right\rangle.$$

□

The only unknown part in this formula is the tangent vector $e'(0)$. With the system of equations (6.7) one can compute a tangent vector in a point on the edge curve. The direction of the tangent is defined by the sign of the constant c. For a positive c the tangent points to the symmetry plane. Because only the sign of the scalar product is of interest for the direction detection, it is not necessary to have the correct length of $e'(0)$. Furthermore, by a rearrangement of $d_e'(t)$ one can see that the equation is independent of the selected generator sphere:

$$\left\langle e'(t), \frac{e(t) - p_s}{\|e(t) - p_s\|} - \frac{e(t) - p_{g_1}}{\|e(t) - p_{g_1}\|}\right\rangle =$$

$$\left\langle e'(t), \frac{e(t) - p_s}{\|e(t) - p_s\|}\right\rangle - \left\langle e'(t), \frac{e(t) - p_{g_1}}{\|e(t) - p_{g_1}\|}\right\rangle$$

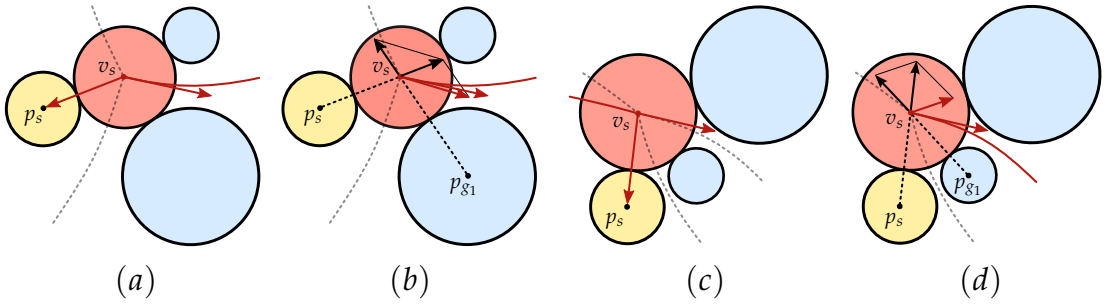


Figure 6.11: *Detection of the edge tracing direction by Medvedev et al. [164] (a) and (c), compared to the presented detection method (b) and (d). The start vertex and the traced edge are depicted in red. The sign of the scalar product of the red vectors defines the tracing direction. Note that for (a) and (c) the tracing direction is equal to the tangent if the scalar product is negative, which is correct in (a) but not in (c). For (b) and (d) the tangent points into the tracing direction if the scalar product is positive, which is correct in both cases.*

The first part of the difference is independent of the input sphere g_1 and the second part describes the angle between the tangent and the normalized vector from p_{g_1} to $e(t)$. Recall that this angle is equal for the three input spheres (Section 6.1.1). Thus, $d'_e(t)$ is independent of the selection of the generator sphere. The correct direction is given by the sign of $d'(0)$. If $d'(0) > 0$ then $e'(0)$ points into the correct direction, otherwise it is the opposite direction. Let $p_m \in \mathbb{R}^3$ be the point on the edge curve with the minimal distance to g_1, g_2, and g_3 and $t > 0$ for $e(t) = p_m$, then the direction can be represented by

$$v_f = \begin{cases} (v_s - p_f) \times (p_m - p_f), & \text{if} \quad d'_e(0) \geq 0 \\ (p_m - p_f) \times (v_s - p_f), & \text{else.} \end{cases} \tag{6.9}$$

In contrast, Medvedev et al. [164] proposed the computation of the direction by the analysis of $e'(0)$ and only the vector $p_s - v_s$. If $\langle e'(0), p_s - v_s \rangle < 0$, the tangent $e'(0)$ points already into the correct direction, otherwise it points into the opposite direction. A simple 2-dimensional example, depicted in Figure 6.11, shows that this can lead to a wrong direction.

Using the result from the Lemma, which leads to Equation (6.9), it is possible to measure distances along the correct edge tracing direction. However, for non-closed edge curves, the maximal distance α_{max} for a possible end vertex is smaller then 2π (Figure 6.10). For non-closed curves, α_{max} is given by

$$\alpha_{max} = \begin{cases} \pi + \measuredangle(v_s - p_f, p_m - p_f), & \text{if} \quad d'_e(0) \geq 0 \\ \pi - \measuredangle(v_s - p_f, p_m - p_f), & \text{else.} \end{cases} \tag{6.10}$$

For closed curves, $\alpha_{max} = 2\pi$. If $v_s = p_m$, then an arbitrary other point on the edge curve needs to be selected. Note that the direction of $e(t)$ depends on the choice of this point. The whole method to compute the end vertex is summarized by the following algorithm.

1: mark end vertex v_e as invalid
2: compute p_m and edge type (closed or non-closed)

```
3: compute fixed point p_f for distance measurement
4: compute tracing direction v_f for distance measurement (6.9)
5: compute maximal distance α_max (6.10)
6: for c ∈ I \ {g1, g2, g3} do
7:    compute possible vertices v_ci for g1, g2, g3, and c (Section 6.1.1).
8:    for all v_ci do
9:       α ← d_f(v_ci, v_s)
10:      if α < α_max then
11:         v_e ← v_ci
12:         α_max ← α
13:         mark v_e as valid
14:      end if
15:   end for
16: end for
```

If this algorithm does not find an end vertex, then the traced edge is unbounded in one direction and, hence, it is infinitely long. Handling such cases is described in the following section.

Computation of Infinity Vertices

For the visualization and analysis of Voronoi diagrams, it is often useful to restrict an unbounded edge by cutting off the part going to infinity. Therefore, a synthetic vertex along the tracing direction is added and marked as infinity vertex. Recall that only non-closed edge curves can be unbounded in the sense that they have only one or no end and thus extend to infinity.

The edge tracing algorithm detects only unbounded edges with one vertex. Since the edge is unbounded in one direction, there exists a point on the edge in the correct tracing direction that has a larger distance to the spheres g_1, g_2, and g_3 than the start vertex v_s. Depending on the application one can choose a distance d_{inf}, which should be larger than the distance of v_s to g_1, g_2, and g_3. Then, the two possible points $p_{inf_{1,2}}$ with distance d_{inf} to g_1, g_2, and g_3 are computed. Only one of the points p_{inf_k} has an angular distance $d_f(v_s, p_{inf_k})$ that is smaller than the maximal distance α_{max}.

Extensions

In 2006, Cho et al. [41] presented an extension of the edge-tracing algorithm to reduce the search space for the 4th input sphere that generates the end vertex of a Voronoi edge. While this extension accelerates the algorithm, the time complexity for this step is still quadratic. Therefore, Maňák et al. [161] presented an optimization that incrementally reduces the search space to achieve an expected time complexity that is linear to the number of input spheres. Even faster computations can be achieved by using a combination of the reduced search space strategy and a 3D data structure, as described by Lindow et. [149]. An additional acceleration can be achieved by parallelization.

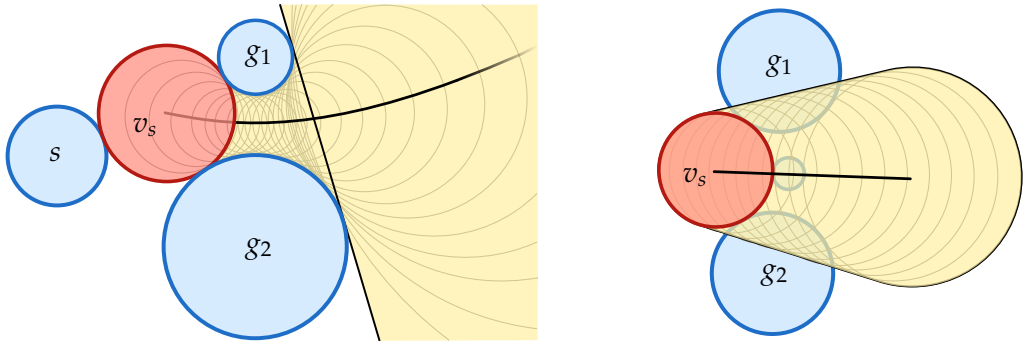


Figure 6.12: *Illustration of the search space (yellow) for the end vertex computation of a non-closed edge (left) and a closed edge (right). The search space (yellow) is the union of all spheres that correspond to potential end vertices of the edge. All input spheres that do not intersect the search space (yellow) can be ignored for the end vertex computation.*

Reduced Search Space During the computation of an end vertex of an edge, nearly all input spheres need to be investigated. For each of these input spheres, the tangent spheres that possibly represent the end vertex are computed. Since this is computationally expensive, a reduction of the investigated input spheres would accelerate the algorithm a lot. To achieve this, one has to quickly identify input spheres that cannot be a generator of the end vertex. Note that the space of possible end vertices is restricted to all points on the edge curve from the start vertex along the correct tracing direction. This means that only an input sphere that intersects a corresponding tangent sphere of this space can be a generator sphere of the possible end vertex. Additionally, if a possible end vertex is detected, the space of possible end vertices further reduces to all points on the edge curve between the start vertex and the potential end vertex. Thus, the space of possible end vertices successively reduces.

For non-closed edge curves, the space of the tangent spheres that correspond to possible end vertices can be decomposed into a half-space and the tangent spheres between this half-space and the start vertex (Figure 6.12). The half-space corresponds to the tangent sphere, the center of which lies on the edge curve at infinity into the correct tracing direction. It is given by the plane tangent to the edge generator spheres g_1, g_2, and g_3. All input spheres that do not intersect one of the two spaces can be ignored for the end vertex detection.

For closed edge curves, the space of the tangent spheres that correspond to the possible vertices of the edge is described by a generalized torus. The torus has an elliptic edge curve as skeleton with a varying small radius from the minimal to the maximal tangent sphere (Figure 6.12). All input spheres that do not intersect the torus can be ignored for the end vertex detection.

In order to quickly identify a potential end vertex that is as close as possible to the expected end vertex, Maňák et al. [161] use the Delaunay triangulation of the centers of the spheres as spatial data structure. However, this requires

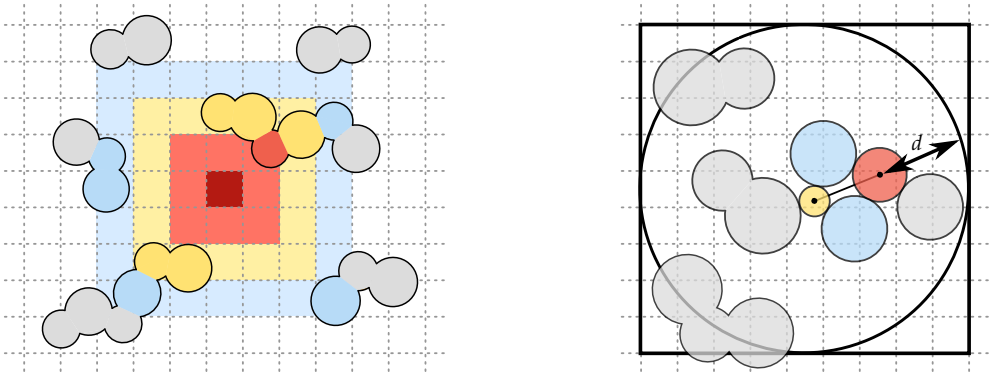


Figure 6.13: *The left image shows the expansion of the search space by looping over rings of grid cells starting from the red cell. On the right side, the break condition $d > r_{max} + r_e$ is illustrated for the blue generator spheres of the edge with the yellow start vertex and the red end vertex.*

the pre-computation of the Delaunay complex which produces an additional overhead.

3D Data Structure Instead of using the optimization by Maňák et al. [161], it is more efficient to reduce the search space by using a 3-dimensional grid as spatial data structure (Section 2.5). The grid stores all input spheres in a way that each grid cell contains all input spheres whose centers lie inside the cell. The grid allows a fast detection of the end vertex without touching all $n - 3$ input spheres. This is done by looping over rings of grid cells. In the first loop, all spheres in the cell of the start vertex are tested. The second loop tests all spheres of all grid cells that touch the cell of the first loop, and so on (Figure 6.13). When a state is reached at which the distance between the current end vertex and the position of the nearest remaining potential input sphere with greatest possible radius r_{max} is greater than the sum of their radii, the end vertex has been found. A lower bound d of this distance for the ith loop is given by

$$d = (i - \frac{1}{2})c_{min} - \|c_g - v_e\| > r_{max} + r_e$$

where c_g is the center of the cell of the first loop and c_{min} is the smallest cell dimension. The vertex position is v_e and its radius is r_e (Figure 6.13). If d becomes larger than $r_{max} + r_e$, the end vertex has been found. The first part of the distance is the radius of the maximal inscribed sphere of the current search space in the grid. Thus, each input sphere, the center of which lies inside this sphere has already been checked. Overall, the equation describes the distance of the sphere that corresponds to the current end vertex to the boundary of the inscribed sphere. If the input spheres are relatively small compared to their overall bounding box and if the spheres are spatially distributed such that their distances to their closest neighbors is similar, the

grid reduces the complexity for the detection of the end vertex to O(1) in regions with high sphere density, but does not speed up the vertex detection at boundaries where edges are unbounded.

Parallelization To further improve the run time, a parallel version of the algorithm was proposed [149]. For this, two stacks of edges are required, an active and an inactive one. The algorithm computes all end vertices for all edges of the active stack in parallel. Then, sequentially the vertices are popped from the active stack. If the vertex already exists in the graph, only the edge is added, otherwise the vertex is also added and the three possible new edges are pushed onto the inactive stack. In the last step, the active and inactive stacks are switched and the procedure is repeated until the active stack is empty. Note that with this strategy the most expensive part of the algorithm is done in parallel.

Boundary Handling For most applications, only the Voronoi elements close to the input spheres are required, for example, all Voronoi elements inside the bounding box of the input spheres. This can be used to improve the computation in two ways. First, only a part of the Voronoi graph needs to be computed, which reduces the computation time, and second, the algorithm becomes more stable because the numerical errors are greater for the Voronoi elements that lie far away from the input spheres. To achieve this, the algorithm simply does not trace edges at Voronoi vertices that lie outside a user-defined region of interest. Additionally, the Voronoi elements can be cut with the region of interest in a post-processing step. For example, if the region of interest is a box, one can cut the Voronoi elements by this box. Therefore it is necessary to compute the intersections of the Voronoi elements and planes. During the cutting process Voronoi elements can completely disappear because they lie outside the box, or they shrink in one direction or even split. For Voronoi elements that will be cut, new artificial Voronoi vertices need to be generated, similar to the Voronoi vertices for the unbounded Voronoi edges.

6.1.3 Topological Structure

The Voronoi diagram of spheres describes the complete topology of the Euclidean distance function of the input spheres. Therefore, the definition of the critical points is adapted from Siersma [215]. The critical points are given by the intersection of the Voronoi elements and their dual structures. The dual structure of a Voronoi diagram of spheres is called quasi-triangulation [112, 111], which is the analogon of a Delaunay complex for a classical Voronoi diagram. The quasi-triangulation consists of regions, faces, edges and vertices. Each region is topologically a tetrahedron defined by the centers of the four generator spheres of its dual Voronoi vertex. A face is a triangle given by the centers of the three generators of its dual Voronoi edge. Accordingly, an edge

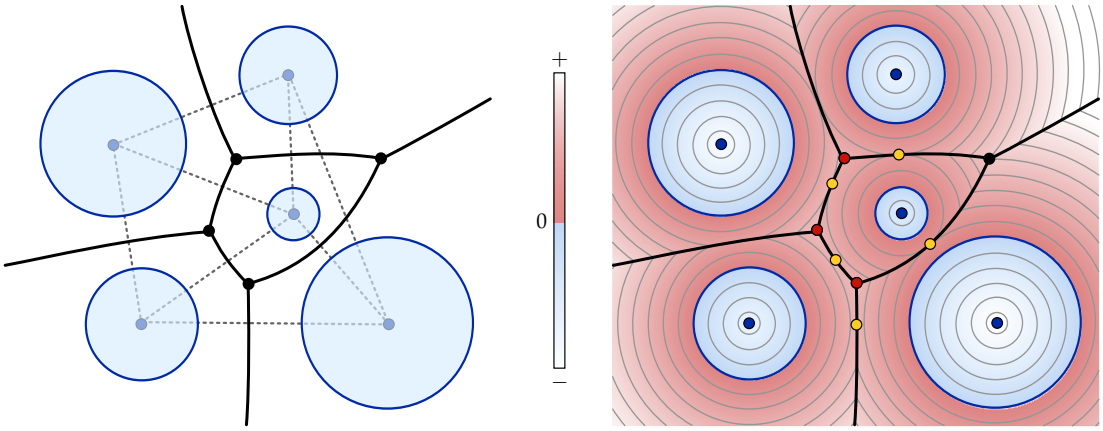


Figure 6.14: *Voronoi diagram of spheres and dual quasi-triangulation (left). Distance function of the input spheres with the corresponding critical points (right). Positive distances are depicted in red and negative in blue. Maxima are shown as red dots, minima as blue dots, and saddles as yellow dots.*

is a line segment connecting the centers of the generator spheres of the dual Voronoi face, and each vertex corresponds to the center of the sphere of its dual Voronoi region. Thus, the quasi-triangulation creates a 3-dimensional triangulation of the centers of the input spheres. However, in contrast to the Delaunay complex of the classical Voronoi diagram, the quasi-triangulation is not always a valid simplical complex.

The critical points of the distance function can be distinguished in four different types:

- Minima. The minima are the centers of the input spheres that lie inside the corresponding Voronoi region.
- Index-1 saddles. An index-1 saddle lies in the Voronoi face at the intersection point with its dual quasi-triangulation edge.
- Index-2 saddles. An index-2 saddle lies on a Voronoi edge at the intersection point with its dual quasi-triangulation face.
- Maxima. The maxima are the Voronoi vertices that lie inside their dual quasi-triangulation region.

An example for the 2-dimensional case is shown in Figure 6.14. Note that the critical points do not have a one-to-one correspondence to the Voronoi structures, that is, there does not exist a critical point on every Voronoi face, edge, or vertex. For each nonempty Voronoi region, the center of the input sphere and thus the quasi-triangulation vertex is always inside the region.

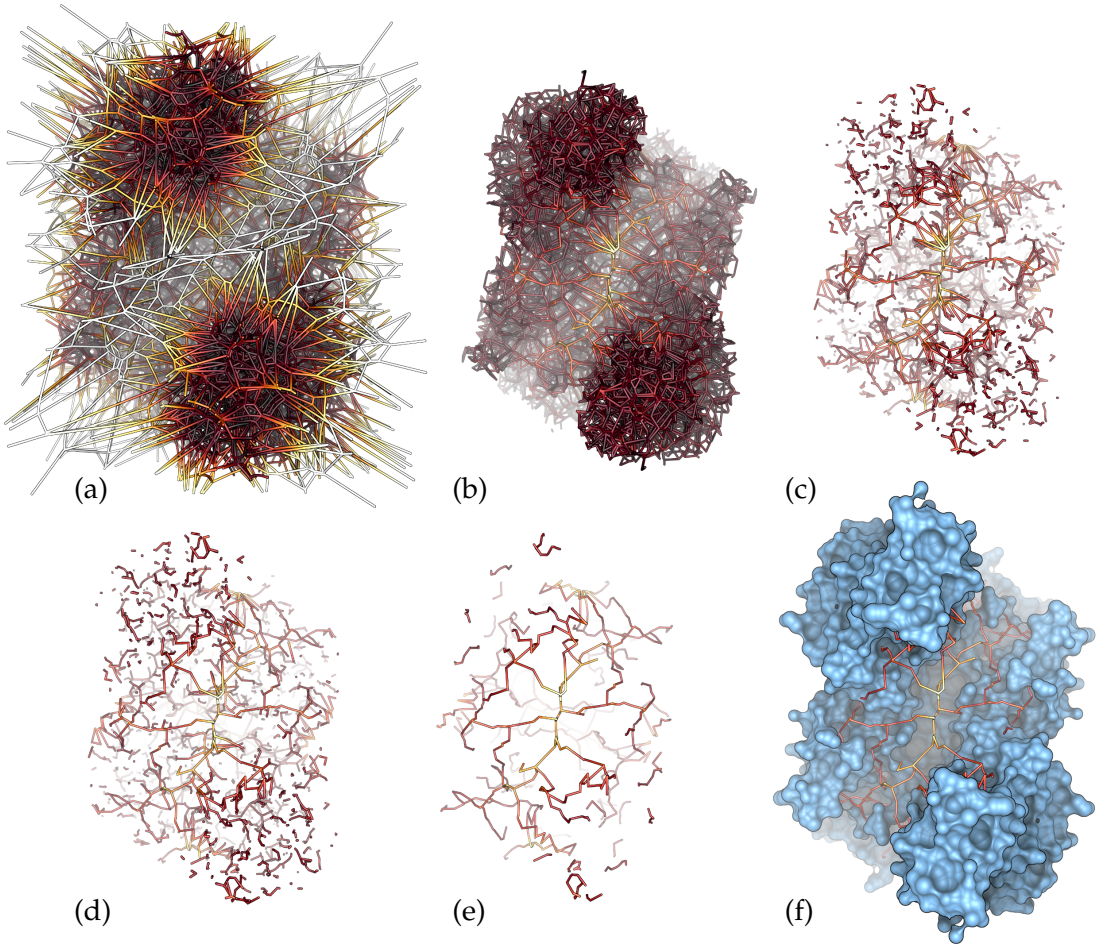


Figure 6.15: *Path filtering pipeline for a putative membrane protein from Corynebacterium diphtheriae (pdb: 3C8I). Starting from the Voronoi diagram clipped by the bounding box of the molecule (a) the following filters are applied: Boundary filter with threshold 0.4 (b), probe filter with $r_p = 1.2\,\mathring{A}$ (c), cycle filter (d), branch filter with threshold 0.9 (e). The most significant paths (e) are visualized together with the SES (f). The path color illustrates the distance to the atoms from black (close distances) to white (far distances).*

6.2 Path and Cavity Computation

Consider the Voronoi diagram of the atom spheres of a molecule. As mentioned before, the vertices and edges describe all paths with the locally maximal distance to the atom spheres. Hence, these paths describe the geometrical optimal paths for spheres, especially, they include all maximal molecular paths for a given probe sphere. In the following, it is described how to extract all maximal molecular paths from the Voronoi diagram and the shape of the corresponding cavities. Additionally, a method is proposed to filter the most significant paths from the set of all maximal paths. A result of this pipeline can be seen in Figure 6.15

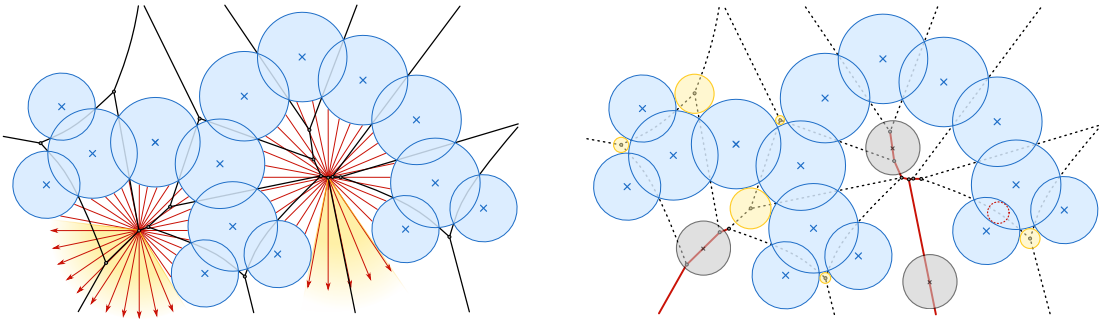


Figure 6.16: *The computation of the boundary values for two Voronoi vertices (left). The directions are discretized (red) to approximate the analytical solution (yellow). The probe filter (right) removes all edges of which the minimal distance is smaller than the probe radius (grey). Thus all dotted edges are removed, and only the red ones remain.*

6.2.1 Boundary Filter

The Voronoi diagram contains many edges outside the domain of the molecule, and a lot of them are infinitely long. These edges are non-relevant for the analysis of molecular paths and create visual clutter around the molecule. In addition, these paths do not lie inside real cavities. Hence, a boundary needs to be defined that separates regions inside and outside the molecule as described in Section 2.4.

To remove these parts, the Voronoi diagram is first cut by the axis-aligned bounding box. Of course, this does not suffice for molecules whose shapes are not box-like. So in the second step, the remaining paths are removed by applying a simple technique that is also used in computer graphics for the approximation of ambient occlusion. The more ambient light is received by a Voronoi vertex, the more it lies outside the molecule. To realize this, at each Voronoi vertex, a set of rays is cast. The rays are uniformly distributed in all directions. The more rays hit the molecule the more likely it is that the vertex resides inside the molecule. For vertices inside cavities and channels, many rays will intersect the molecule. One interpretation for this method is that vertices with sparse ambient illumination are inside or near the molecule. The ratio of ambient occlusion is given by the ratio between the number of rays hitting the molecule and the overall number of rays. All vertices with an ambient occlusion ratio smaller than a user-defined threshold are removed together with their interconnected edges. Note that a value of 0.5 approximately cuts the Voronoi vertices with the convex hull of the molecule. Figure 6.16 (left) illustrates the analytical and discrete boundary values for two vertices and a result of the filter is shown in Figure 6.15 (b).

In order to implement this technique efficiently, the set of uniformly distributed rays can be precomputed using, for example, sphere sampling algorithms or subdivision of an icosahedron [190]. This distribution is then used for all vertices. Like in the previous ray casting algorithm (Section 4.2.1), the atom spheres are stored in a grid and the ray traversal by Amanatides [2]

is used to quickly check for ray atom intersections. Since each ray and each Voronoi vertex can be handled independently, the algorithm can be easily parallelized using, for example, OpenCL. To achieve this, the grid can be realized by two simple arrays that store the cells and the atom sphere, as it was described in Section 2.5. A further array stores the precomputed ray directions. Then the parallel kernel program computes either the intersection test for a single ray or all tests for a single vertex.

6.2.2 Probe Filter

In contrast to the boundary filter, which removes paths outside the domain of the molecule, the probe filter removes paths whose distance to the atom spheres is too small. Similar to the SES, the probe is a hard sphere with a user-defined radius $r_p \in \mathbb{R}$, which approximates the size of an ion, solvent, or substrate. The probe is not allowed to penetrate the atom spheres of the molecule while moving its center along the Voronoi edges. Hence, all Voronoi edges with a smaller minimal distance to the atom spheres than r_p cannot be visited or completely passed by the probe. For this reason these edges are removed, because they are not part of a maximal molecular path.

The point on the edge with the minimal distance to the atom spheres is the index-2 saddle on the edge. This is the intersection point of the edge with the dual face of the quasi-triangulation. Its computation was described in Section 6.1.1. If the edge does not intersect the dual face, which means the edge does not contain an index-2 saddle, the point with the minimal distance to the atom spheres is one of the end vertices of the edge. After removing all edges whose minimal distance is smaller than r_p, some of the Voronoi vertices are possibly not interconnected with an edge anymore. These vertices will be also removed. The filter is illustrated in Figure 6.16 (right) and a result can be seen in Figure 6.15(c).

The user-defined probe should be selected carefully. For example, the minimal bounding sphere of a substrate used as probe will remove too many edges and thus underestimates the accessibility. Using a too small sphere, on the other hand, leads to an overestimation of the accessibility. The maximal probe that guarantees to not miss a geometrically accessible path, is the maximal sphere which lies completely inside the van der Waals surface of the substrate. For small substrates, this sphere is often equivalent to the largest atom sphere in the substrate. Furthermore, the probe depends on the selected atomic radii. In many applications using van der Waals radii, r_p is set to 1.4 Å to approximate a water molecule

6.2.3 Cavity Extraction

After filtering, the remaining graph of Voronoi vertices and edges contains all maximal molecular paths that are accessible to the selected probe sphere. And each connected component in the reduced graph represents the skeleton

of a single cavity. For simplicity, from now on, the connected components are called *path components*. Recall that each Voronoi vertex defines an empty sphere tangent to four generator atom spheres whose Voronoi regions created the vertex. Furthermore, each point on a Voronoi edge defines an empty sphere tangent to the three generator atoms whose Voronoi regions created the edge. In order to visualize, measure and analyze the shape of a cavity, a finite subset of empty spheres of the corresponding path component is used to represent the cavity. The union of theses empty spheres approximates the volume of the cavity.

The quality of this approximation naturally depends on the sampling density along the edges. The empty spheres along the Voronoi edges are placed such that the largest circle inside the intersection of neighboring spheres is at least r_p. With this criterion, a probe sphere is guaranteed to be able to move along all the paths of a path component without colliding with the surface of the cavity approximated by the set of empty spheres. In detail, the following procedure is used to extract the spheres. First, all spheres which correspond to Voronoi vertices are extracted and all spheres on the edges which correspond to index-2 saddles. Then the intersection circle between two neighboring spheres is computed. If the radius of this circle is smaller than r_p, a further sphere on the edge is extracted between the two spheres. In the following, the term cavity is also used for its approximation by the finite set of spheres.

6.2.4 Significant Paths

Although the possible molecular paths are already reduced to the set of maximal molecular paths, it is still visually difficult to analyze them. Many of the paths are redundant, because there exists a more significant representative path in the same cavity. The detection of these redundant paths was already described in a previous work [149]. However, the filtering requires 4 different steps and is based on geometrical heuristics. In this section, an improved version is presented, which can be separated in 2 different steps. In the first step, all redundant paths are removed by analyzing cycles of Voronoi edges. The remaining branches are evaluated in the second step by an importance-based measurement. In contrast to the previous version, the new filtering is much faster and geometrically more stable.

Topology Graph Creation

The filtering of the most significant paths is not directly applied to the remaining Voronoi vertices and edges, but on a new graph, called *topology graph*. The topology graph is a directed vertex-labeled graph $\vec{G} = \{V, E\}$ that is based on the topology of the distance function (Section 6.1.3). Its undirected version is denoted by G only. The set of vertices V consist of all remaining Voronoi vertices and all index-2 saddle points on the remaining edges. The corresponding

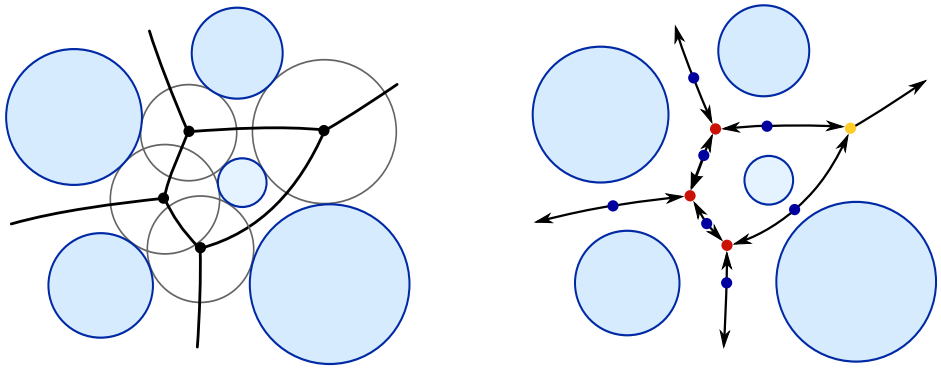


Figure 6.17: *Illustration of the Voronoi diagram (left) and the corresponding topology graph (right). Maxima are shown in red, minima in blue and regular vertices in yellow.*

labels are given by the mapping $l : V \to \{maximum, minimum, regular\}$. For a vertex $v \in V$, $l(v)$ is defined as

- **maximum** if v is a local maximum of the distance function.
- **minimum** if v is an index-2 saddle of the distance function. This means that index-2 saddles of the distance function turn into minima in the topology graph. They represent the narrowest points on the maximal molecular paths.
- **regular** if v is neither a maximum nor a minimum.

The edges E of $\vec{G}$ are now constructed as follows. For each $u \in V$ with $l(u) = minimum$, two edges (u, v) and (u, w) are added to E, where v and w are the end vertices of the corresponding Voronoi edge on which u lies. For each remaining Voronoi edge without an index-2 saddle, an edge (u, v) is added to E if $d(u) < d(v)$, where u and v are again the end vertices of the corresponding Voronoi edge. Note that v can be either a *regular* vertex or a *maximum*. In Figure 6.17 a topology graph of a Voronoi diagram is shown for the 2-dimensional case. In the topology graph, each minimum has only outgoing edges, each maximum only incoming edges, and each regular vertex incoming and outgoing edges.

Cycle Filter

Based on the structure of a Voronoi diagram, the undirected topology graph consists of many cycles that bound the Voronoi faces. If a cycle lies completely in a single cavity, it always consists of a redundant path. By removing an arbitrary edge, the probe is still able to visit all vertices in the topology graph. Let $w : E \to \mathbb{R}$ be the minimal distance of an edge to the atom spheres. This means $w((u, v)) = \min\{d(v), d(w)\}$, for $(u, v) \in E$. By removing the smallest edge in such a cycle the remaining paths have a larger distance to the atom spheres than the paths including the removed edge. For this reason, the paths

that include the minimal edge are defined as redundant and the minimal edge is removed.

The idea of this cycle cancellation is similar to the previous presented cycle filter. In the former version [149], cycles were detected by applying a modified depth first search with backtracking on the undirected topology graph. The investigation if a cycle lies completely inside a single cavity was done by a geometrical heuristic. A set of rays was cast from the center of the cycle to regularly sampled points along the cycle. If none of the rays hit the atom spheres, the cycle lay completely inside a cavity. With the new method the detection of cycles is accelerated and the geometrical heuristic is replaced by an analytical solution.

Instead of searching cycles in the topology graph with a depth first search, all minimal cycles are already described by completely bounded Voronoi faces. The search for these bounded Voronoi faces can be done quickly by tracing the edges in the topology graph. Since the Voronoi diagram is non-degenerate, each face is created by exactly two atom spheres and each edge by exactly three atom spheres. So each Voronoi edge is part of three faces. Note that the edges in the topology graph correspond to edges of the Voronoi diagram. Let $e \in E$ be such an edge whose corresponding Voronoi edge was generated by the atom spheres $i, j, k \in I$. Then e is a boundary of the faces given by the atom spheres (i, j), (j, k), and (k, i). To get all edges of the face given by (i, j), all those adjacent edges to e are traced, whose corresponding Voronoi edges were also generated by (i, j) and a third atom sphere. Based on the properties of the Voronoi diagram, at most one edge at each end of e can exist. The tracing is continued with the new edges until the cycle is closed or the tracing stops because no adjacent edge was generated by i and j. In the first case, the edges are stored in a list according to their cyclic order. In the second case, the face is either unbounded, or an edge of the face was removed in a previous filtering step. For each edge, three possible cycles can be traced.

A cycle lies completely inside a cavity if each point of the corresponding face is a valid center of the probe such that the probe does not intersect the atom spheres. In other words, if the minimal distance of the face to the atom spheres is larger than r_p, the face lies inside a single cavity. The point with the minimal distance on a face is the index-1 saddle, given by the intersection of the face with its dual quasi-triangulation edge. If the intersection is empty and the face does not contain an index-1 saddle, the minimal distance is given by a point on a boundary Voronoi edge. Hence, it is either an index-2 saddle or a Voronoi vertex. In the topology graph, this corresponds to the vertex of the cycle with the minimal distance to the atom spheres. All cycles whose corresponding minimal face distance is smaller than r_p are removed from the following investigations.

The remaining cycles are processed iteratively until the list of valid cycles is empty. First, the redundant part of the cycle is removed. It is defined as the edge with the minimal distance to the atom spheres, and all traced adjacent vertices and edges until the end branching vertices are detected. Note that a

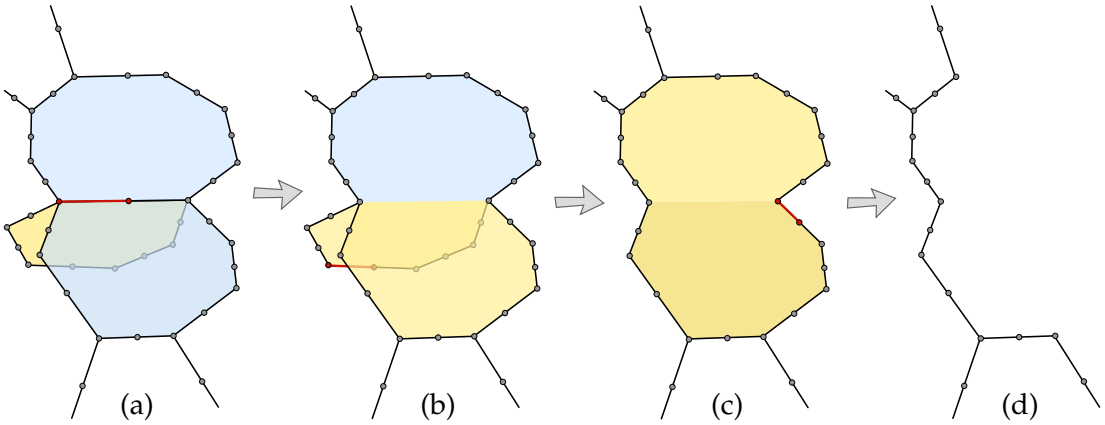


Figure 6.18: *Illustration of four steps of the cycle filter. In the first step (a), the yellow cycle is selected the minimal edge of which is highlighted in red. The cycle shares this edge with two other cycles (blue). After removing the edge and the corresponding branches, only the two neighboring cycles remain (b). Then, the next cycle is selected (yellow) and its minimal edge is selected (red). After removing the cycle, only one cycle remains (c). Finally, the minimal edge and the corresponding branches of this cycle are removed (d).*

branching vertex in G has at least three adjacent edges. The redundant part of a cycle is possibly part of other cycles. To remove this part, it is necessary to check if all involved cycles share only the redundant part and no further vertices and edges. In this case, all vertices and edges of the redundant part can be removed in G except the end vertices. Otherwise, the algorithm proceeds with the next cycle. After removing the redundant part, the other involved cycles are extended by the remaining part of the original cycle and the original cycle is removed from the list of cycles. This can be seen as a merging of the corresponding Voronoi faces (Figure 6.18). The new cycles does not correspond to a single Voronoi face anymore but to a set of neighbored faces that all lie in the same cavity. For the special case that a cycle has only a single branching vertex, only the minimal edge is removed.

After applying the cycle filter, the topology graph either contains no more cycles or the remaining cycle can not be removed. Thus only significant paths remain in the topology graph. An illustration of the cycle filter is shown in Figure 6.15

Branch Filter

After removing all redundant parts in cycles, there still exist many branches that do not represent significant paths. A branch is a connected subgraph G_b of G, where each vertex has at most two adjacent edges in G and at least one vertex has degree one. A branch with exactly one vertex of degree one is connected to a branching vertex in G. During this filtering step insignificant branches are detected and removed from the topology graph.

A branch that consists only of *regular* vertices is called *regular branch*. A regular branch describes a directed path in $\vec{G}$. From the beginning to the end

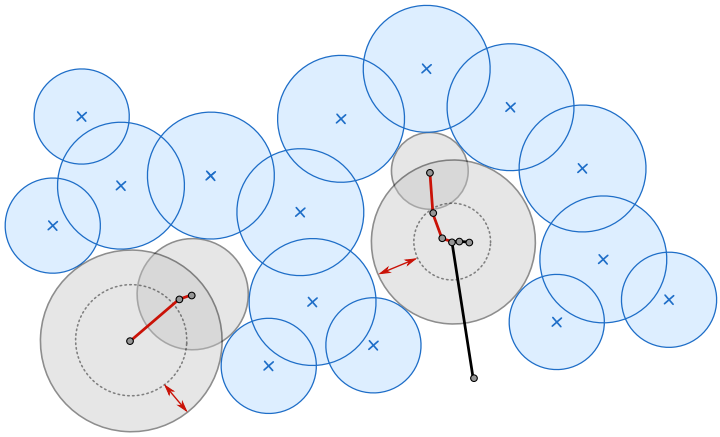


Figure 6.19: *Illustration of the importance measurement of the branch filter for two examples (red). The minimal and maximal distances along the branches are illustrated by the grey circles.*

of the path, the distance to the atom spheres monotonically increases. This means a regular branch does not contain a real bottleneck. Accordingly the branch does not describe a path into a new region of the cavity. For this reason, regular branches can be removed from the topology graph.

A further branch filtering cannot be done without a user-defined evaluation, because all remaining paths are geometrically possible significant paths. However, while these paths are geometrically significant, the user might want to filter even more branches. One possible measurement of the importance of a branch can be derived from difference between the smallest vertex and the largest vertex (Figure 6.19). The smallest vertex is defined as the vertex with the smallest distance to the atom spheres. Accordingly, the largest vertex has the greatest distance to the atom spheres. The difference between the smallest and largest vertex can be either evaluated using the absolute difference of their distances or relatively using the ratio between the smallest and the largest distance. The threshold for this difference is given by the branch significance parameter $b_s \in \mathbb{R}$. For the absolute difference, all branches are removed whose significance is smaller than b_s. In the case of the relative difference, all branches are removed whose significance ratio is larger than b_s. Note that $b_s \in [0, 1]$ for the relative significance. An example of the branch filter, using the relative importance, can be seen in Figure 6.15

6.3 Results

In the following, the performance of the computation of the Voronoi diagram and the filtering in order to compute the cavities and the significant molecular paths is analyzed. To this end, several molecules from the PDB [184] are investigated. Table 6.1 shows the results for the full Path computation pipeline.

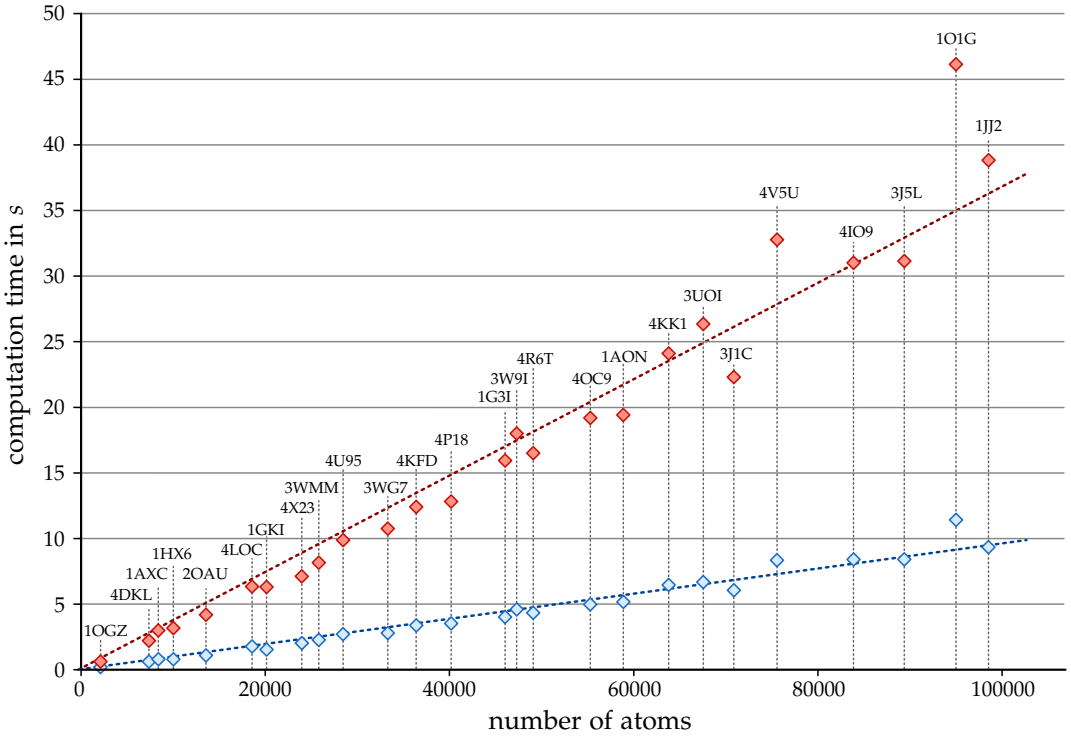


Figure 6.20: *Times for the computation of the Voronoi diagram of spheres for 27 molecules from the PDB [184] using a single CPU core (red) and 6 cores (blue).*

All results presented here were obtained on an Intel Xeon X5650 E5540 2.66 GHz system with 6 cores and an NVIDIA Geforce GTX470 graphics card.

6.3.1 Voronoi Diagram Computation

The computations of the Voronoi diagram were run on a single CPU core as well as on multiple cores using OpenMP [175]. The computation times for 1 and 6 cores are given in Table 6.1. The algorithm was also tested on other machines with 4 and 8 cores. On all machines, the performance speed-up due to parallelization was approximately $0.5 \cdot c$, where c is the number cores. The theoretical time complexity of the algorithm is $O(n \cdot e)$ [109], where n is the number of input spheres and e is the number of Voronoi edges. However, with the optimizations presented in Section 6.1.2, for the tested PDB molecules with up to 100 000 atoms, the algorithm scales almost linearly with the number of atoms (Figure 6.20).

Furthermore, the performance was compared to the results of Maňák et al. [161], who are the only ones presenting timings for molecules of the PDB. As for the results of Maňák et al., an Intel Core 2 QUAD 2.4 GHz system was used for the computation. The computation times are approximately two times faster in contrast to their approach on a single core. For example, for the protein *pdb:1HX6* it takes 5.8 seconds in contrast to 11.2 seconds with Maňák's algorithm.

PDB-ID	#Atoms	VD		BF			r_p	P
		1	6	1	6	GPU		
2OAU	13 573	4.2	1.1	10.4	1.9	0.2	1.0	0.3
1GKI	20 150	6.3	1.5	13.6	2.4	0.3	1.4	0.3
1G3I	46 040	15.9	4.0	36.2	6.4	0.7	2.0	0.4
1AON	58 870	19.4	5.2	46.9	8.6	0.8	3.0	0.3
1JJ2	98 543	38.8	9.3	82.9	15.2	1.4	1.4	1.2

System: Intel Xeon X5650 E5540 2.66 GHz, NVIDIA Geforce GTX470.

Table 6.1: *Computation times in seconds of the Voronoi diagram (VD), the boundary filter (BF) and the cavity and path computation (P). The VD and BF were computed with 1 and 6 cores using OpenMP. Additionally, the BF was computed on the GPU using OpenCL*

6.3.2 Path Filtering and Cavity Computation

The boundary filter depends on the number of rays and the hit ratio (Section 6.2.1). For all tests, 100 rays and a hit ratio of 0.5 were used. The timings for one and six CPU cores as well as for the GPU implementation are given in Table 6.1. The timings show that the boundary filter scales linearly with the number of atoms. On the CPU it is the most expensive part in the computation pipeline. The speedup for six cores is around 5.5 and for the GPU it lies between $50 - 60$ with respect to a single CPU core.

After applying the boundary filter, the probe filter is applied with the user-defined radius r_p (Section 6.2.2). Then all cavities are extracted as described in Section 6.2.3). Finally, the most significant paths are filtered using the cycle and branch filter (Section 6.2.4). The timings for all these steps are summarized in the last column of Table 6.1. For the branch filter, a further parameter is required to steer the reduction. In all tests the relative distance between the smallest and largest vertex was used with a threshold parameter of 0.95. Note that the most expensive part of these steps is the cycle filter. The speed of the filtering and cavity computation depends mainly on the number of atoms, but also on the number of elements in the graph after applying the boundary filter. Nevertheless, in contrast to the previous version [149], it is so fast that the computation time is nearly negligible. Overall, the full filtering pipeline is done within a few seconds, even for large molecules. As comparison, CHUNNEL [44] requires several hours to detect all channels in molecules with only a few thousand atoms.

6.4 Discussion

In this chapter, the computation and filtering of potential molecular paths and cavities based on the geometry of the van der Waals spheres of the atoms was described.

By computing the molecular paths from the Voronoi diagram of spheres, it becomes possible to extract the paths from the exact distance transform of the molecule in contrast to others [188, 163, 246], who use approximations. The Voronoi diagram computation, as described in this chapter, is about two times faster than the optimized algorithm presented by Maňák et al. [161]. One reason might be the overhead by computing the Delaunay complex for a fast neighbor search instead of using a grid data structure. The algorithm by Medvedev et al. [164] was also tested. However, their algorithm is restricted to data with periodic boundary, which allows them to use a simpler edge direction detection, which fails for typical proteins. After several minutes, the program still computes new Voronoi vertices although the number of maximal elements is already reached. This leads to the assumption that the algorithm runs in an infinite loop.

From the Voronoi diagram of spheres, the most significant paths and cavities are computed by applying a sequence of methods. Among these, the boundary filter is the most expensive one. There are several alternatives to the presented approach. For example, one could cut all vertices v of the Voronoi diagram with $d(v)$ greater than a given value. This filter is quite similar to a boundary definition by a large probe sphere that rolls over the atoms to create an envelope. Such a filtering would be very fast, but it would also eliminate paths inside large cavities. Another alternative is to use a mask approximating the structure of the molecule. However, creating such a mask would involve user interaction and might be time consuming for complex molecules. The presented boundary filter seems to be a good compromise between fast computation and good boundary estimation. Furthermore, the hit ratio will be computed only once and is stored. The fixed ratio of 0.5 in the tests, cuts the paths approximately at the convex hull, like in many other molecular path detection algorithms. A cut with the convex hull can be done much faster, but with the ambient occlusion approach the user can change the ratio to keep for example only paths deep inside the molecule. Empirically, 100 rays seem enough to cover the domain around a vertex. The implementation for graphics cards greatly accelerates this method.

To the best of the author's knowledge, the presented approach was the first that detects *all* maximal paths through tunnels as well as into cavities. For example, CHUNNEL [44] detects only paths through tunnels. Additionally, the method described here is faster by at least two orders of magnitude. Finally, for several examples, it could be shown that the filtering keeps only the significant paths, which contained also the main paths extracted by others [32, 44, 188, 189].

7 Cavity Analysis

The analysis of cavity structures is as least as important as its computation. The analysis comprises measurements, like the volume or area, as well as visualization in order to analyze the location and shape of the cavities. Both topics will be investigated in this chapter. An example of a result is depicted in Figure 7.2. Most of the techniques were described in "*Voronoi-Based Extraction and Visualization of Molecular Paths*" [149], "*Dynamic Channels in Biomolecular Systems: Path Analysis and Visualization*" [147], and in "*Exploration of Cavity Dynamics in Biomolecular Systems*" [148]. The methods are applied to the cavity description proposed in the previous chapter.

Recall that the skeleton of each cavity is a connected path component of the filtered Voronoi graph of the atom spheres. The graph consists of Voronoi vertices and edges. Each cavity will be represented by a finite set of empty spheres. The positions of these spheres lie on the Voronoi edges of the path component and their radii are maximal such that they do not intersect the atom spheres (Section 6.2.3).

7.1 Measurements

The analysis of cavities involves computing their volumes and areas. A correct computation of the area and volume of the union of a set of spheres, where the spheres can intersect each other, was described, for example, by Gibson et al. [72] and Petitjean et al. [187]. Depending on the input spheres, these algorithms are often complex, difficult to implement, and numerically unstable. Hence, an approximation of the volume and area using a discretization of $\mathbb{R}^3$ is usually preferred in practice. In the following, several discrete solutions for

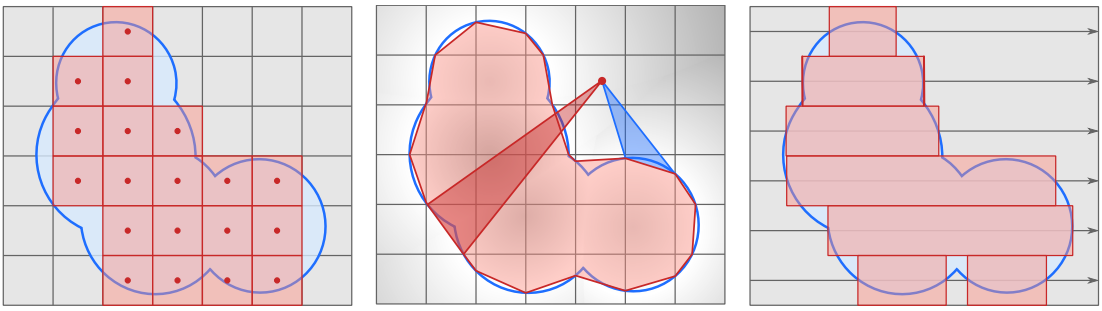


Figure 7.1: *Concept of three algorithms to discretize the volume of a cavity given by a finite set of spheres. Left: Counting the voxels the centers of which lie inside the cavity. Middle: Distance approximation and triangulating of the cavity using marching cubes [154]. Right: Ray casting to approximate boxes inside the cavity.*

the volume and area estimation of general surfaces are investigated for their application to molecular cavities. For all techniques, let $c_{p_j} \in \mathbb{R}^3$ be the positions and $c_{r_j} \in \mathbb{R}$ be the radii of the empty spheres representing a cavity, with $j \in \{1, ..., m\}$. Furthermore, let $a \in \mathbb{R}$ be a sampling width, which is used to discretize $\mathbb{R}^3$ in each direction.

7.1.1 Volume

For the computation of the volume of a cavity, three methods are described in this section.

- **Volume by Voxels.** The axis-aligned bounding box of the cavity is sampled in all directions by a, which divides the box into small cubes with side length a, called voxels. The volume of the cavity is defined as the sum of the volumes of all voxels, the centers of which lie inside at least one empty sphere of the cavity. This is the simplest technique to approximate the volume of a cavity (Figure 7.1, left).

- **Volume by Triangulation.** Again, the bounding box is sampled in all directions with an edge length a. Then, for each sample position $p_{x,y,z} \in \mathbb{R}^3$ the distance $d_{x,y,z} \in \mathbb{R}$ to the cavity is approximated by

$$d_{x,y,z} = \min_{j \in M} \left\| p_{x,y,z} - c_{p_j} \right\| - c_{r_j}$$

  This creates a discrete distance field of the cavity. Using the marching cubes algorithm [154], a closed triangulated surface of the cavity is extracted from the distance field. The volume of a closed triangulated surface can be computed in the following way. First, compute the volumes of all tetrahedra given by the triangles of the surface and an arbitrary but fixed point. In case the dot product of the triangle normal and a vector from the fixed point to a triangle vertex is negative, also the sign of the corresponding volume of the tetrahedron needs to be inverted.

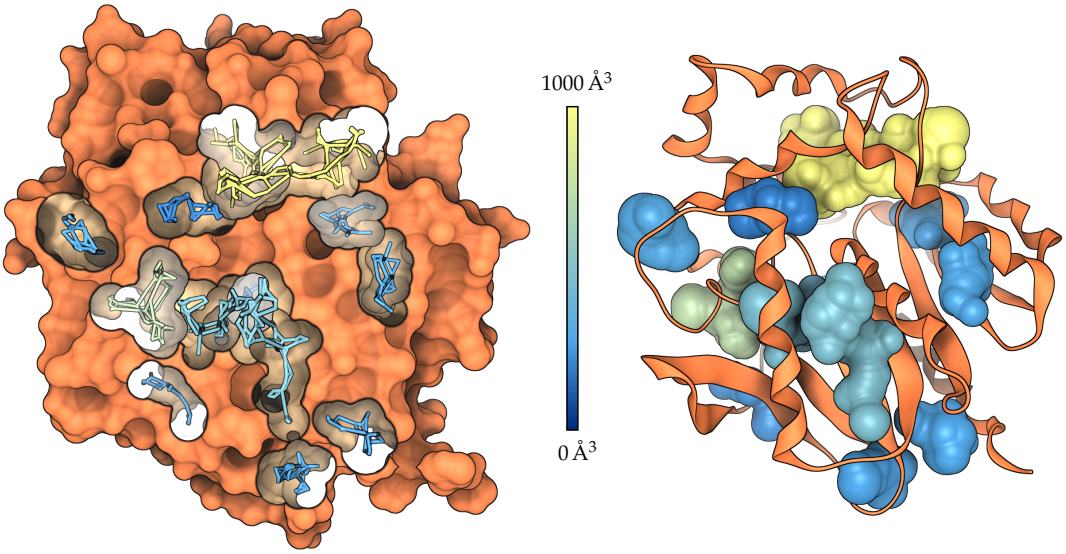


Figure 7.2: *Cavity structure inside a dehalogenase mutant from Rhodococcus rhodochrous, pdb:4WCV. Left: Path components with clipped molecular surface. Right: Cavities with secondary structure. The path components and cavities are colored according to the cavity volume.*

Finally, the overall volume is the sum of all tetrahedra volumes. Note that the result is negative if the fixed point lies inside the cavity. The approach is illustrated in Figure 7.1 (middle).

- **Volume by Ray Casting.** The technique, as presented here, is based on the method by Phillips et al. [192]. It describes the measurement of volumes and areas for molecular surfaces. However, the general concept is a classical discrete integration of the molecular surface volume. In contrast to the previous two techniques, only one side of the bounding box is sampled by a, for example the minimal $xy-$side. Then from each sample position a ray is cast into the remaining direction, which is the $z-$direction in this case. For each ray, all intersection points with the cavity spheres are computed and sorted by their order of appearance. From these intersection points all pairs of points are computed, where the ray enters and leaves the cavity. Each pair corresponds to an axis-aligned box whose width is given by the distance of the points and whose remaining side lengths are a. The overall volume is the sum of all box volumes. An illustration of the method is shown in Figure 7.1 (right).

7.1.2 Area

In contrast to the volume computation, it is more difficult to compute a good approximation of the surface area. For example, it is not possible to modify the voxel approach by summing up the areas of the outer voxel faces. The

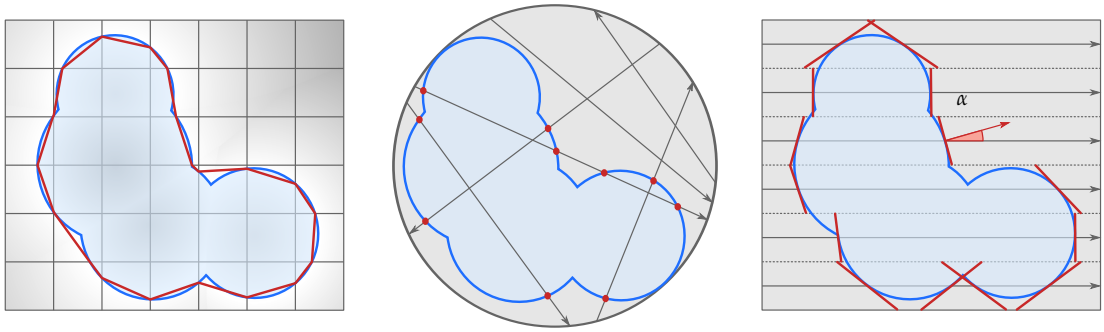


Figure 7.3: *Concept of three algorithms to discretize the ares of a cavity given by a finite set of spheres. Left: Distance approximation and triangulating of the cavity using marching cubes [154]. Middle: Counting intersection points of uniformly distributed rays within a bounding sphere. Right: Ray casting to approximate quadrangles tangent to the surface which approximate the area.*

approach would not converge to the correct result. Consider, for example, a squared diagonal plane and its discrete voxel approximation given by stairs. While increasing the resolution would lead to finer stairs, that are closer to the diagonal plane, the surface area would not change. Even the application of the algorithm based on marching cubes does not necessarily converge to the correct value [115]. However, it gives a much better approximation than the voxel approach. For many years people have been working on efficient algorithms that are easy to implement in order to compute surface areas by discretization. Some common approaches are summarized by Brimkov et al. [29].

In this chapter, three of these algorithms are investigated. The first is quite similar to the triangulation-based method for the volume computation. The second and third are based on ray casting, where one method is quite similar to the ray casting for the volume computation.

- **Area by Triangulation.** The triangulation-based method is equal to the volume approach, except that the volume computation of the tetrahedra is replaced by the computation of the areas of the surface triangles. The rest of the algorithm is identical. An illustration is given in Figure 7.3 (left).

- **Area by Ray Casting and Surface Angles.** Phillips et al. [192] described a modification of their volume computation approach to approximate the surface area. As described above, first the entry and exit points of the ray with the cavity are computed. At each of these points the area is approximated by the square with side length a scaled by $\left|cos(\alpha)^{-1}\right|$, where α is the angle between the ray direction and the surface normal at the intersection point. Note that this approximation is not stable in case the two vectors are nearly orthogonal and even undefined in case of orthogonality. For this reason it is necessary to bound the range for $cos(\alpha)$. The algorithm is illustrated in Figure 7.3 (right).

- **Area by Ray Casting and Counting Intersections.** The original algorithm was proposed by Li et al. [142] for general surfaces and later parallelized by Juba and Varshney [101] for the computation of molecular surface areas. The main idea of the algorithm is to apply the Cauchy-Crofton formula in order to estimate the area of a surface. Briefly, the surface area of a cavity is estimated by counting the intersection points of uniformly distributed lines with the cavity and with a bounding sphere. The area $A_c \in \mathbb{R}$ is then given by

$$A_c \approx \frac{k_c}{k_s} \cdot A_s \ ,$$

where k_s is the number of intersections with the bounding sphere, k_c is the number of intersections with the cavity, and A_s is the surface area of the bounding sphere. By increasing the number of lines, A_c converges to the correct surface area.

In practice, first a tight bounding sphere of the cavity is computed. The smaller this sphere, the less lines are required. In order to keep the algorithm simple and stable, one can use the center of the axis aligned bounding box of the cavity and compute the radius accordingly. While this sphere does not represent the minimal bounding sphere, it is usually still small enough. Then, a set of uniformly distributed lines is computed. Recall that lines that do not intersect the bounding sphere and thus also not the cavity are useless. For this reasons, the lines are computed by sampling pairs of points on the sphere. Hence k_s is twice the number of lines. Uniformly distributed points $p_s \in \mathbb{R}^3$ on a sphere with radius $r_s \in \mathbb{R}$ can be sampled by

$$p_s = r_s \cdot \begin{bmatrix} \sqrt{1-u^2}cos\alpha \\ \sqrt{1-u^2}sin\alpha \\ u \end{bmatrix}, u \in [-1,1], \alpha \in [0, 2\pi).$$

Instead of using default pseudo random values for u and α, low-discrepancy sequences, like the one by Niederreiter [169], generate better distributions of samples. Finally, the number of intersections of the lines with the cavity surface are computed.

7.1.3 Implementation

All algorithms were implemented for the CPU with OpenMP for parallelization. Additionally, the voxel-based approach and the triangulation-based algorithms were also implemented for the GPU using OpenCL. The main acceleration structure for all algorithms is a grid that stores the empty spheres that represent the cavity.

Grid

The grid consists of cubical cells and stores a sphere inside a cell if the sphere intersects the cell (Section 2.5). For the voxel-based volume computation, the grid is used to quickly check if the voxel center lies inside a sphere. To do so, the grid cell in which the voxel center lies is computed and the distances of the center to all spheres, stored in the grid cell, are computed. In case a distance is negative, the volume of the voxel is added to the overall volume and the algorithm immediately continues with the next voxel. Several practical experiments showed that a cell size of approximately $5 \cdot a$ gives a good performance on CPU and a cell size of $15 \cdot a$ performs well for the GPU, where a is the sampling width for the voxels.

For the triangulation-based algorithms, the grid is used to quickly compute the approximation of the distance function. Briefly, the marching cubes algorithm iterates over all cubes with side length a, given by the sample positions. In case the distance values of the 8 vertices of a cube are positive and negative, the surface must intersect the cube and the algorithm creates triangles that represent the intersection. For cubes whose vertices have only positive distances the algorithm assumes that the cube lies completely outside the surface and, thus, will not create any triangles. Similar, for cubes with pure negative distances the algorithm assumes that the cubes lie completely inside the surface. For this reason it is not necessary to correctly compute all distance values of all samples. Only the samples of cubes with positive and negative distances need to be correct. For the rest, it is sufficient that the sign of the samples is correct. Hence, for each sample position, the grid cells are detected that intersect the cube, the center of which is the sample position and the side length is $2 \cdot a$. Then, the minimal distance of the sample position to all spheres in these cells is computed. Since the side length of a grid cell is usually larger than $2 \cdot a$, at most 8 grid cells need to be investigated. If all these grid cells are empty the distance at the sample position is set to a. Similar to the voxel-based approach, a cell size of $5 \cdot a$ performs good for the CPU version and $15 \cdot a$ works well for the GPU version.

The approaches based on ray casting use a grid to quickly compute the intersection points of the rays with the cavity spheres. For this task, the ray voxel traversal by Amanatides [2] is applied, which was already successfully used for the fast ray casting of biological structures (Section 4.2.1). Briefly, the algorithm iterates over the grid cells that intersect with the ray in the order of their intersection. For each cell, the intersections of the ray with the spheres, stored in the cell, are computed. In contrast to the GPU version used for rendering (Section 4.2.1), the volume computation on the CPU uses a bit field to avoid multiple intersection computations with the same sphere. For this algorithm a cell size of $16 \cdot a$ results in a good performance for protein cavities.

OpenMP

The voxel-based approach consists of a single parallelized loop over all centers of voxels. In each iteration, the 3-dimensional position of the voxel center is calculated. Then, the corresponding grid cell is detected and the distance computation with the spheres is performed. In case the center lies inside a sphere, the volume of the voxel is considered for the overall volume.

The triangulation-based method consists of two steps. In the first step the discrete distance function is computed. Similar to the voxel-based technique, a single parallelized loop over all samples is executed. In each iteration, the 3-dimensional position of the sample is computed, and the distance to the spheres is approximated using the grid data structure. Note that this algorithm requires memory for the 3-dimensional discrete distance function. In the second step, a modified marching cubes algorithm is applied. Again, a single parallelized loop over all sample cubes is executed. Depending on the distance values at the cube vertices, the surface triangles inside the cube are computed. Since the triangles can be handled independently of each other for the volume and area computation, the volume of the tetrahedrons or the triangle areas are computed immediately and the triangles are discarded afterwards. Note that the actual volume or area computation is nearly negligible in contrast to the computation of the distance function and the triangulation. For this reason, it is advisable to compute directly both quantities in a single run.

For the ray casting algorithms, a single parallelized loop over all rays is executed. In each iteration, the intersection points of the ray with the spheres are computed and sorted using the introsort algorithm [168]. Finally, the pairs of intersections where the ray enters and leaves the cavity are detected and the volume or the area is computed accordingly.

To avoid a synchronization of the threads, in all algorithms each thread has its own volume or area variable. Finally, the sum of all these sub-volumes or sub-areas is computed in a separate loop.

OpenCL

In contrast to the CPU version of the voxel-based algorithm, for the GPU version a thread is started for each voxel center in the x- and y-direction. Then, each thread iterates over the remaining z-direction and adds up the volume of all voxels, the centers of which lie inside a cavity sphere. Finally, the sum of the sub-volumes in x- and y-direction is computed on the CPU.

The triangulation-based algorithm is divided into two separate OpenCL kernels. The first computes the distance function and the second implements the modified Marching cubes algorithm. Similar to the voxel-based algorithm, for both kernels a thread for each sample position or cube in the x- and y-direction is started and each thread iterates over the z-direction.

For both algorithms the grid of the spheres is represented by three arrays as described in Section 2.5, and a sphere is stored into a grid cell if it intersects the cell.

7.2 Visualization

One of the most challenging problems in 3-dimensional visualization is to render features surrounded by their corresponding geometry. Although the molecular paths can be visualized by illustrating the filtered Voronoi edges, it is difficult to analyze them without seeing the surrounding molecular structure. In this section several techniques are proposed to show molecular paths or cavities together with an adequate molecular visualization.

7.2.1 Paths

Molecular paths can be visualized by rendering the filtered path components or the extracted significant paths as 3-dimensional tube-like structures. Therefore, the paths are converted into piece-wise linear curves. Each linear segment is rendered as a cylinder that is cut by the angle bisector planes to the next segments at the ends. Thus, it fits perfectly together with the neighboring segments. In case of a very acute angle between two segments, one can cut the segments with the orthogonal planes at these ends and place an additional sphere at this position. One can also use such spheres for the ends of the paths.

To quickly generate and visualize this 3-dimensional geometry for the paths, again ray casting is used. The technique is the same as described in Section 3.2 for the ray casting of smooth molecular surfaces. Briefly, for each linear piece of the path a vertex is generated that stores the properties of the cylinder segment. In the geometry shader, the vertex is expanded to a quadrangle that encloses the cylinder segment after projection. In the fragment shader, the actual ray casting is performed. A better visual tracing of the paths can be achieved by smoothing the paths in a pre-processing step. Examples of this kind of path visualization are shown in Figures 7.4 and 7.6. An extension is the use of cones instead of cylinders to change the thickness of the paths according to their distance to the surrounding atoms or other properties. Furthermore, the paths can be color-coded to show the distance to the surrounding atoms or to visualize other properties. In Figure 7.2, the paths are colored according to the volume of the corresponding cavities and in Figure 6.15 the distance of the paths to the atoms is color-coded.

7.2.2 Cavities

Recall that the cavities are geometrically approximated by a well selected finite subset of corresponding empty spheres. But instead of rendering only

a set of spheres, a better visualization with a geometrically more accurate shape of the cavity is achieved by using the skin surface of the empty spheres (Section 2.1.4). The skin surface helps to compensate the discretization of the cavity by a set of finite spheres. However, one needs to be careful with the selection of the shrink factor. Too small a shrink factor leads to an overestimation of the size of the cavity, which results in intersections of the skin surface with the surrounding atoms. Practical investigations have shown that a shrink factor around 0.7 avoids such intersections and creates a smooth surface of the cavity. The analytical description as well as a fast computation and visualization of the molecular skin surface was given in Chapter 3. These techniques can also be used for the visualization of the skin surface of the cavities. Finally, the surface can be colored according to local features like the size of the closest empty sphere.

7.2.3 Illumination

Illumination plays an important role for shape and depth perception of 3-dimensional objects. Although direct illumination is a fast way to emphasize the 3-dimensional shape, it does not provide a good depth perception, which is of particular interest for cavity structures. To enhance the correlation between the computed paths and the surrounding molecular structure, a novel technique is presented here. The main idea of this technique is to place many small point lights along the computed molecular paths to draw the user's attention to the channels and cavities reached by the paths. Results of this technique are depicted in Figure 7.4.

In order to guarantee a high visual quality as well as interactive speed for many light sources, deferred shading [207, 85] is used. The technique was developed to reduce the costs of illumination to the number of pixels in the final image instead of lighting all fragments of the scene including hidden ones. This is realized by rendering the molecular structure into multiple render targets, represented by a set of textures. During this first rendering the fragments are not illuminated, but their color, normal, and depth is stored in three target textures. In the following rendering passes, only a screen-space filling quad is rendered. During these passes, the previous textures are used as input and the illumination is computed only for the visible fragments. Additional passes can be implemented, for example, to add ambient occlusion. The final pass computes the direct illumination by a global directional light as well as the illuminations by the point lights placed along the paths. The illumination passes are described in more detail in the following.

Ambient Occlusion

Ambient occlusion is a technique to approximate global illumination. The goal is to measure the possible incoming ambient light for all points in the scene. Points that lie in a cavity often receive less ambient light, because the

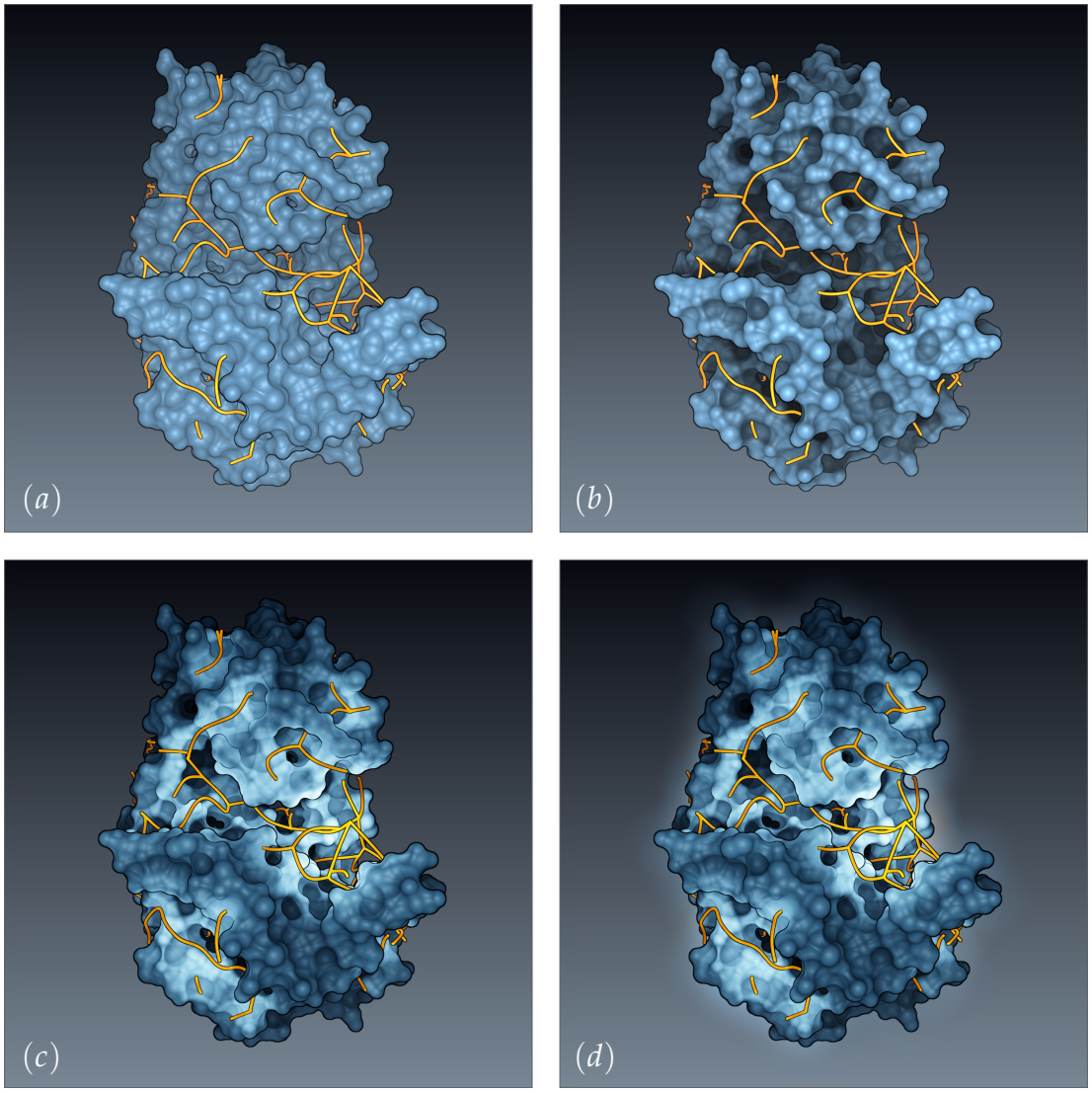


Figure 7.4: *Visualization of the illumination pipeline for the most significant paths in Porcin pepsin (pdb: 5PEP). Direct illumination with depth cueing (a), additional screen space ambient occlusion (b), added point lights along the paths with HDR (c), and an extra glow effect (d).*

light is reflected by occluding geometry. Showing this effect leads to a much better depth perception. Because it is such a relevant technique, in many computer graphical fields a lot of work was done in the past years especially for real-time rendering of dynamic scenes, which plays an important role for the game industry. Similar to the first definition by Zhukov et al. [251], ambient occlusion is often modeled for a point $p \in \mathbb{R}^3$ on a surface with normal $\vec{n} \in \mathbb{R}^3$ as

$$a(p, \vec{n}) = \frac{1}{\pi} \int_{\omega \in \Omega} v(p, \omega) \, |\omega \cdot \vec{n}| \, d\omega,$$

where ω is a direction given by the hemisphere Ω in the point p with rotational axis $\vec{n}$. Furthermore, v is a visibility function which returns 1 if p

is not occluded from direction ω and otherwise it returns 0. Typically Ω is discretized by a fix number of uniformly distributed rays. If a ray hits the surrounding geometry, the ambient light is blocked in this direction, and v returns 0. Over the years mainly two groups of approaches were proposed to achieve ambient occlusion at interactive frame rates.

The first group is a post-processing technique, called screen-space ambient occlusion (SSAO). The probably most simple variant of such an algorithm is depth darkening [157], which was briefly described in Section 3.2.4. More realistic approaches sample the surrounding depth values of a fragment to get an approximation of the occluding geometry in the hemisphere. Because most approximations lead to noisy results, a blurring filter is applied afterwards. More information about these techniques as well as implementation details can be found, for example, in the works by Mittring [167], Bavoille et al. [15], and McGuire et al. [162]. An example of the implementation of the algorithm by Bavoille et al. can be seen in Figure 7.4. Due to the computation of the ambient occlusion in screen-space, the technique has usually a constant upper bound for the time complexity per frame and can be implemented completely on the GPU. Thus, ambient occlusion can be achieved interactively for fully dynamic scenes and is state of the art in most 3-dimensional computer games. Nevertheless, depending on the quality and resolution, screen-space ambient occlusion can still decrease the frame rate. Furthermore, due to the limited information around the surrounding geometry it can create incorrect results in certain situations.

In addition to the screen-space versions, there exists a group of algorithms that work in object-space. Thus it is called object space ambient occlusion (OSAO). The most common techniques pre-compute ambient occlusion for the objects in the scene and store the result in textures. These textures can then be used directly during the rendering. For the computation, the ambient light directions are discretized on the hemisphere and intersection tests for each ray are performed with the surrounding geometry. Depending on the number of rays and resolutions for the textures, the ambient occlusion is often more realistic and creates better results than the SSAO variants. However, the pre-computation requires a lot of time and memory as well as a parametrization of the objects for the texture mapping. This makes it impractical for dynamic scenes. One example, is the approach presented by Tarini et al. [224] for the van der Waals surface and the ball-and-stick representation of molecules. Other approaches create ambient occlusion volumes. Grottel et al. [78] presented such an approach to approximate ambient occlusion for molecular dynamics data. Instead of analyzing the occlusion by surrounding geometry, the density of the geometry is directly used as indicator of occlusion. Consider therefore a regular grid representing a 3-dimensional scalar field. At each grid point, the surrounding density of the geometry is approximated. The density grid is stored in a 3-dimensional texture. During the rendering, the ambient occlusion in a point is calculated by the density into the direction of the normal. While a high density indicates to a high ambient

occlusion, for a low density the point probably receives much more ambient light. The technique is applicable even for dynamic molecular data. However, the result and quality depends a lot on the resolution of the density grid and it is difficult to implement the technique for arbitrary geometries.

Path Lighting

While ambient occlusion improves the depth perception and allows quickly identifying cavities at the outer surface, it is still difficult to distinguish between the accessible cavities and all existing cavities. Therefore, a new technique is applied, which places many point lights along the detected molecular paths. A point light is defined by its position and a radius. Furthermore, the strength of the light fades out linearly from the center to its radius. At each remaining Voronoi vertex $v \in \mathbb{R}^3$ of a path component, a point light is placed. The radius of the point light is set depending on the distance $d(v)$ from the vertex v to the atom spheres. To take effect, the radius needs to be larger than $d(v)$, otherwise the light would not reach the molecular structure. One suitable selection is $\min(2 \cdot d(v), 4 \cdot r_p)$, where r_p is the radius of the probe.

The described placement can generate several thousand point lights for an average-sized protein. Even with classical deferred shading, it is not possible to render scenes with so many lights interactively. However, due to the bounded lighting size it is not necessary to consider all point lights for each fragment in the scene. Only light sources closer to the fragment than their radius can illuminate it. For a fast detection of the lights, again a 3-dimensional grid data structure is used (Section 2.5). The grid stores a light into a cell if the light position lies inside the cell. In addition to the texture buffer object with the light positions and radii, a further buffer object can be used to store for each light a color.

During the deferred shading, for each fragment all point lights within a radius of $4 \cdot r_p$ are detected. Each light creates a simple diffuse illumination of the fragment using Lambertian reflectance. Additionally, a global weak directional head light is applied to avoid darkness at positions without paths. Working with so many lights often creates white artificial areas due to the color clamping. To solve this problem, high dynamic range (HDR) rendering is used. Instead of color components between $[0, 1]$, HDR allows using values in the range of $[0, \infty)$. In a further rendering pass, a simple tone mapping function is applied to map these values back to $[0, 1]$. Together with a gamma correction, a much better illumination is obtained (Figure 7.4).

In order to make the whole illumination model more intuitive for the user, a glow effect is added at the boundary of the molecular structure. This creates an impression of lighted cavities like the active chambers in a mine. The effect is realized by collecting for each boundary fragment all neighboring colors in a certain radius. The new fragment color is then set to the average of the collected colors. Note that for large radii, the glow effect can be expensive to compute due to the large number of texture fetches. One possibility to reduce

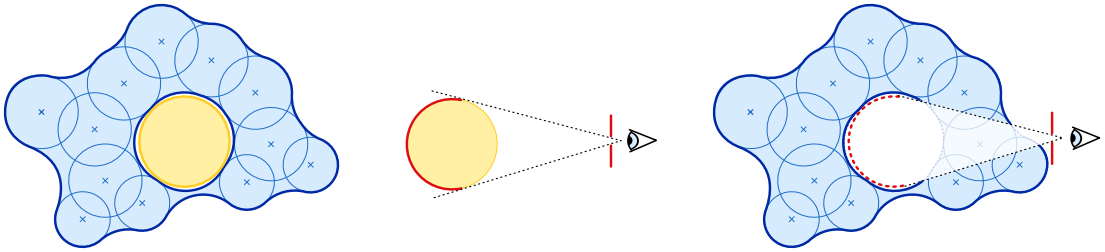


Figure 7.5: *Illustration of the clipping technique. Left: Cross-section of the molecular surface (blue) and the cavity (yellow). Middel: Depth rendering (red) of the cavity. Right: Clipping of the molecular surface by the cavity depth.*

this amount for large radii is to use the same strategy as for screen-space ambient occlusion. Only a small set of randomly selected sample points is selected. These samples shall be uniformly distributed in the neighborhood of the fragment. An additional smoothing filter reduces noise artifacts in the result (Figure 7.4).

7.2.4 Clipping

The methods presented so far enable the user to get a good overview of the interesting cavities and paths. But if the molecule is very complex, it becomes difficult to follow the paths and cavities into buried regions, even if the secondary structure or the ball-and-stick representation is used. To solve this problem, a simple but effective clipping method is presented. The molecular structure, in particular the molecular surface, is clipped using the surface of the selected cavity. A similar technique was previously used by Jones et al. [100] for visualizing flow trajectories together with volume data.

Since there are several possibilities to realize such a clipping, the developed method is presented here in detail. The first step is to render the surface of the cavity into a texture without storing colors and normals or computing illumination, because only the depth values are of interest. In contrast to the default rendering, where the closest depth values to the camera are stored, the depth buffer is cleared with 0 distance and stores the farthest depth values into a texture. During rendering of the molecular structure, this texture is used to discard all fragments closer to the camera than the value given by the texture. The algorithm is illustrated in Figure 7.5.

Two additional features help to distinguish between the area behind the cut and the rest of the structure. The first is a boundary around the cut and the second is a brightness change behind the cut. To compute the boundary, for all fragments with a depth value of 0 in the texture, the surrounding depth values within a given radius are checked. The ratio r_{col} between the number of depth values equal to 0 and the overall number of values is used to color the cutting boundary. The final pixel color c_{pix} is determined by $c_{pix} = r_{col} \cdot c_{mol} + (1 - r_{col}) \cdot c_{bound}$, where c_{mol} is the color of the molecular structure and c_{bound} is the boundary color. The result can be seen, for example,

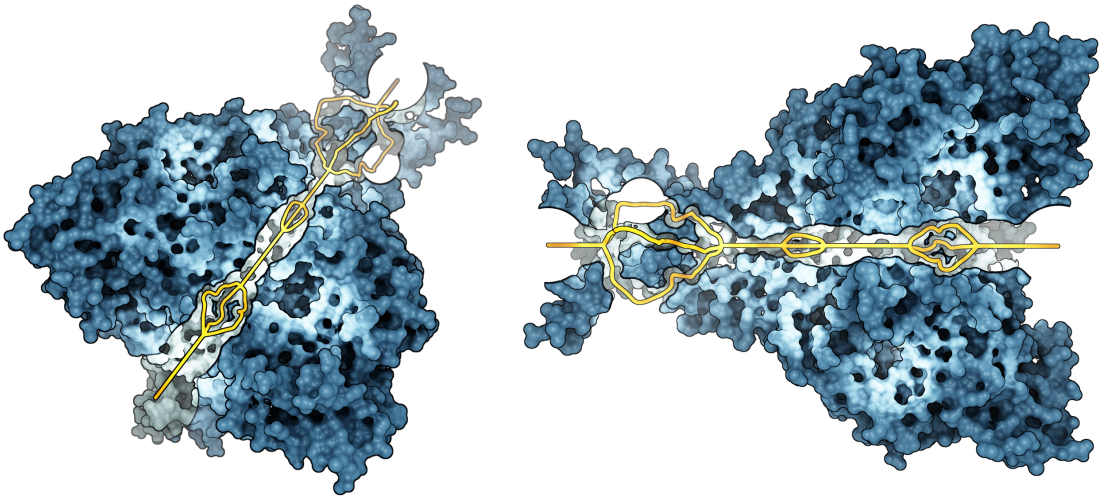


Figure 7.6: *Sodium ion channel (pdb: 3HGC). The channel is important for epidermal ion transport into the cell. Blocking the channel by amiloride is a way to treat cystic fibrosis. The molecular surface is clipped in order to see the internal structure of the channel.*

in Figure 7.6, where the boundary appears to be black. For the second feature, the brightness of all fragments behind the cut is changed by a user-defined value. In Figure 7.6, the pixel values behind the cut have been brightened.

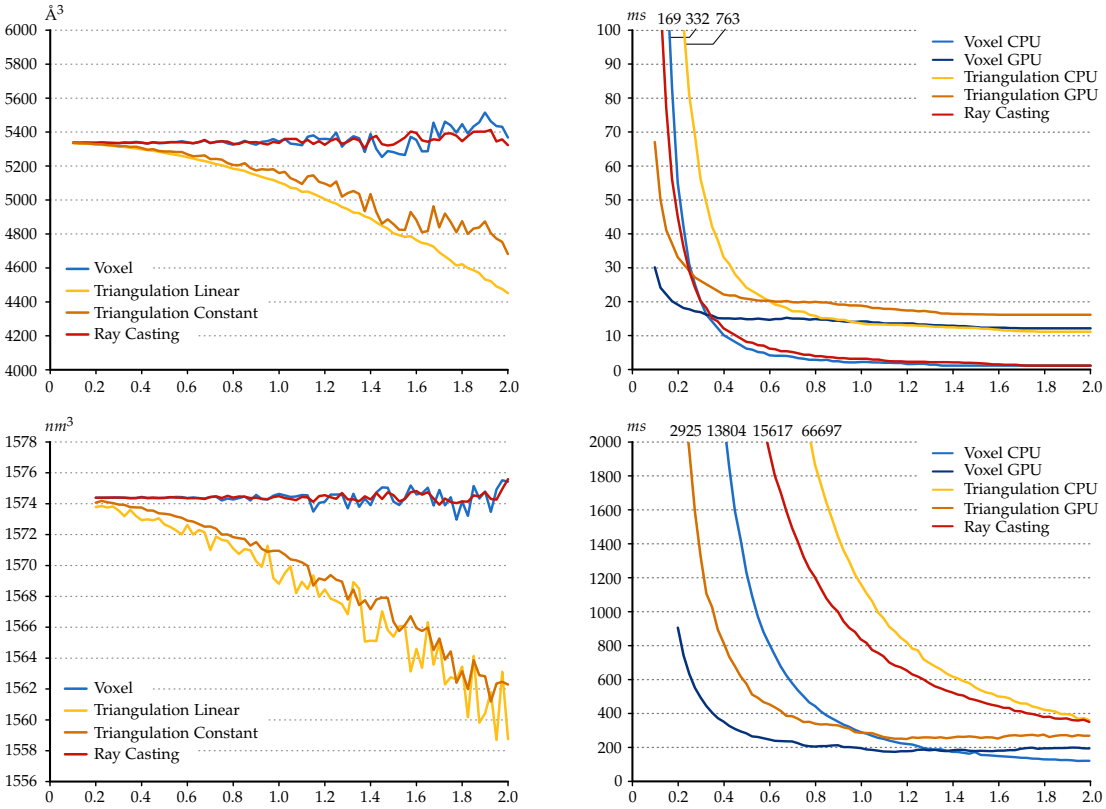
7.3 Results

This section contains some experimental results for the cavity measurement algorithms and the visualization techniques. Especially for the measurement algorithms, this helps to select the most suitable algorithm depending on the underlying data and requirements. Furthermore, it will be shown that the visualization techniques are interactive, even for large proteins.

7.3.1 Measurements

For the evaluation of the performance and accuracy of the volume and area computation methods, the cavities of several molecules were investigated that differ in both size and shape. The results for two representative molecules, *pdb: 3MTH* and *pdb: 1AON*, are summarized in Figure 7.7 and 7.8. The diagrams show the sum of the cavity volumes and areas as well as the computation times depending on the sampling width a. While 3MTH has a common protein size with cavities, the size of which ranges from several hundred Å^3 to a few thousand Å^3, 1AON is much bigger. It has a huge cavity structure located in the center with a volume of approximately 1 574 nm^3, the shape of which has many small features.

The most important result is that the voxel-based volume computation and the ray casting-based technique converge much faster than the triangulation-based algorithm. The latter starts quite fast to underestimate the volume



System: Intel Xeon X5650 E5540 2.66 GHz, NVIDIA Geforce GTX 680

Figure 7.7: *Cavity volumes (left column) and their computation time (right column) depending on the grid resolution in Å for three algorithms and two proteins: pdb:3MTH (top row) and pdb:1AON (bottom row).*

with increasing sampling width. This is even worse if linear interpolation is used for the Marching cubes algorithm to compute the vertices of the surface. Additionally, the ray casting-based approach converges slightly better than the voxel-based technique. To achieve a similar quality for the triangulation-based algorithms than for the other methods, the sampling width needs to be between a quarter and a half of the original sampling width, which can be seen in Figure 7.7.

The convergence of the algorithms for the surface area computation is worse than those for the volume computation techniques. The ray casting algorithm which counts the intersection points with the cavity has the best convergence, when Niederreiter pseudo random sampling is used. Furthermore, it is the only algorithm of the three that converges to the correct result. This is not guaranteed for the other two algorithms. Both, the ray casting algorithm with surface angle measuring and the triangulation with marching cubes underestimate the real surface area in most cases. Especially, the angle measuring technique can never converge to the correct result, because of the clamping of angles close to 90 degrees. However, it converges faster than the triangulation based approach, in particular for large cavities. The smaller the angular bound, the better is the result for fine resolutions but the larger is the error

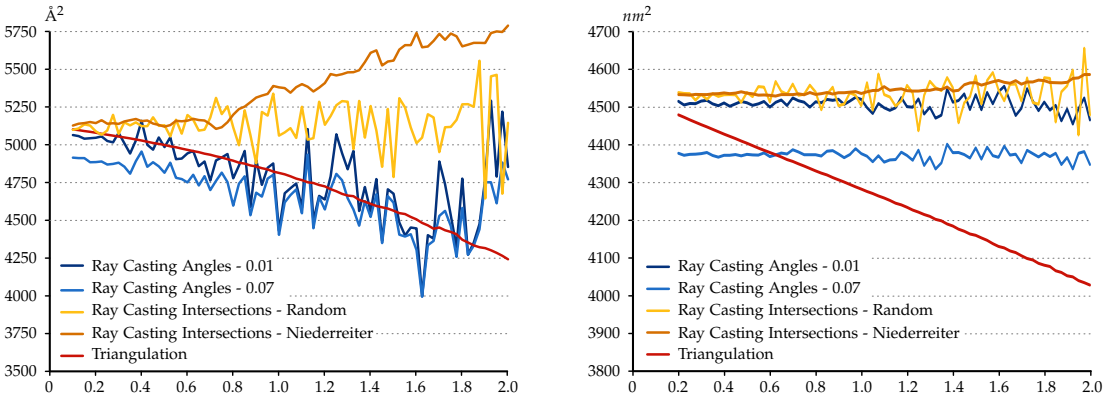


Figure 7.8: *Cavity surface areas for three algorithms and two proteins: pdb: 3MTH (top row) and pdb: 1AON (bottom row).*

for decreasing resolutions. In all cases, a bound of 0.01 for the cosine of the angle performed better than 0.07. For smaller bounds, the resolution needs to be much higher such that it is impractical. In order to compare the ray casting technique which counts the cavity intersections with the grid-based approaches, the same number of rays was used as for the ray-casting-based algorithm with the surface angle measurement.

In general, the voxel-based approach is the fastest technique. The ray casting-based method is a bit slower, especially for large cavities. However, in several cases it performs slightly better for very small sampling widths for mid-size and small cavities. The Marching Cubes-based method performs much worse in comparison to the other two techniques. The GPU implementations in OpenCL of both, the voxel-based method and the Marching Cubes-based approach perform, in general, better than the CPU implementations using OpenMP, especially for large data sets and small sampling widths. With increasing sampling width, the overhead for the data transfer to the GPU and the creation of the GPU specific data structures becomes too big relative to the profit of the highly parallel computation. In such cases the CPU implementations of the voxel-based method and the ray casting-based approach perform better. Note that the ray casting-based algorithms for the surface area computation have the same performance as the ray casting algorithm for the volume computation. Furthermore, the ray casting-based algorithm which measures surface angles can compute the volume at the same time with nearly no overhead. The same is true for the triangulation-based techniques.

7.3.2 Visualization

The visualization of the smooth molecular surfaces SES and MSS was already described in Chapter 3. It provides performance measurements for both surface types. In this section, the focus lies on the performance of the presented illumination techniques.

PDB-ID	#Atoms	FR(%)	DL	#PL	+PL	+SSAO	+Glow
2OAU	13 573	65	61	36 897	27	19	13
1GKI	20 150	65	45	19 655	24	18	12
1G3I	46 040	55	43	21539	20	15	10
1AON	58 870	55	42	10 787	22	15	10
1JJ2	98 543	83	28	101 201	16	12	9

Graphics card: NVIDIA Geforce GTX 470.

Table 7.1: *Rendering performance given in frames per second w.r.t. the given fill rate (FR). First, the SES and the paths were rendered with only one directional head light (DL), then the point lights (PL) were added, the screen space ambient occlusion (SSAO), and finally the glow effect.*

In order to test the performance of the illumination techniques, several molecules of different size were investigated together with their molecular paths. To do so, direct illumination was compared with the advanced illumination by switching on one extension after the other, starting with the screen-space ambient occlusion over path lighting to the glow effect. Note that the molecules and the paths used for this performance tests are the same as presented in Table 6.1. The rendering results were obtained from a fixed image size of 1024×1024. The frame rates for this setup are shown in Table 7.1. The paths were rendered as tube-like structures, see Section 7.2.1, and the molecular structure was shown using the solvent excluded surface (SES). The fastest rendering was achieved with only a single directional head light. By adding the point lights along the paths, the performance decreased to nearly half the frame rate. However, for all tested molecules the visualizations were still interactive, even if screen-space ambient occlusion and the glow effect were added. With the most expensive illumination, still a frame rate of 9 fps was achieved.

In addition to the illumination tests, the performance of the clipping by the cavities was analyzed. Of course, the performance depends mainly on the complexity and size of the selected cavities. Nevertheless, a great performance loss could not be recognized because of the relatively large shrink factor. For example, the clipping of the SES of *pdb:1AON* by the large cavity in the middle decreased the frame rate from 15 to 13 fps.

7.3.3 Application Results

In Figure 7.9, the filtered paths of a dendritic core multi-shell nanotransporter (CMS-NT) are shown. Image (c) shows all extracted paths for a probe radius of 3 Å. A probe of this size approximates a morphine molecule. The goal of the design of this nanotransporter is to carry morphine through the epidermis. To achieve this, morphine needs to bind as deep as possible in the transporter. None of the paths reaches the core of the molecule. Only some

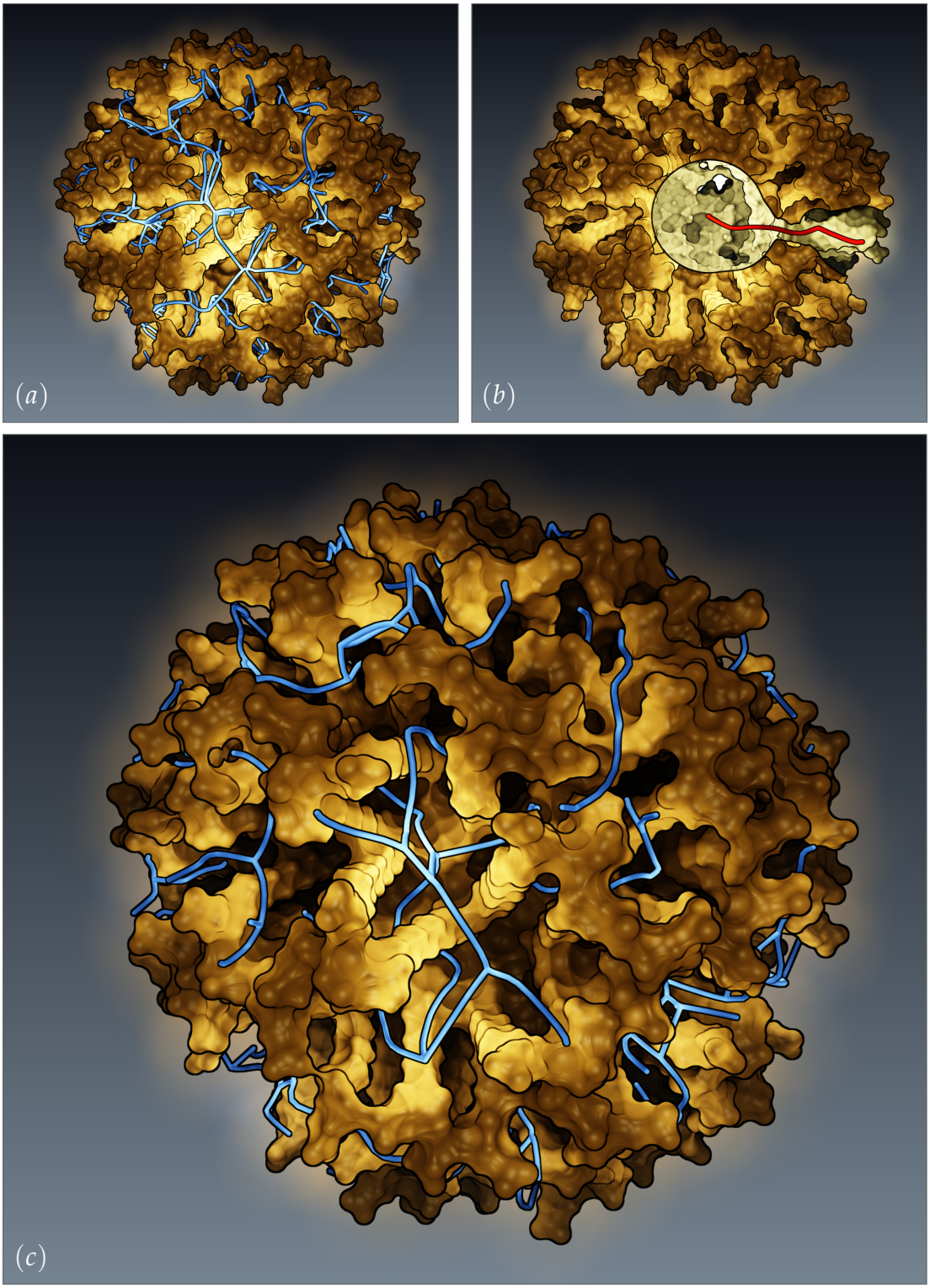


Figure 7.9: *Molecular paths in a CMS nanotransporter for a water probe with a radius of* $1.4\,\mathring{A}$ (a). *These paths can reach the inner core, which is shown by clipping the molecular surface with the cavity of a selected path* (b). *Paths for morphine, which is approximated by a probe radius of* $3.0\,\mathring{A}$ *can reach cavities in the outer shell but not the core* (c).

deep cavities in the outer shell are detected. For comparison, in image (a) all paths are shown for a probe with radius 1.4 Å, which is approximately the size of a water molecule. Several of these paths can reach the inner core of which one is shown by surface clipping in the right image.

Another example is given in Figure 7.6, which shows the feasible sodium channels in *pdb: 3HGC*. These channels are accessible for NA+ ions but can be blocked by amiloride. Note that NA+ ions and amiloride have approximate radii of 1 Å and 2.5 Å, respectively. The image shows the paths extracted using a probe filter of 1 Å. As can be expected from the 3-fold rotational symmetry of 3HGC, the extracted paths also show such a symmetry.

7.4 Discussion

For the computation of the volume and area of the cavities, three algorithms were proposed. In order to get the best mixture of accuracy and performance, the voxel-based approach and the ray-casting based technique, implemented with OpenMP, seem to be the best algorithms for the volume computation of mid-size and small cavities. The experiments showed that usually a sampling width smaller than 0.5 Å does not seem to be necessary for both techniques. However, if high accuracy is desired or for large cavities, the implementation of the voxel-based approach using OpenCL is superior to the other methods. For the surface area, the algorithm with the best accuracy is the ray casting approach that counts the intersections with the cavity. Especially with the Niederreiter sampling, it converges faster than the other techniques. The convergence of the ray casting approach with the angular measurements is only slightly worse, but it does not converge to the correct surface area. It is difficult to estimate the error depending on the angular threshold. On the other hand, a threshold of 0.01 for the cosine of the angle produced good estimations in all tested cases from small to large cavities. The triangulation-based approaches for volume and area measurements performed worst. A sampling width smaller than 0.2 Å is needed to get a similar accuracy as with the other algorithms with 0.5 Å. The triangulation-based approach and the ray casting that measures the surface angle can be slightly modified to compute both, volume and area, with nearly no overhead. Thus the GPU implementation of the triangulation-based approach could be interesting for large molecules.

The bad convergence of the triangulation-based algorithms has mainly two reasons. The first is the approximation in the distance function of the cavity. While this function is correct for points outside the surface, for points inside the surface, the distance is often overestimated. As an example, consider a point that lies in the intersection region of two empty spheres that represent a cavity. The distance of this point to the cavity surface is usually the distance to the intersection circle of the two spheres. But the approximation takes the smaller distance to the surface of one of the spheres, which is typically larger. This overestimation of the distance leads to a smaller triangulated surface by

the Marching Cubes algorithm. However, the sign of the distance is always correct and the distance converges to the correct distance, the closer the point lies to the cavity surface. Computing the correct distance function would be too time consuming and can be numerically difficult. The second reason for the bad accuracy is that most parts of the cavity surface are spherically convex. The Marching Cubes algorithm computes vertices on the cavity surface and triangulates them. The planar triangles always cut off a part in these convex regions. For the other two algorithms the overall volume is given by the sum of the volume of boxes. In some regions, these boxes overestimate the volume, and in some regions they can not cover the whole volume of the cavity. This leads to a compensation of over- and underestimation and thus to a better convergence for the volume computation of cavities.

Most of the described approaches for the volume and surface area computation use a uniform sampling of a region of $\mathbb{R}^3$. In the literature one can find many modifications that use other pseudo random sampling techniques to get a better convergence. This was observed for the experiments of the surface area measurement with the ray casting technique which counts the intersections. The sampling by the Niederreiter sequences performed better than the default pseudo-random sampling. For the other algorithms, except the triangulation-based approach, similar tests were performed, where the result was close to the uniform sampling and only in rare cases slightly better. But on the flip side of the coin, these samplings can require some more computation time.

The interactive rendering with the combination of screen space ambient occlusion and the novel lighting of cavities provides a good depth perception and allows one to quickly identify the important regions. Nevertheless, it can be difficult to understand still pictures, because it is more intuitive for the user to see dark cavities, where typically no light exists. To make the illumination more consistent and the visualization more intuitive, often a dark background and a blur effect at the boundary is used, which creates the impression of glow.

Instead of using lights, one could also color the molecular surface according to the distance to the paths. Illumination, however, has the advantage that one can still use the color to display other properties of the molecule. Furthermore, one could use the lights for dynamic illumination of selected paths only and for path animations.

The visualization of the cavities using the ray casted skin surface is much faster than previous approaches, since it is not necessary to triangulate the surface or to create a 3-dimensional scalar field. The clipping of the molecular surface by the surface of the cavities greatly improves the understanding of the location of the paths with respect to the molecular structure.

8 Cavity Dynamics

Untill now, only the computation and analysis of cavities for a certain time step of a molecule was investigated. However, due to the dynamics of the molecules, also the cavity structure changes over time. These dynamics can lead to dynamic channels that work like pumps or to opening and closing pockets that lead to binding sites. To study cavity dynamics, it is necessary to trace them over time and to identify topological changes. The algorithm that is presented in this chapter consists of a pre-processing step where the structure of the cavities for each time step of the trajectory is computed from the Voronoi diagram of the van der Waals spheres using the algorithms described in Chapter 6. This step is followed by an interactive stage where the user can explore specific cavities and their spatiotemporal evolution. An overview of the cavity dynamics can be derived by rendering the dynamic cavities in a single image that gives the cavity surface colored according to its time-dependent dynamics. Additionally, more abstract graph representation of the cavity dynamics help to detect and keep track of important events.

The applicability of the resulting tool is shown for molecular dynamics trajectories of bacteriorhodopsin embedded in a hydrates lipid membrane. Bacteriorhodopsin is a light-driven proton pump in which proton translocation is coupled to protein structural rearrangements and relocation of internal water molecules [92, 4]. Because access of water and protons from the two sides of the lipid membrane is tightly regulated during vectorial ion transport, understanding the location, geometry (volume/spatial extent), and the dynamics of cavities large enough to host water is an important aspect of investigating conformation-coupled proton transport.

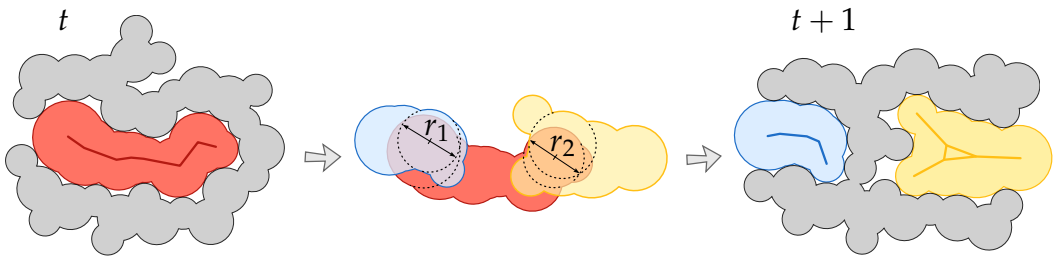


Figure 8.1: *Illustration of the tracing algorithm in 2D. The red cavity of time step t intersects the two cavities of time step $t+1$ with maximal radii r_1 and r_2 of the intersection circles. Both radii are large enough (i.e. larger than the user-defined minimal radius of the intersection sphere) such that the tracing detects a split event.*

8.1 Cavity Tracing

In this section, the computation of dynamic molecular cavities is described. As mentioned before, the computation consists of two stages. In the first stage, all static molecular paths and the corresponding cavities are computed for each time step separately. In the second stage, the paths and cavities are interactively traced over time. To achieve a high geometric accuracy, the Voronoi approach described in Chapter 6 is used for the first stage. Recall, that the result contains for each time step all maximal static molecular paths, that is all paths whose distance to the atom spheres becomes locally maximal and does not fall below a user-defined value, the probe radius. Additionally, the corresponding static cavities are given by the union of the empty spheres along these paths (Section 6.2.3).

From these static paths and cavities, the dynamic molecular paths and cavities will be derived in the second stage. A dynamic molecular path is defined by the connection of static paths over time. Due to the restriction on the computation of purely geometric paths and the discretization of the time, valid connections between static paths over time can be defined as follows. Consider, two connected components of static paths representing the skeletons inside two cavities of consecutive time steps. The probe sphere can be moved along the paths without intersecting the receptor molecule. A connection of two points of the two components is valid if the probe sphere can be moved along a continuous curve between these two points and lies always completely inside both cavities. This means, the overlap of the two cavities is large enough to host the probe sphere. Because it is not possible in practice to compute all possible connections, only a discrete number of points is checked. Additionally, a fast heuristic is used to accelerate the test if the probe lies completely inside both cavities.

8.1.1 Tracing

Often, the analysis of cavities and their dynamics depends on the application and interests of the user. For ligand binding applications it is possible that the

user wants to focus on a single cavity or a small subset of the cavities. But for some applications, the user wants to keep track of all cavities. To achieve this flexibility, the cavity tracing can be initialized by a manual selection of one or more start cavities of interest in an arbitrary time step. When proceeding to the following or previous time step, the selected cavities will be automatically traced over time. For each cavity, four topological events can occur. A cavity can appear, disapear, split, or merge with one or more cavities. If the user focuses on a selected subset of cavities, a new cavity cannot appear during the tracing.

As briefly described before, the tracing detects the evolution of cavities over time. In other words, it decides for each cavity if it belongs to a cavity of the next or previous time step. For a tracing over m time steps, this evolution can be stored in a directed m-partite graph $G = (C, E)$, where C consists of m disjoint sets of vertices C_t, with $t = 1, ..., m$. Each cavity of time step t is represented by a single vertex in C_t. If a cavity $c \in C_t$ belongs to a cavity $u \in C_{t+1}$ then $(c, u) \in E$. Furthermore, for each edge $(c, u) \in E$ with $c \in C_t$ and $u \in C_s$, $s - t = 1$. The topological events of the cavities are directly given by the degrees of the corresponding vertices in the graph. Let c be the vertex, which represents a cavity. The cavity appears if the indegree of c is zero. Analogously, it disappears if the outdegree of c is zero. If the cavity splits, the outdegree of c is larger than one, and in case that the cavity is the result of a merge, the indegree of c is larger than one. Note that the computations of the edges in G are independent of each other and can be done in parallel. The graph G is called *time graph* in the following.

Consider two path components of consecutive time steps together with their corresponding cavities. Recall that each cavity is represented by a finite set of empty spheres, as described in Section 6.2.3. To achieve an interactive tracing, the connections between the two cavities are computed by measuring the size of intersection circles of pairs of empty spheres. If the radius of an intersection circle between an empty sphere of the first cavity and an empty sphere of the second cavity is equal or greater than the user-defined probe radius r_p, the cavities belong to each other and the centers of the spheres will be connected, see Figure 8.1. On the other hand, if all radii of intersection circles between the two cavities are smaller than r_p, the cavities do not belong to each other. In detail a complete tracing step is performed in the following way.

For each empty sphere s of each cavity in time step t, all empty spheres of all cavities in time step $t + 1$ are detected that intersect s with an intersection circle larger than r_p. For all pairs of spheres that fulfill this condition, the corresponding cavities are mapped onto each other, which means an edge between the cavities is added to E in the time graph G. In addition, the path components will be connected at the centers of the spheres. Note that here the intersection circle of two spheres is defined as the largest circle inside the intersection volume of both spheres. The tracing can be restricted to a certain subset of cavities in t by considering only the empty spheres of these cavities.

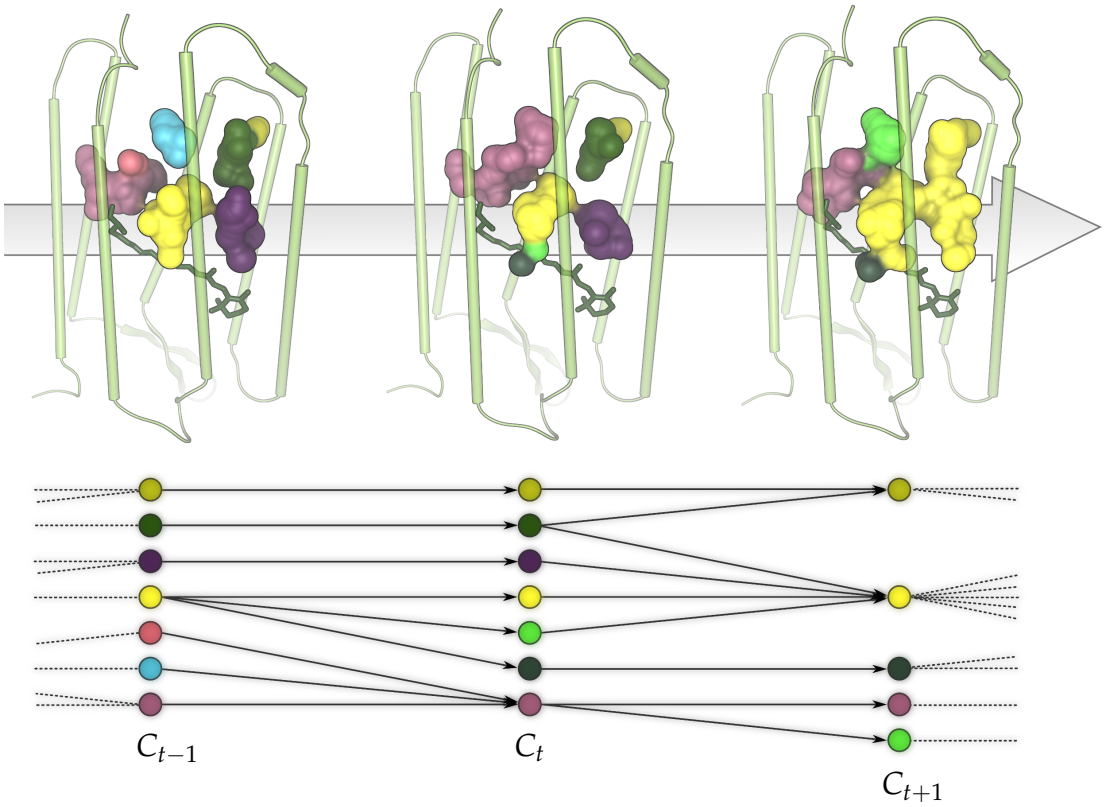


Figure 8.2: *Two tracing steps of the cavities in a bacteriorhodopsin monomer. The time graph is illustrated, and the identification numbers computed by the matchings are color-coded. Splits and merges can be identified, for example, from time step $t-1$ to t the yellow cavity splits into three cavities and then merges with four cavities from time step t to $t+1$.*

A further optional feature of the tracing algorithm is the detection and removal of dead ends. Dead ends are cavities that disappear during the tracing. These cavities are automatically identified, removed, and traced back in time until a splitting in the time graph is found. This allows the user to focus on stable cavities and to reduce visual clutter.

8.1.2 Assignments

While G contains already all relationships between the cavities over time, it does not contain unique assignments. In order to support the visual tracking, each component is assigned an identification number. This number can be related to a specific color and allows the user to easily visually trace single cavities over time, which can be seen in Figure 8.2. To represent the assignments of cavities, the identification numbers are computed by the following iterative approach. Assume that the identification numbers for all cavities in time step t are already computed. The numbers in $t+1$ are determined by computing a matching in the bipartite subgraph, which contains the cavities of t and $t+1$ and the edges between them. This matching represents the best

mappings of the tracing, so if cavity c_t is matched to cavity c_{t+1}, then c_{t+1} gets the same identification number as c_t. All unmatched cavities of $t+1$ are splits and all unmatched cavities of t are not traced to $t+1$ or merge into one or more cavities of $t+1$. The main problem is the definition of a matching criteria that represents the best tracing correlations. This is mathematically difficult to describe and might be ambiguous in several cases. Hence, the following heuristic is proposed.

For each edge in the subgraph, the intersection volume of the corresponding cavities is computed, as described below. Then the edges are sorted according to the intersection volumes, starting with the largest one. The sorted edge are consecutively processed. Let c_t and c_{t+1} be the cavities of the current edge in the list. If one of the two components is already matched, this edge is ignored and the algorithm continues with the next edge in the list. Otherwise, c_t is matched with c_{t+1}. This procedure is repeated until all edges in the list are processed. In the last step, the identification numbers are set according to the matching. For all unmatched cavities in $t+1$, new identification numbers are set so that finally each cavity in a time step gets a unique number. For the initial time step the cavities are numbered consecutively.

To compute the intersection volume of two cavities, the voxel approach described in Section 7.1.1 is modified. The two sets of spheres representing the cavities are combine into a single one. Let the first n spheres represent all empty spheres of the first cavity followed by the spheres of the second cavity. The rest of the algorithm is quite similar to the one described above. Only the condition during the distance tests is modified, such that the sample must be inside at least two spheres where one sphere has an index smaller than or equal to n and the other has an index larger than n. The performance of the implementation can be improved by reducing the number of samples. Instead of the bounding box of the combined set of spheres, one can take the intersection of the bounding boxes of both cavities, which again is an axis-aligned bounding box. Then, the grid is reduced to this bounding box, and all spheres that do not intersect the box are ignored. A result of this assignment is shown in Figure 8.2.

Based on the matching criteria, another option of the tracing system is to forbid splits. For this, only the matched cavities of $t+1$ are kept and all unmatched ones are removed. This allows the user to explicitly follow single cavities and analyze their locations and dimensions. In Figure 8.3, the absolute volume of a single cavity is plotted over time. The tracing of single cavities together with the volume computation also allow the user to compare the volume of the cavity with the actual number of water molecules contained in the cavity.

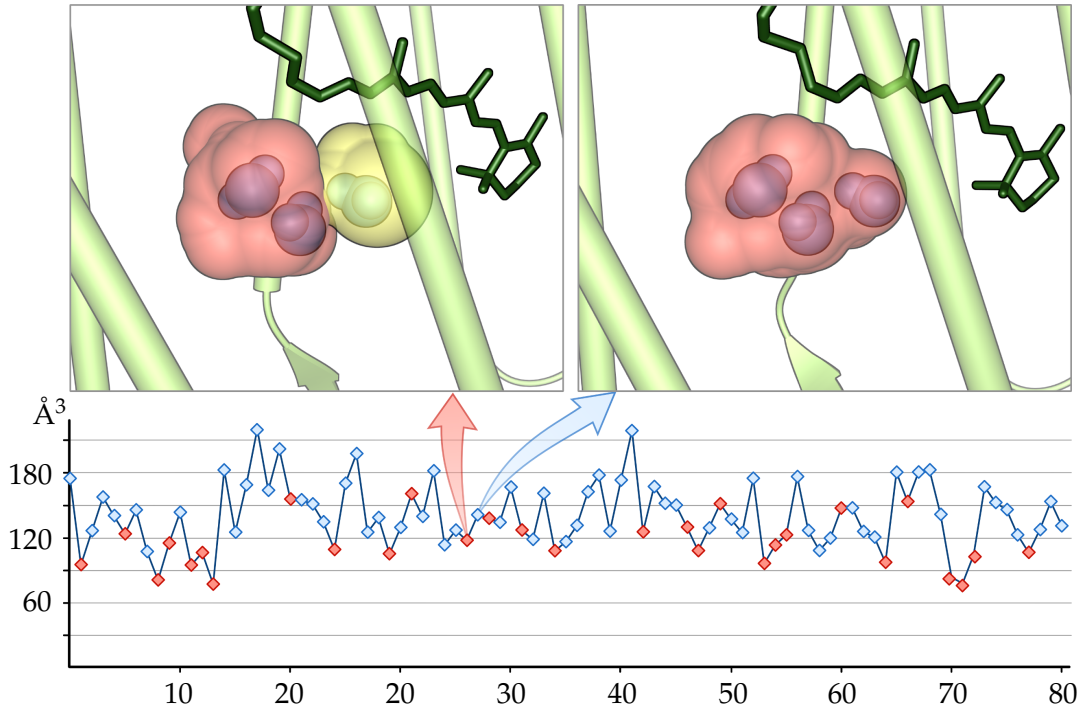


Figure 8.3: *Analysis of the correlation between the volume and the actual internal water molecules for a single cavity. The plot (bottom) shows the volume of the cavity, where blue markers indicate 3 internal water molecules and red markers 2. For two examples the 3-dimensional structure of the cavity (red) is shown.*

8.2 Dynamics Visualization

For the interactive cavity exploration, a visualization system was developed, which is described in this section. The system offers all common molecular representations, such as ball-and-stick, molecular surfaces, and secondary structure (Section 2.3). The surface representations include the van der Waals surface, the solvent accessible surface (SAS), the solvent excluded surface (SES), and the molecular skin surface (MSS). Onto all molecular representations, attributes can be mapped using pseudo-coloring. These attributes can represent properties of atoms, residues, and functional groups. Furthermore, filters can be applied to hide parts of the molecule that are of less interest.

Several methods to visualize molecular paths and cavities for a single time step were already provided in the previous Chapter in Section 7.2. Since the computation timings for these techniques are very fast, they can be also used to visualize time dependent data. If the user proceeds to the next or previous time step, the path and cavity visualizations will be immediately updated. The visualizations can be restricted to the selected and traced cavities and the identification numbers, computed during the tracing can be used to color the cavities.

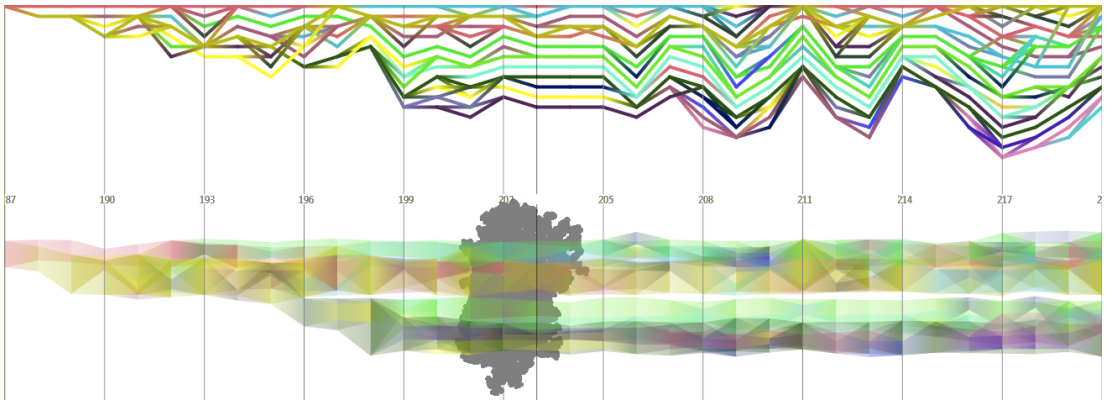


Figure 8.4: *While the split and merge graph (top) gives an abstract representation of topological changes of the cavities over time, the evolution graph (bottom) shows the spatial position and estension of the cavities for a user-defined direction.*

8.2.1 Timeline Visualizations

While the 3D visualization of cavities provides a good representation of their size and location, it does not necessarily allow efficient detection of splitting and merging events that can occur during the trajectory. For this reason, two different timeline visualizations of the time graph were developed. These 2-dimensional graph representations show topological and geometrical changes in a user-defined time range (Figure 8.4). Note, that the range of time can be changed interactively.

Split and merge graph

The first graph representation shows topological events like splits and merges in a similar way as in the sketch in Figure 8.2. Each traced cavity is visualized as a polyline from left to right, representing the direction of time. If a cavity splits into two or more cavities during tracing, the corresponding polyline also splits. Accordingly, polylines merge if the corresponding cavities merge. Due to the splits and merges, intersections of polylines can occur over time. These intersections could be possibly reduced using an optimized graph layout. Currently, however, the cavities are simply rendered from top to bottom. To keep consistency between the 3-dimensional visualization and the timeline graph, the same colors are used for both representations. Additional information, like the size of a cavity, can be encoded by the line thickness. Figure 8.4 (top) shows an example of a split and merge graph.

Evolution graph

The second timeline visualization shows the evolution of the traced cavities along a user-defined direction. Especially for membrane molecules, whose main path direction is often along the membrane normal, it is interesting to analyze the geometrical evolution of the cavities over time. Similar to the

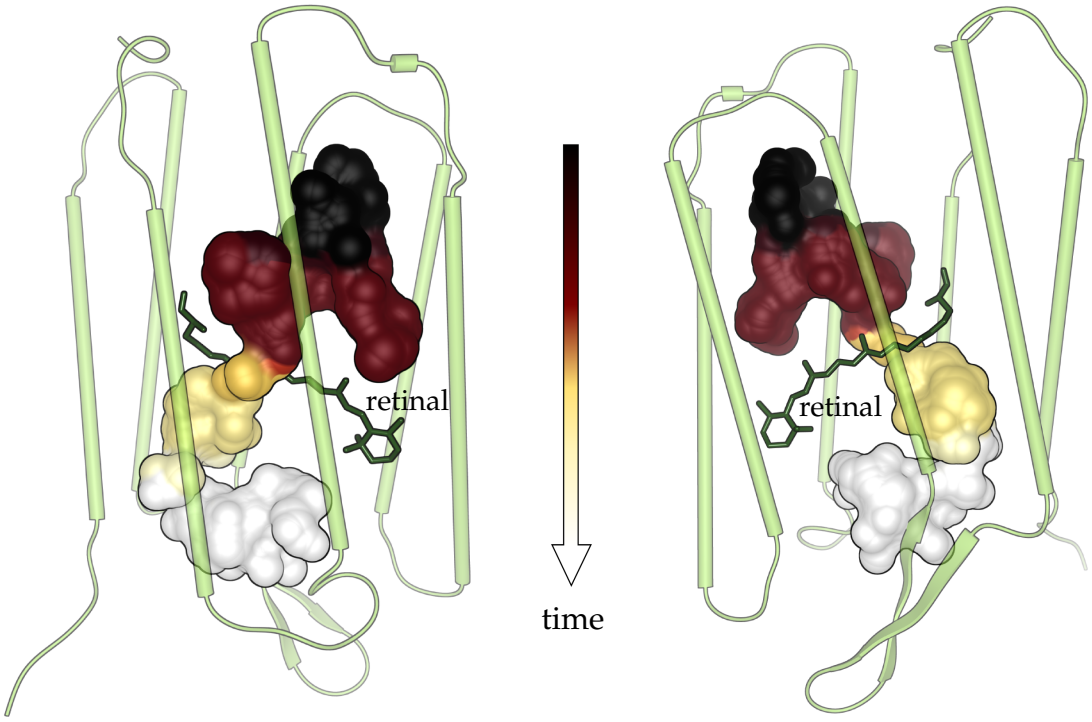


Figure 8.5: *Front and back view of a dynamic molecular cavity crossing the region of the retinal. The color indicates the evolution of the cavity over time.*

split and merge graph, each cavity is depicted as a polyline from left to right. The main difference is that the position and the thickness of a line depend on the position and expansion of the corresponding cavity along the selected direction. By using alpha blending, occlusion of lines is avoided. In addition, an orthogonal projection of the molecule is added to the background of the graph. This helps to identify the location of the cavities and to stay connected with the 3D view. One can see an example of an evolution graph of bacteriorhodopsin in Figure 8.4 (bottom). A single cavity that was traced creates a dynamic channel from the cytoplasm side (top) to the extracellular side (bottom). During the first time steps, only cavities above the retinal (Figure 8.5) get interconnected. Then, at time step 196, the cavities on the cytoplasmic side of the retinal connect with cavities on the extracellular side. It is important to mention here again that this connectivity across the retinal Schiff base region is a pure geometrical construct: it does not necessarily imply that a physically stable channel forms through the retinal region.

8.2.2 Cavity Dynamics

Instead of only animating the traced cavities and the molecular structures over time, the system provides a visualization of the dynamics of a cavity as compact representation in a single image. Recall that a dynamic path is a union of static path components that are connected over time. The user can extract an arbitrary dynamic path by selecting a cavity in the split and merge

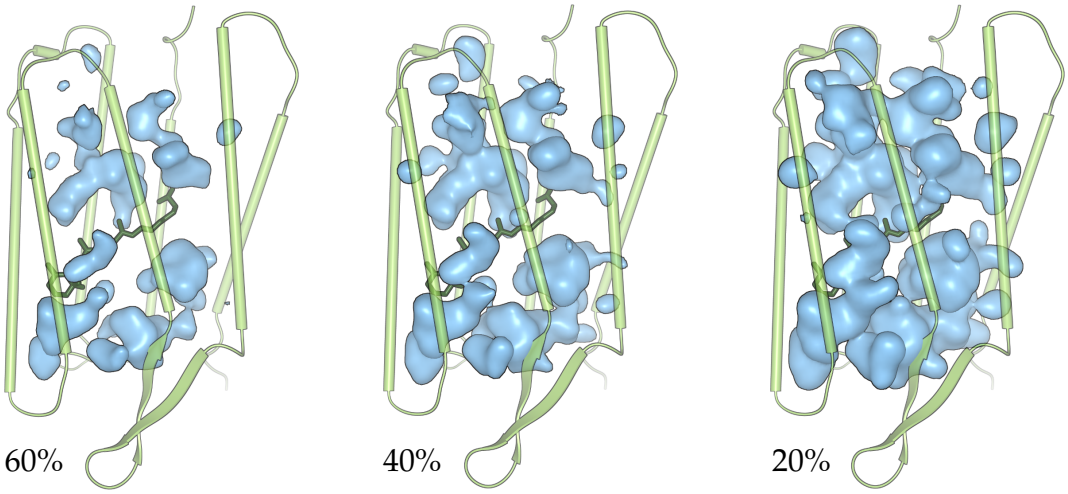


Figure 8.6: *Cavity residence probabilities for a complete monomer over the full trajectory visualized as isosurfaces. Only for probabilities smaller than 25%, a connection exists between cavities from the cytoplasmic region to the extracellular region.*

graph. To assist the selection, the evolution graph can be used for an overview of the progression of penetration. Once a time region of particular interest has been identified, the user can select a cavity in the split and merge graph. The end of the dynamic path is given by a further selection of a cavity in a subsequent time step. Then the path between these two cavities is computed by a modified depth first search. A different path can be achieved by adding further intermediate selections.

The dynamic cavity corresponding to the selected path can be rendered efficiently using again the skin surface approach. Therefore, the skin surface is computed of all empty spheres of all cavities belonging to the dynamic path. Thus, a static representation of a dynamic process is achieved. To still keep track of the path dynamics, a color-coding is added according to the time evolution of the cavities. In Figure 8.5 an example of a dynamic cavity, crossing the region of the retinal, is shown.

8.2.3 Cavity Probability

Visualizing the skin surface of a dynamic cavity is suitable to analyze its maximal dimension and dynamic progress, but sometimes the user is more interested in the residence probability instead of a hard boundary. This global representation of the cavity dynamics is given by the residence probabilities of all cavities. For a given time interval the *residence probability* of a point is the proportion of time in which the point is inside a cavity. These probabilities can be easily computed by regularly sampling $\mathbb{R}^3$ in the axis aligned bounding box of the molecule. For each sampling point, the number of time steps is detected in which the point lies inside a cavity. For a fast detection, again a 3-dimensional grid structure is used to store the cavities. Finally, the prob-

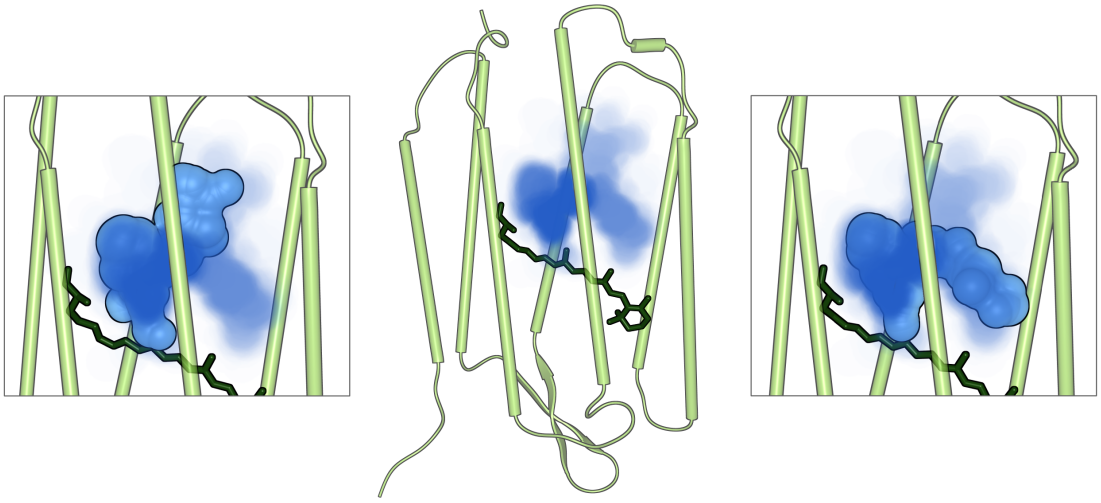


Figure 8.7: *Residence probability for a single user-selected cavity visualized by a maximum intensity projection. Two examples of the cavity variation are shown left and right.*

ability of each sampling point is the quotient of the detected number of time steps and the overall number of time steps. The residence information can be visualized using volume rendering or isosurfaces. Images for different isovalues for bacteriorhodopsin are shown in Figure 8.6. Note that the residence probabilities of this approach are similar to the ones described by Raunest and Kandt [200] and Jardón-Valadez et al. [99], in which water molecules are accumulated over time to generate a density that is visualized with an isosurface.

Additionally to this global visualization over all cavities, the user can also focus on a single cavity. For a selected time interval, the residence probability is computed as described above but restricted to a single cavity and its axis aligned bounding box. A volume rendering of the probability with a maximum intensity projection is suitable to highlight the fixed core of the cavity. In Figure 8.7, the maximum intensity projection (MIP) of the residence probability of a single dynamic cavity is shown together with two corresponding static cavities of two time steps. The stable core can clearly be distinguished from the more unstable cavity regions. In uncertainty visualization, similar techniques are used to distinguish between certain and uncertain regions of data sets [77, 194].

8.3 Results

8.3.1 Performance

Detailed computation and filtering timings of the molecular paths and cavities as well as rendering timings for the visualizations of the paths, cavities and the molecular structure were already given in Sections 6.3 and 7.3. In this section, the focus lies on the computation and visualization of the dynamic

cavities of a trajectory of one monomer of bacteriorhodopsin with 3 624 atoms and 2 000 time steps. These time steps were taken from the last 20 ns of an equilibrated simulation trajectory. The precomputation of the cavities took 21 min on an Intel Xenon 5650 with 2.67 GHz and 6 cores. Thus, the average computation time for one time step was approximately 630 ms. For the computation of the cavities, a probe radius of 1.4 Å was used, which approximates a water molecule. Since for the shown application, the internal cavities were more interesting, the boundary filter was set to 15%.

During visualization, a worst case frame rate was obtained when rendering the solvent excluded surface of the molecule clipped by the cavities. For this setting, frame rates between 40 and 50 fps were reached including path lightening and screen space ambient occlusion with depth cueing. For all other renderings, the frame rates were even higher. Note that for these performance measures a screen resolution of 1024×1024 was used and the whole protein with an average fill rate of 75% was shown. The rendering was done for a single time step with an NVIDIA Geforce GTX 470 graphics card. When the next time step was selected, the frame rate decreased due to the recomputation of the molecular surface and the tracing of the cavities. Nevertheless, the frame rate was still approximately 25 fps. The path tracing itself is so fast that it has nearly no influence on the performance.

8.3.2 Cavities in Bacteriorhodopsin

Visualizing the cavities of single time steps of the protein shows a structural barrier in the region retinal. Cavities from the cytoplasmic half are not connected to those on the extracellular side (Figure 8.8). This confirms the indications of previous molecular simulations [81] and reaction path computations [22]. The barrier prevents water molecules from moving between the two halves of the protein on the nanosecond timescale.

Due to the cavity dynamics and their topological changes, 3-4 dynamic channels can be traced, which connect the two halves of the protein (Figure 8.5). These channels are geometrically large enough to transport a water molecule. The appearance of cavities in the region of the retinal can also be seen in the path probability visualizations (Figure 8.6). However, these dynamic channels are the result of purely geometric computations, which do not consider electrostatics or non-bonded physical forces. Nevertheless, these channels provide new insights that need to be further investigated.

By analyzing single cavities over time, a cavity on the extracellular side of the retinal was detected that is large enough to host four or even five water molecules, but only three can be found in the crystal structure and in simulations (Figure 8.3). This is an interesting finding that clearly needs to be investigated. One reason can be the high dynamics of the cavity including topological changes such as splits.

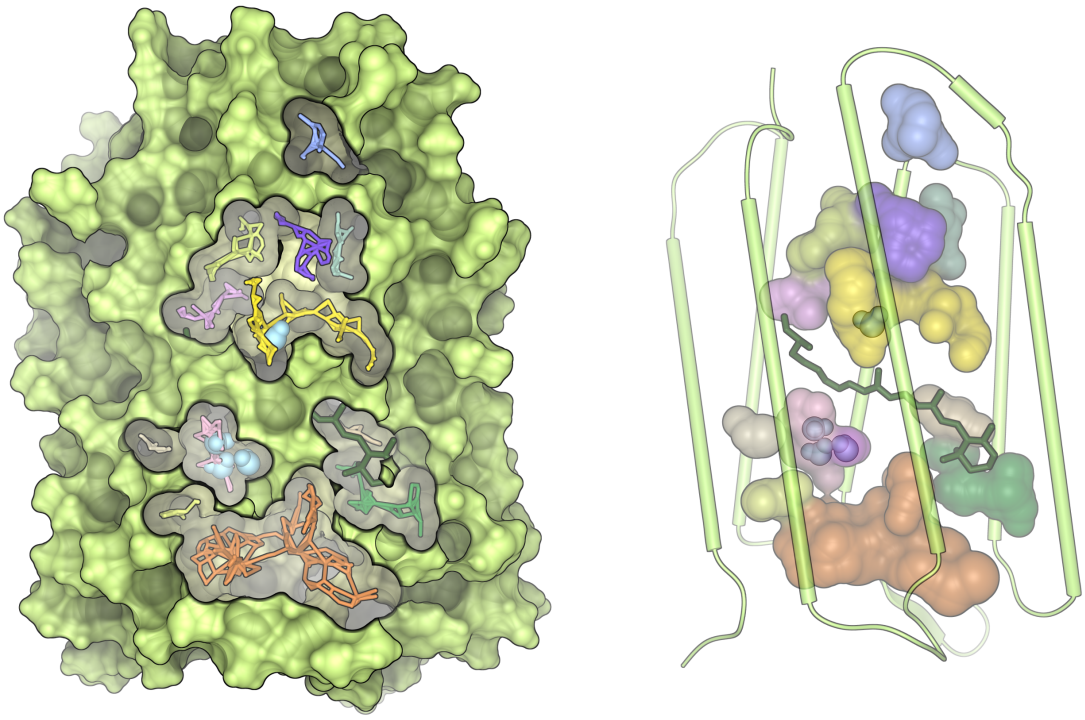


Figure 8.8: *Visualization of the cavities of a snapshot of a monomer of bacteriorhodopsin as surface cut (left) and in combination with the secondary structure (right). The water molecules close to the retinal are shown in blue.*

8.4 Discussion

Exploring the dynamics of cavities is of potential interest to study protein dynamics. The versatile tool presented in this chapter combines the advantages of high geometric accuracy and advanced path and cavity visualization with interactive analysis of molecular dynamic trajectories. The tool is the first that can be used to compute all possible geometric cavities with a user-defined minimum constriction size for time-dependent data.

Although, the tool requires a precomputation of the cavities, it lies still in a reasonable amount of time to study even long trajectories. One could also think about an extension that enables already the analysis of the processed time steps during the precomputation. Another idea is to use a simpler approach that can be applied in real-time as preview.

The quality of the tracing depends mainly on two factors. First, how good the intersection of two empty spheres can reflect the overlap of the full cavities. Since the probe itself is a sphere, the intersection between two empty spheres seems to be the most reasonable bottleneck connection. In all experiments, complex overlaps that result from multiple empty spheres, without a valid tracing of two empty spheres, were not observed. And second, the resolution of the trajectory over time needs to be sufficiently high. The closer the

time steps, the better is the tracing. Too large time steps can lead to missing, false positive, or and false negative tracings.

The residence probability of all cavities computed for a certain time range gives a good overview of cavity dynamics and could be used to identify cavities that could be subject to closer investigation. When computed for a single dynamic cavity selected by the user, the residence probability gives insight into the dynamics of a particular region of the protein. The visualization of the cavity dynamics could be further enhanced by adding movement illustrations as suggested, for example, by Meyer-Spradow et al. [166].

The current implementation of the cavity analysis tools allow the user to exploit the timeline graphs to rapidly identify events where cavities split or merge. This task can become difficult in case of graphs containing a large number of cavities. To circumvent this limitation, an optimized layout of the split and merge graph would be helpful.

9 Ligand Excluded Surface

For the accessibility visualization using the SES in Chapter 3 as well as for the cavity analysis in Chapters 6 and 7, the ligand is approximated by a probe sphere. With this approximation, a real-time computation of the SES and a cavity analysis can be achieved in a reasonable time, even for dynamic data. However, despite the fact that the geometrical accuracy for the cavity detection is higher than in most other approaches, if the shape of the ligand differs significantly from a sphere, a different approach is necessary. In this chapter a new surface model is presented that shows the accessibility for a receptor molecule based on the van der Waals surface of the ligand and its dynamics. Thus receptor and ligand are geometrically represented in the same manner. According to SES, the new surface is called *ligand excluded surface* (LES). The surface was presented in "*Ligand Excluded Surface: A New Type of Molecular Surface*" in 2014 [151]. An example of an LES is shown in Figure 9.1, which also demonstrates the difference to the SES.

First, the LES is defined as an implicit surface of a distance field, then it is discretized for the computation using a grid representation. In order to approximate the distance field well enough, a possibly large number of orientations of the ligand needs to be considered. Since a brute-force approach placing the ligand at all positions of the grid with a large number of orientations would be computationally too expensive, a two-phase approach is applied that significantly reduces the computational effort. Moreover, an implementation of the method on the GPU is proposed, which further speeds up the computation.

Fortunately, apart from the surface, the algorithm for computing the LES can be easily extended to compute also cavities that are large enough to host

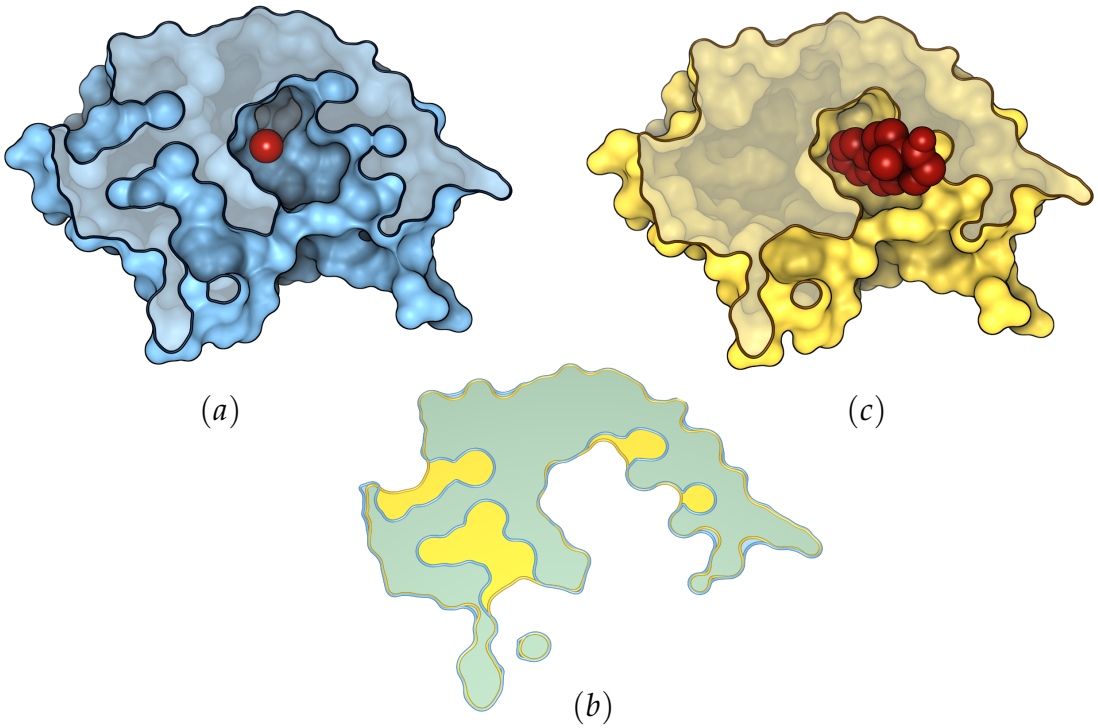


Figure 9.1: *Comparison of solvent excluded surface (SES) and and ligand excluded surface (LES) on the example of ketosteroid isomerase (pdb: 1OGZ). To simplify the comparison, both surfaces have been cut. (a) SES of isomerase with probe radius* 1.4 *Å. The probe is depicted as red sphere. (c) LES of isomerase for the ligand equilenine, computed with grid spacing* 0.25 *Å and* 200 *orientations. The ligand (red) is depicted in its active position. (b) Overlaid surface cuts showing differences between SES and LES.*

the ligand molecule (Figure 9.7). In addition to the geometry of the cavities, also information about how the ligand is positioned inside the cavities are obtained. This might be of particular interest for the application of subsequent docking simulations.

9.1 LES Definition

Informally, the LES of a receptor can be defined as the surface enclosing the space around the receptor that is not accessible to a specific ligand. Based on this, a mathematical definition of the LES is derived.

Let r be the receptor molecule for which the LES is defined based on a ligand l. Let the receptor consist of n atoms with positions $p_i^r \in \mathbb{R}^3$ and radii $r_i^r \in \mathbb{R}$, with $i = 1, ..., n$. Furthermore, let the flexibility of the ligand l be given by a set of c conformations, and let the ligand consist of m atoms with positions $p_{jk}^l \in \mathbb{R}^3$ and radii $r_j^l \in \mathbb{R}$, with $j = 1, ..., m$ and $k = 1, ..., c$. The state of the ligand is described by a conformation $k \in \{1, ..., c\}$, a translation $T \in \mathbb{R}^3$, and an orientation $R \in \mathbb{SO}^3$, which is given by a rotation (Figure 9.2, left). A state is defined as valid if no ligand atom intersects any

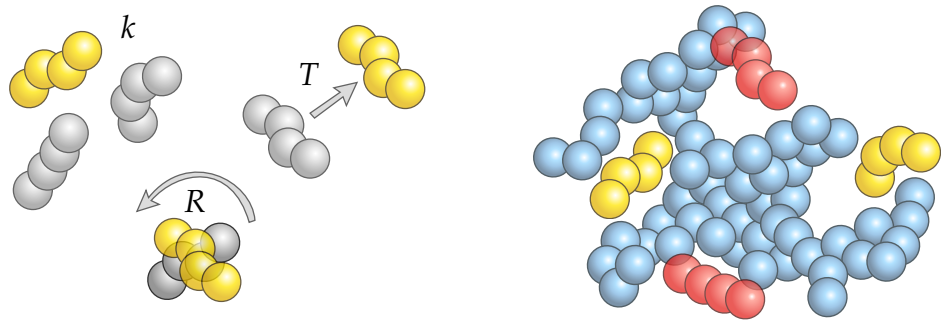


Figure 9.2: *Left: A ligand state (yellow), given by a conformation k, a rotation R, and a translation T. Right: Valid (yellow) and invalid (red) ligand states according to the receptor molecule.*

atom of the receptor molecule. This can be formulated using a binary function $I_{r,l} : (\mathbb{N}, \mathbb{R}^3, \mathbb{SO}^3) \to \{0,1\}$ that is 1 if the state is valid and 0 otherwise (Figure 9.2, right).

$$I_{r,l}(k,T,R) = \begin{cases} 1, & \left\| p_i^r - (Rp_{jk}^l + T) \right\| \geq r_i^r + r_j^l, \forall i = 1,...,n, \forall j = 1,...,m \\ 0, & otherwise. \end{cases}$$

Now the ligand excluded surface can be defined as the surface that encloses all points in $\mathbb{R}^3$ that are not reachable by a valid ligand state. This is mathematically described by a ligand-dependent distance function $d_{r,l}$ with

$$d_{r,l}(p) = \max_{\substack{k=1,...,c, \\ T\in\mathbb{R}^3, R\in\mathbb{SO}^3}} \begin{cases} \max\limits_{j=1,...,m} r_j^l - \left\| p - (Rp_{jk}^l + T) \right\|, & I_{r,l}(k,T,R) = 1 \\ -\infty, & otherwise. \end{cases}$$

The ligand excluded surface is then given as the implicit function of all points p with $d_{r,l}(p) = 0$. The inner maximum is a distance function of a valid ligand state. It returns a reverse distance, which means the distance is positive if the point lies inside the ligand and negative if it lies outside (Figure 9.3, left). For simplicity, but without loss of generality and correctness, the distance function does not return the correct Euclidean distance for points inside the ligand. However, the distance converges monotonically to 0 if the point approaches the ligand boundary. The outer maximum selects a valid ligand state that results in the largest distance for p. Thus, for a point inside the LES, the closest valid ligand state does not contain the point, which results in a negative distance, because of the ligand reverse distance function, (Figure 9.3, right). Note that $d_{r,l}$ is bounded in positive direction by the largest atom radius of the ligand. The exception that no valid state exists for a point p is only of theoretical interest.

For a 'ligand' consisting of a single atom, the LES is equal to the SES; thus the LES is a generalization of the SES. In contrast to the SES, it is difficult to compute the ligand excluded surface analytically. For this reason the surface

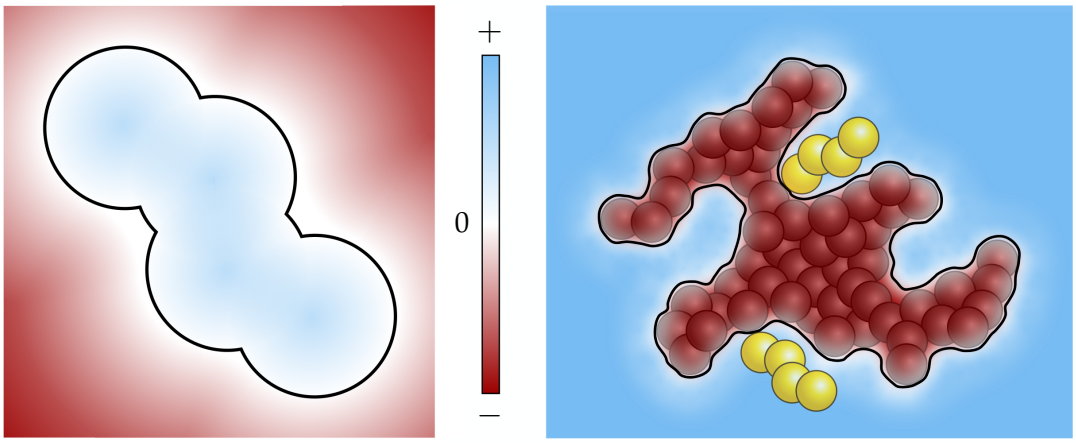


Figure 9.3: *Local reverse distance function of a ligand state (left) and overall distance function $d_{r,l}$ (right). The LES (black) is the implicit surface of this distance function.*

is geometrically approximated by discretizing the space of ligand states. By using a finite set of c conformations, the dynamics of the ligand are already discretized. Additionally, the orientations and positions of the ligand are discritized. To do so, two parameters are introduced, the number $o \in \mathbb{N}$ of ligand orientations and the spacing $g \in \mathbb{R}$ of the cubic grid being used for uniform sampling of the ligand positions. Based on this discretization, the LES can be computed as follows.

9.2 Algorithm

Before the algorithm is described in detail, an overview of the main steps is given (Section 9.2.1). Then, the two phases in which the algorithm can be subdivided are presented (Section 9.2.2 and 9.2.3). This is followed by the computation of the cavity structure based on the LES (Section 9.2.4).

9.2.1 Overview

According to the LES definition, first a signed distance field is computed that is negative inside the LES and positive outside. Hence, the LES is the implicit surface of this distance field. To compute the LES, it is not necessary to compute the complete exact distance field; it needs to be exact only in the vicinity of the LES. The field is discritized by a cubical grid that is initialized with $-\infty$. In order to efficiently approach the correct distance field values close to the LES, a two-phase approach is applied.

In the first phase, a sphere completely enclosing the ligand is used to update the distance field at positions of the grid where the ligand can rotate around its center without ever intersecting any atom of the molecule. In the following, this sphere is called *ligand bounding sphere*. For all positions at which the ligand bounding sphere (blue points in Figure 9.4, left) can be placed, an

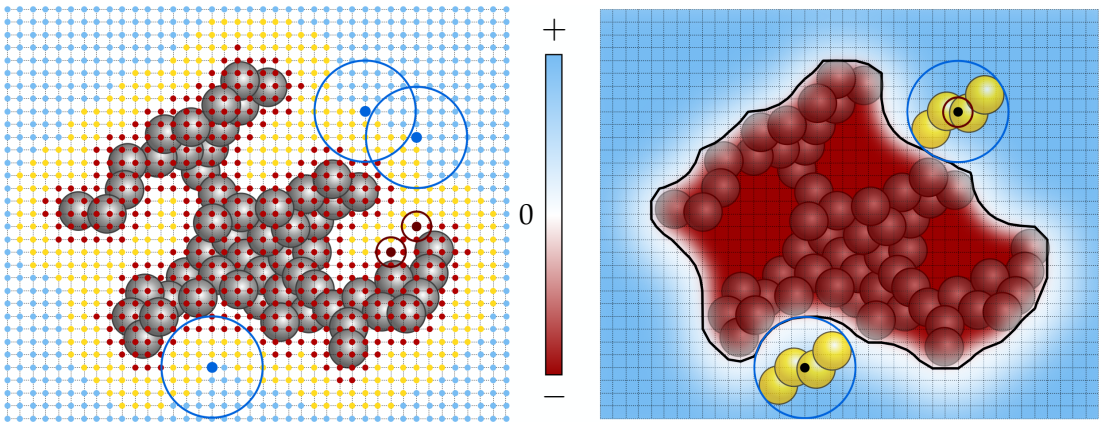


Figure 9.4: *Illustration of phase I of the algorithm. Left: The gray spheres depict the receptor molecule and the grid points show the discretization of the space around the molecule. Red dots mark sample positions where the ligand inscribed sphere (red circles) cannot be placed without intersecting the receptor; blue dots mark sample positions where the ligand bounding sphere (blue circles) can be placed without intersecting the receptor. The yellow dots mark the remaining sample positions, which need to be processed in phase II. Right: The distance field after completing phase I. The resulting implicit surface is equal to the SES of the ligand bounding sphere. The yellow spheres depict the ligand molecule enclosed by its ligand bounding sphere.*

update of the distance field for all grid points inside the bounding box of the ligand bounding sphere is applied. The resulting distance field is depicted in Figure 9.4 (right) together with its implicit surface. This implicit surface is a discrete representation of the SES for a probe sphere equal to the ligand bounding sphere. Note that after computing the distance field, the largest distance value will be equal to the radius of the ligand bounding sphere. This value is set at all positions where the ligand bounding sphere could be placed. In addition to modifying the distance field, all positions where the ligand bounding sphere could be placed are marked. These positions do not need to be considered any further. In this first phase, a second sphere is used to conservatively mark grid points at which no ligand can be placed without intersecting the molecule, including grid points inside the molecule but also some grid points close to the LES. These grid points are depicted in Figure 9.4 (left) as red points.

The second phase completes the computation of the distance field in the vicinity of the LES. It uses all information computed in the first phase, in particular the grid positions that are neither marked blue nor red. These are marked as yellow points in Figure 9.4 (left) and represent the remaining sampling positions to be considered. At these positions, it will now be tested whether the ligand can be placed without intersecting with any atom sphere of the molecule. To approach the distance field as well as possible, the ligand will be rotated at each of these positions. Furthermore, if the ligand is flexible, more than one conformation can be considered. For each rotation and each conformation for which the ligand does not intersect any atom of

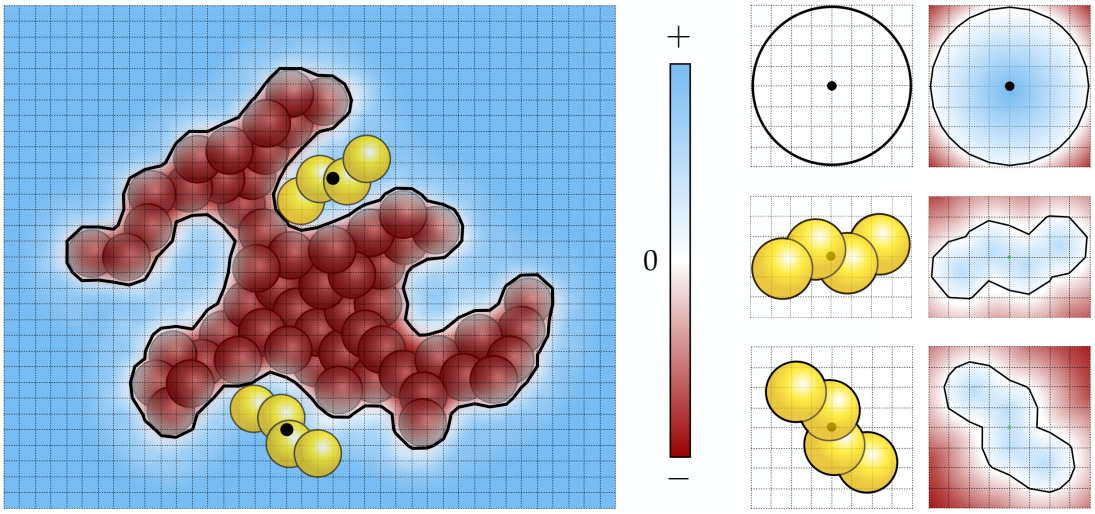


Figure 9.5: *Left: The distance field and resulting LES after completing phase II of the algorithm. Each grid position in the vicinity of the LES stores the minimal distance to a valid ligand. Right: Illustration of local discrete distance fields. The first row shows the ligand bounding sphere and the corresponding distance field which is used for the distance updates in phase I. The second and third rows show two ligand orientations and the corresponding local distance fields. These fields are used for the distance updates in phase II.*

the molecule, the distance field of all grid points inside the bounding box of the ligand will be updated. Note, however, that the distance values will be updated only if the values increase. The result of phase II is illustrated in Figure 9.5 (left).

9.2.2 Phase I

In phase I, the discrete field is sampled with two spheres. The first one is a small sphere that is inscribed in the ligand atom spheres and is therefore referred to as *ligand inscribed sphere* (see red spheres in Figure 9.4, right). The second sphere is usually much larger than the first one and completely contains all ligand atom spheres. As mentioned previously, this sphere is denoted as *ligand bounding sphere* (see blue spheres in Figure 9.4, right). These spheres are used to initialize the distance field and to make later computations more efficient.

Ligand Bounding Sphere

The ligand bounding sphere is defined as the smallest sphere completely containing all ligand atom spheres. If $p_b \in \mathbb{R}^3$ is the position and $r_b \in \mathbb{R}$ is the radius of the ligand bounding sphere, then $\|p_b - p_i^l\| + r_i^l \leq r_b$, for all $i = 1, ..., m$, and no smaller sphere exists for which the inequality also holds. The minimal bounding sphere of a set of spheres can be tangent to more than four input spheres, but it is already defined uniquely by at most four spheres. It can be computed with a simple iterative algorithm. More efficient

algorithms have been proposed [65], but since the number of atoms in a ligand is generally small and the computation has to be performed only once per conformation, the simple algorithm is sufficient. The algorithm starts by computing the minimal bounding sphere of four randomly selected input spheres [70]. Then, the bounding condition of this bounding sphere is checked for all remaining input spheres. If a sphere is not enclosed by the current bounding sphere, the sphere is interchanged with one of the previous four selected spheres such that the radius of the new minimal bounding sphere becomes maximal. This procedure is repeated until all input spheres are enclosed by the current bounding sphere.

The ligand bounding sphere is computed for each ligand conformation separately. Then, the overall ligand bounding radius $r_{max} \in \mathbb{R}$ is the maximum of the radii of all ligand bounding radii r_b computed for all conformations. The sphere with radius r_{max} can be used to determine positions on the grid, where all ligand conformations, no matter what orientations they have, can be placed without intersecting any receptor atom sphere. This is illustrated in Figure 9.4.

Ligand Inscribed Sphere

The ligand inscribed sphere is also computed for each ligand conformation separately. To determine the ligand inscribed sphere for a single conformation, the sphere is placed at the center of the ligand bounding sphere. Then the maximal radius is determined such that the sphere is completely enclosed by the atoms of the ligand. The ligand inscribed sphere is illustrated as red circle in Figure 9.4. Note that the ligand inscribed sphere has a negative radius, if the center of the ligand bounding sphere lies completely outside the ligand conformation.

From the ligand inscribed spheres of all ligand conformations the radius $r_{min} \in \mathbb{R}$ is computed as the minimum radius of the ligand inscribed spheres over all conformations. The sphere with radius r_{min} will be used to exclude grid positions from being tested with either the ligand bounding sphere or the ligand geometries. The reason for this is that if a sphere with radius r_{min} intersects the protein atoms, all ligand conformations placed at the same position, no matter what orientations they have, will also intersect the protein atoms (Figure 9.4).

The two spheres with radii r_{max} and r_{min} accelerate the surface computation as will be described next.

Distance Field Initialization

In phase I, the ligand positions are uniformly sampled and the discrete distance function is initialized. According to the definition, a sample position will be denoted by T. Note that the same samples are used for the ligand positions and the grid of the distance function. In addition, all positions will

be detected for which all ligand conformations and orientations have to be investigated.

To compute the discretization of the sample positions and the distance field, first the minimal axis-aligned bounding box of the atom spheres of the receptor molecule is computed. This box is then extended in all directions by r_{max}. Afterwards, the box is uniformly sampled in all directions with a user-defined grid spacing g.

The distance field is initialized with the negative value of the maximal atomic radius r_r of the receptor. Then the algorithm iterates over all sample positions T and performs at each position an intersection test of the receptor atom spheres with the minimal ligand inscribed sphere with radius r_{min}. If an atom i intersects the sphere, that is $\left\| p_i^r - T \right\| < r_i^r + r_{min}$, all ligand conformations and orientations also intersect this atom. Hence, the position can be ignored for further investigations. In case that the minimal ligand sphere does not intersect the receptor, a second intersection test is performed with the maximal ligand bounding sphere having radius r_{max}. If this sphere does not intersect the receptor atom spheres, all ligand conformations and orientations at this position are valid, that is, they do not intersect the receptor. In this case, the distance value at this position is set to r_{max}. For all remaining positions $\tilde{T}$, where the maximal ligand sphere intersects the receptor and the minimal ligand sphere does not, the ligand conformations and orientations need to be investigated in phase II. These positions are marked by yellow points in Figure 9.4, left.

Finally, the algorithm iterates over all sample positions T at which the distance field has a value equal to r_{max}. If one of the 26 neighboring grid points has a distance that is smaller than r_{max}, the distance function in the neighborhood is updated using the local distance function of the ligand bounding sphere, which is depicted in Figure 9.5, right. This function has a value of r_{max} in the center, 0 at the border of the sphere, and outside the sphere, the values take on the negative distance to the sphere border. The new distance at a neighboring position p is set to the maximum of the old distance and $r_{max} - \|p - T\|$. After this phase, the implicit function defined by the current distance function represents the discrete SES with probe radius r_{max} as can be seen in Figure 9.4, right.

9.2.3 Phase II

In the second phase of the algorithm, the discrete distance function is refined in the vicinity of the LES. For this, all sample points $\tilde{T}$ (yellow points in Figure 9.4) are considered that might contribute to this refinement. At all these points, all conformations of the ligand are placed one after another and in case the state of the ligand is valid, the distance field is updated accordingly. Of course, considering only a single orientation per ligand conformation will lead to large errors in approximating the LES. Therefore, a user-defined number of o orientations for each ligand conformation is used. These orientations

should be sampled such that the space of all orientations is well represented. An algorithm for computing such a sampling of orientations is described in the next section followed by a detailed description of the refinement step.

Precomputation of Ligand Orientations

To obtain the most different ligand orientations, the computation of the orientations is done for each conformation separately. Consider a single conformation with m atoms whose positions are $p_i^l \in \mathbb{R}^3$ and atomic radii are $r_i^l \in \mathbb{R}$, with $i = 1, ..., m$. For the purpose of sampling the orientations, the ligand is moved such that the center of its ligand bounding sphere lies in the origin. The transformed atom positions $\tilde{p}_i^l \in \mathbb{R}^3$ are given by $\tilde{p}_i^l = p_i^l - p_b$, for all $i = 1, ..., m$. Due to transforming the ligand such that its bounding sphere center lies in the origin, for each rotation of the transformed ligand, all atoms are always inside the ligand bounding sphere with radius r_b. For the computation of the o orientations, first a uniform distribution of points on the surface of the unit sphere is computed. The vectors from the center of the unit sphere to the sampled points represent the axes for the rotations of the orientations. Furthermore, the angles for the rotations around each axis are uniformly sampled. Initially $\tilde{o} = 10 \cdot o$ orientations are computed. Afterwards, the differences between orientations are computed and the most significant o orientations are selected. These orientations are defined as the ones with the maximal minimal distance between the ligand orientations. This strategy allows filtering unnecessary orientations due to, for example, symmetry properties of the ligand. Let R_j be the initial sampled rotations with $j = 1, ..., \tilde{o}, \tilde{o} \gg o$. The difference between orientations can be approximated by the root mean square metric (RMSM) of the atom positions. Let $D \in \mathbb{R}^{\tilde{o},\tilde{o}}$ be the symmetric matrix that stores the distances between the orientations, then

$$d_{j,k} = \sqrt{\sum_{i=1}^{m} (R_j \tilde{p}_i^l - R_k \tilde{p}_i^l)^2}.$$

with $d_{j,k}$ being the matrix element in row j and column k. The goal is to find the o most different orientations of the $\tilde{o}$ orientations. This can be mathematically formulated by finding the set of o orientations where the minimal distance becomes maximal. One can approximate the solution of this optimization problem using k-means clustering, which is much faster than computing the optimal solution. The k-means clustering algorithm starts by randomly selecting o orientations as cluster centers. Then, each orientation R_i is assigned to the cluster center R_j with the minimal distance $d_{i,j}$. This creates o clusters of orientations. In each cluster, the orientation is computed the maximal distance of which within the cluster becomes minimal. If this orientation is different to the previous selected center, the orientations are interchanged. Then the assignments and the new cluster centers are recomputed. This is repeated until no cluster center changes anymore. To achieve an even better

solution, the algorithm can be run several times. The best solution is then the one with the maximal minimal distance.

Distance Field Refinement

With the discretization of the ligand positions, orientations, and conformations, the final part of the algorithm is executed. During this last step, at each position of $\tilde{T}$, intersection tests of the ligand with the receptor molecule are performed for all conformations and orientations. If the ligand does not intersect the protein for a certain position, orientation, and conformation, the distance function is updated using the reverse distance function of the ligand as depicted in Figure 9.3, left. In detail, the following steps are performed.

1: **for** $k = 1, ..., c$ **do**
2: **for all** orientations R **do**
3: **for all** ligand positions $\tilde{T}$ **do**
4: **if** $I_{r,l}(k, \tilde{T}, R) = 1$ **then**
5: **for all** distance samples $p \in \mathbb{R}^3$ **do**
6: $$d_{r,l}(p) = \max\left(d_{r,l}(p), \max_{j=1,...,m} r_j^l - \left\|p - (Rp_{jk}^l + \tilde{T})\right\|\right)$$
7: **end for**
8: **end if**
9: **end for**
10: **end for**
11: **end for**

Note that the values of the distance function are only updated if they become larger. During the distance updates, points reachable by a ligand become positive and unreachable points stay negative, but might still increase. Since the ligand excluded surface is the surface which separates positive and negative values, it is defined by the implicit description $d_{r,l}(p) = 0$.

9.2.4 Cavity Structure

Apart from computing the LES, the described sampling approach can be also extended to extract the cavity structure of the receptor. To do so, all valid and invalid ligand states need to be stored during the algorithm. The states are maintained in a bit array of length $c \cdot o$ at each sample point. In this bit array, for each ligand conformation and orientation, it is stored whether the ligand is valid or not. If the conformation and orientation is valid, the bit is set to 1, otherwise it is set to 0. Note that for sample points where the maximal ligand bounding sphere can be placed, all bits of the array are set to 1. On the other hand, all bits of the array are set to 0 if the minimal ligand inscribed sphere intersects the receptor molecule. Thus, it is only necessary to store bit arrays for all sample points $\tilde{T}$, depicted as yellow points in Figure 9.4, left.

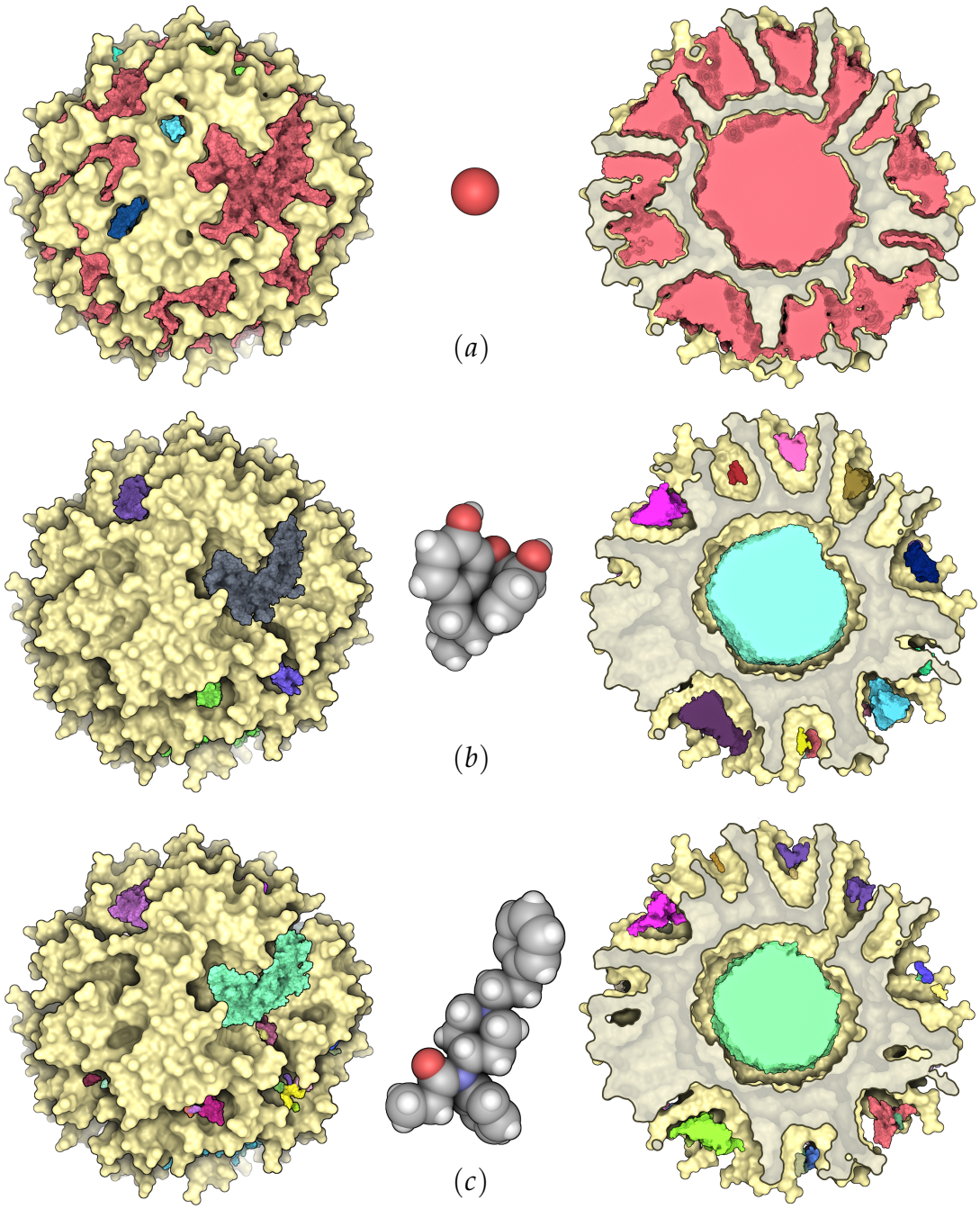


Figure 9.6: *SES and LES with cavity cores of a dendritic core multi-shell nanotransporter: (a) SES, (b) LES of morphine, (c) LES of fentanyl. In the image, the probe sphere and the ligands have been scaled by a factor of 5. The surface of the nanotransporter is depicted in yellow, the cut of the surface is gray, and the cavity cores are colored according to their clustering.*

All points with at least one bit set to 1 define valid sample points of a cavity. However, usually one is only interested in positions that represent real cavities. This requires the definition of a boundary (Section 2.4). To do so, the

same ambient occlusion technique is applied as described in Section 6.2.1. For each sample position with at least one valid ligand state, a set of uniformly distributed rays is cast from this position. The ambient occlusion value for this sample is the quotient of the number of rays that hit any receptor atom sphere and the overall number of rays. Thus, the higher the ambient occlusion value the less ambient light is received at this position and the deeper the sample lies inside the receptor molecule. All sample points with a value less than a user-defined threshold are removed from the following considerations.

The remaining sample positions are then clustered to get the cores of the cavities. Two neighboring sample positions belong to the same cluster if at least one entry is 1 at the same position in both bit arrays. This means, for all samples of the same cluster, the ligand can move from one sample position to a neighboring one without changing the orientation or conformation. At a sample position, the ligand can change its orientation or conformation to another valid state. Each cluster defines a core of a cavity. Additionally, small clusters can be filtered out according to a user-defined minimal cluster size. Examples of clustered sample positions to cavity cores can be seen in Figure 9.6 for a nanotransporter and different ligands.

From the cores of a cavity, its surface can be easily generated using again a discrete distance field. This is also initialized with a negative value, for example $-r_r$. Then the distance field is updated according to the maximum of all distance functions of valid ligand states of the cavity. The surface of the cavity is then defined as the implicit surface given by this distance field. In Figure 9.7, the three main cavities are shown. Based on the storage of valid and invalid ligand states and the clustering, one can easily compute trajectories of the ligand from inside the cavity to the outside or vice versa.

9.3 Implementation

The algorithm, as described in the previous section, is quite expensive in case of a naïve implementation. In this section, several optimizations and their implementation are proposed for the most critical parts of the algorithm.

9.3.1 Intersection Tests

During the execution of the algorithm, many intersection tests between spheres have to be carried out. In phase I, the algorithm performs intersection tests between the minimal ligand inscribed sphere and the maximal ligand bounding sphere with the atom spheres of the receptor. In phase II, intersection tests between the ligand atoms and the receptor atoms need to be computed. Without using a data structure to reduce the number of intersection tests, $n \cdot m$ sphere-sphere intersections tests need to be carried out for a single receptor ligand intersection test in the worst case, where n is the number of receptor atoms and m the number of ligand atoms. Of course, the algorithm stops

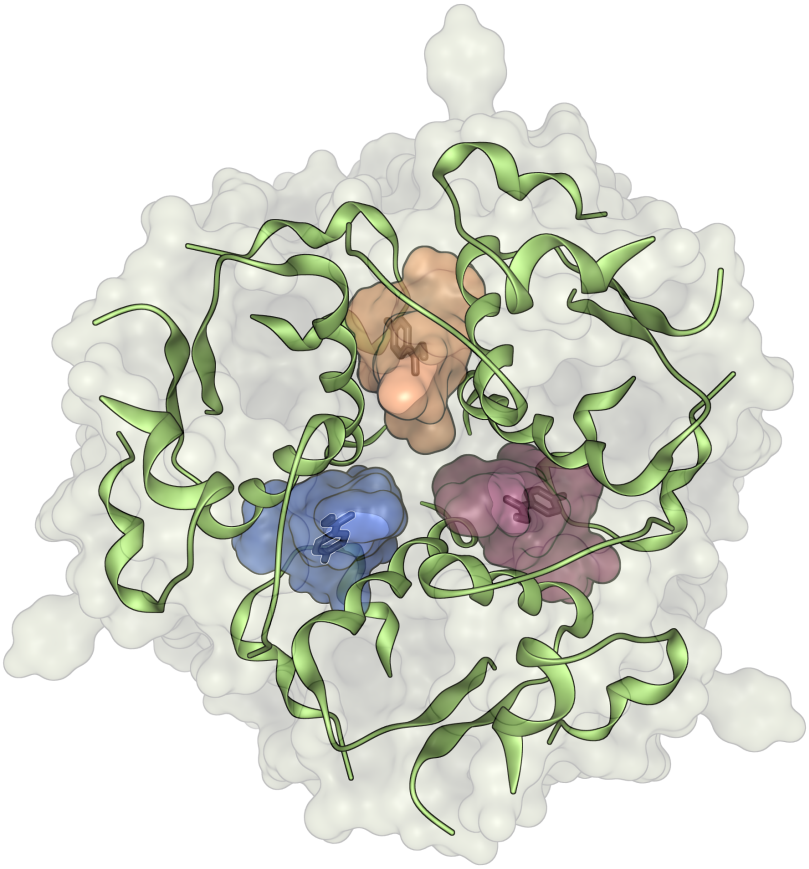


Figure 9.7: *Cavities of hexameric insulin (pdb: 3MTH) for methylparaben, computed by the LES algorithm (with 200 orientations). The three main cavities are depicted by their surfaces together with methylparaben. The insulin molecule is represented by its secondary structure and blended with the LES.*

testing if the first intersection has been found, but if the molecules do not intersect, all intersection tests are needed. Hence, a data structure is required to reduce the number of intersection tests. Since the receptor molecule is static, the atoms can be stored in a 3-dimensional grid data structure (Section 2.5). This reduces the complexity for the intersection test of the ligand with the receptor from $O(n \cdot m)$ to $O(u \cdot m)$, where u is a constant, which is typically smaller than 20.

In addition to the usage of a 3-dimensional grid as accelerator, one can further speed-up the intersection tests by computing them in parallel. This is possible since intersection tests of spheres with the receptor molecule are independent of each other. On the CPU, OpenMP [175] is a good solution for parallelization. Even faster computations can be achieved by running the intersection tests on the GPU. In order to be platform independent, OpenCL [174] is suitable. Each thread computes the intersection of one sphere

with the receptor molecule. Here, the grid is stored using two arrays as described in Section 2.5 and illustrated in Figure 2.22.

9.3.2 Distance Field Updates

The distance field updates are the most expensive part of the algorithm. Recall that distance field updates are needed for two types of instance, that is, for the maximal ligand bounding sphere and for the ligand conformations with different orientations. If any of these instances is placed at a particular sample position, updating the distance field means that for each point in the distance field, the algorithm needs to compute the distance to the boundary of this instance. For the maximal ligand bounding sphere, this boundary is the sphere surface. For a ligand conformation, this boundary is the van der Waals surface of the ligand atoms, which is the surface enclosing all ligand atom spheres.

Since the distance field defining the LES is negative inside the LES and positive outside, the signs of the distances to the instance boundaries need to have inverted signs. Hence, the distance of a point to the instance is positive if the point lies inside the instance and negative if it lies outside. Recall that the distance field is initialized with the negative maximal atom radius of the receptor. For computing the correct distance field, an initialization with $-\infty$ is required. However, since the distance field itself is not of interest but the implicit surface defined by $d_{r,l}(p) = 0$, only the grid points close to the LES need to have the correct values. For all other grid points, it suffices to have the correct distance sign. Once, the distance of a particular grid point to the instance boundary is computed, the distance is updated only if the new distance value is larger than the current one. That is, the values of the distance field only increase.

Now, if s is the number of grid points and the algorithm updates the distance field in the naïve way, that is, all grid points for each instance at all sample position, the running time is $O(s^2)$. Even for medium-sized grids, this would result in very long computation times. Hence, in the following, two ways are described to reduce this running time.

Local Distance Fields

The first observation is that it is not necessary to update the whole distance field, but only part of it that is in the local neighborhood of the sample point. Here, the size of the neighborhood is defined by the particular instance. In fact, it is sufficient to consider all those grid points $p \in \mathbb{R}^3$ that are inside the axis-aligned bounding box of the instance plus those points within a distance $\|p - p_{bb}\|_\infty \leq g$ to the closest point $p_{bb} \in \mathbb{R}^3$ of the bounding box (Figure 9.5). Thus, the instance will be completely surrounded by grid points with negative distance to the instance.

Moreover, the distances can be precomputed for the local neighborhood of an instance, because the grid points of the distance field are the same points as the sample points of the instances. To do so, the algorithm computes local distance fields placed in the origin with the same grid spacing g. This is done for each instance, that is, for the maximal ligand bounding sphere and all ligand conformations and orientations (Figure 9.5). Then, during an update step, the local distance field only needs to be moved to the current position and the algorithm compares the values of the global distance field with those of the local distance field directly. The maximum of these two values determines the new value.

Since the average size $\tilde{s}$ of the local distance field is usually much smaller than s, the running time reduces from s^2 to $\tilde{s} \cdot s$. However, since $\tilde{s}$ is a fixed fraction of s, that is, $\tilde{s} = \alpha s$, where α is constant, the complexity of the algorithm remains $O(s^2)$.

Use of KD-Tree

A second observation can be used to further reduce the number of grid points that need to be updated: The only grid points defining the implicit surface $d_{r,l}(p) = 0$ are the ones in whose neighborhood the sign of the distance values changes. Furthermore, recall that the distance values only increase over the duration of the algorithm. Hence, if a grid point p has a positive value and all of its 26 neighbors also have positive distance values, p will not contribute to the definition of the implicit surface. Hence, the algorithm does not need to consider p any further.

For this purpose, a KD-tree is maintained that only contains those sample points that possibly need further updates. Using the KD-tree, these points can be quickly queried. Furthermore, dynamical updates of the KD-tree remove all samples that become positive and also have only positive neighbors. Thus, if an instance is placed at a particular sample point T, the KD-tree is used to quickly identify the grid points in the local neighborhood of T with respect to the instance size and only those points are updated if needed.

Parallel Distance Updates on the GPU

While the KD-tree is used to accelerate the CPU implementation, for the GPU a parallel distance field update was implemented. For this, the OpenCL kernel receives three inputs: the overall distance field, the local distance field, and a set of valid ligand positions for this field. Each thread corresponds to one sample point of the ligand distance field. The thread iterates over the ligand positions and detects in each step the corresponding sample point in the distance field. If the value in the distance field is smaller than the value in the ligand distance field, it is replaced by the new value. Note that this can lead to concurrent accesses of different threads on the same sample point in the distance field. However, as mentioned before, the values in the distance function can only increase. Thus, if the OpenCL kernel is executed several

times, the distance values converge to the correct result. To ensure the correct result, a single boolean variable is used, which is initialized with 'false' before each call of the kernel. If during the call a distance value has changed, the variable is set to 'true'. The kernel is called again as long as the variable is 'true'. Thus, the kernel is usually executed at least two times and in practice on average three to four times.

9.4 Visualization

The results of the above algorithms are a discrete scalar field describing the distance function in the local vicinity of the LES, and the cavities given by all their valid ligand states. In this section, a short description is given about how to render the LES and the cavities.

The LES can be visualized either by extracting a triangular mesh with the marching cubes algorithm [154] or by direct isosurface ray casting [83]. With todays GPU implementations of marching cubes, the surface can be generated nearly as fast as with direct ray casting. With some additional effort, the triangular surface can be colored according to arbitrary molecular properties. It can also be used to measure the area of the surface and its enclosed volume. Since the discrete distance function describing the LES is a signed distance function which is positive outside the LES and negative inside, ray casting the LES can be further accelerated. Outside the LES, the distance in the distance field is always smaller or equal to the minimal Euclidean distance to the LES. Hence, instead of using a constant step size for the direct isosurface ray casting, the algorithm can directly use the distance in the discrete field. This is similar to the classical sphere tracing by Hart [86], which is usually faster than a ray casting with constant step size.

The cavities can be visualized both by its cores and their surfaces. To render their surfaces, for each cavity a distance field can be constructed similar to the LES distance field, which is then rendered in the same way (Figure 9.7). The cavity cores are visualized by placing at each core sample position a small sphere (Figure 9.6). For the sphere rendering, modern GPU-based ray casting is used as presented in Chapter 3. The depth perception of all representations can be improved by ambient occlusion, silhouettes, or surface cuts.

9.5 Results

All results have been produced on an Intel Xeon X5650 E5540 2.66 GHz with 6 cores and an Nvidia GeForce GTX 680. For measuring the quality and performance, several molecules from the PDB [184] were used as well as two other data sets from cooperation partners.

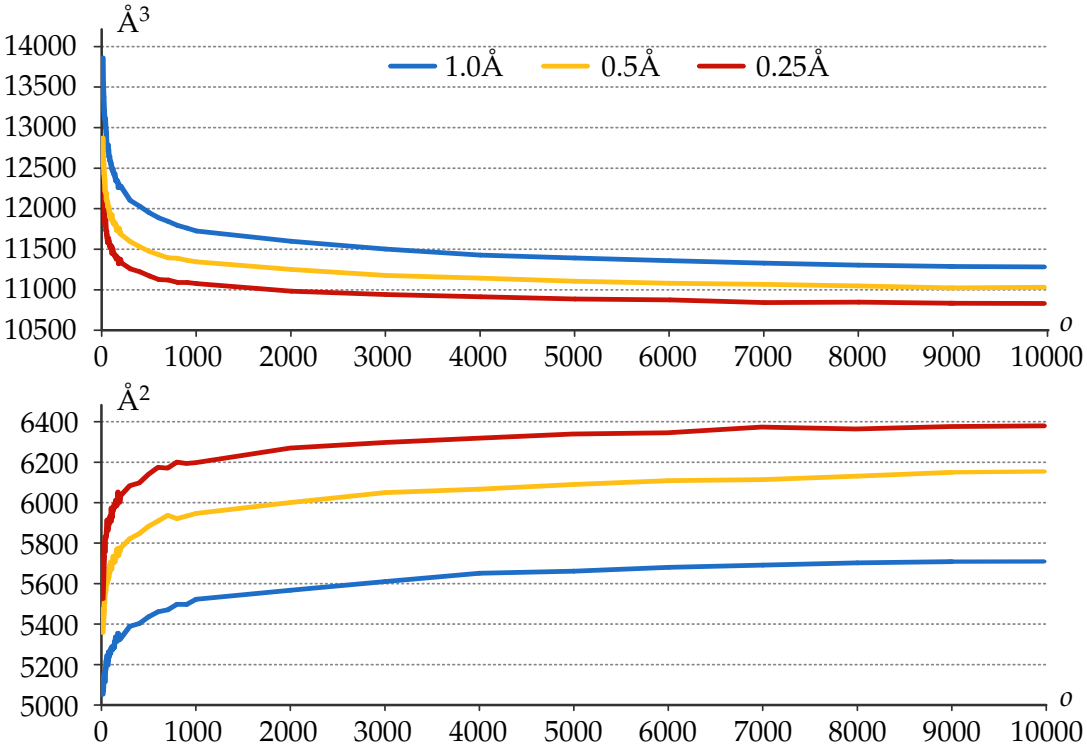


Figure 9.8: *For the graphs shown above, the LES of ketosteroid isomerase (pdb:1OGZ; also see Figure 9.1) with respect to the ligand equilenine was computed for different grid spacings and an increasing number of orientations. Note, however, that the used orientations were independent of each other. The graphs in the two images plot the enclosed volume (top) of the LES and the surface area (bottom) of the LES against the number of ligand rotations used for computing the LES.*

9.5.1 Parameters

Although the surface definition is independent of parameters, the algorithm requires parameters due to the discretization. The two parameters used for the LES computation are the grid spacing g and the number of orientations o. In Figure 9.8, the enclosed volume and the surface area of the LES is plotted depending on the number of orientations. For this, ketosteroid isomerase with ligand equilenine was used (*pdb:1OGZ*, Figure 9.1). Between 10 and 200 orientations the LES was computed by increasing the number of orientations by 5 in each step. From 200 to 1 000 orientations, the number of orientations was increased by 100 each, and from 1 000 to 10 000 orientations, a step size of 1 000 orientations was used. For each number of orientations, the triangular mesh of the LES was generated and the surface area and volume were computed. One can observe, that the main changes in surface area and volume occur between 10 and 1 000 orientations. Furthermore, it can be observed that for a smaller grid spacing the changes are more rapid than for larger grid spacings.

In most of the following experiments, 200 or fewer orientations were used. Only in the case of the HIV-protease with inhibitor amprenavir the algorithm

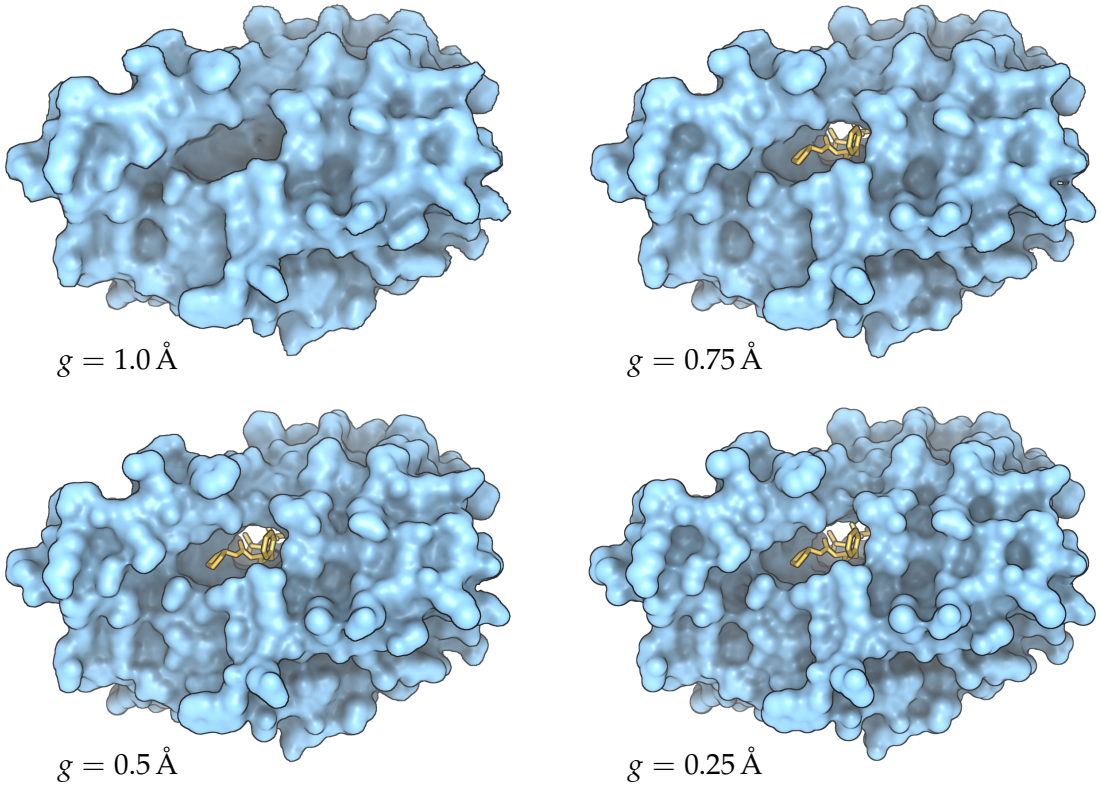


Figure 9.9: *LES of HIV protease (pdb:1HPV) for ligand amprenavir (yellow) with different grid resolutions. For the computation of the LES, four ligand conformations with 1 000 orientations were considered. For a grid spacing of 1.0 Å, the ligand binding site was not found.*

was not able to find the binding site with this number of orientations. With 1 000 orientations, however, it found the binding site even with a grid spacing of 0.75 Å (Figure 9.9).

The cavity analysis only needs one parameter, which determines when a sample point is considered to lie outside the boundary of the molecule. To determine the degree of burriedness, ambient occlusion was used by casting a fixed number of 100 rays. The user can then set a threshold at which ambient occlusion value a sample point is considered to lie outside the molecule.

9.5.2 Performance

Let $s \in \mathbb{N}$ be the number of sample positions, which is defined by the grid spacing g and the size of the receptor. Note that s grows cubically when linearly decreasing g. Because of the grid data structure, the cost for each intersection test with the receptor molecule becomes constant. Hence, the intersection tests have a complexity of $O(c \cdot o \cdot s)$. An upper bound for the number of values in the local distance field of the ligand is $\alpha \cdot s$, where α is the ratio of the volume of the local field and the overall distance field. Hence the updates of the distance field have a complexity of $O(c \cdot o \cdot \alpha \cdot s^2)$. The k-means

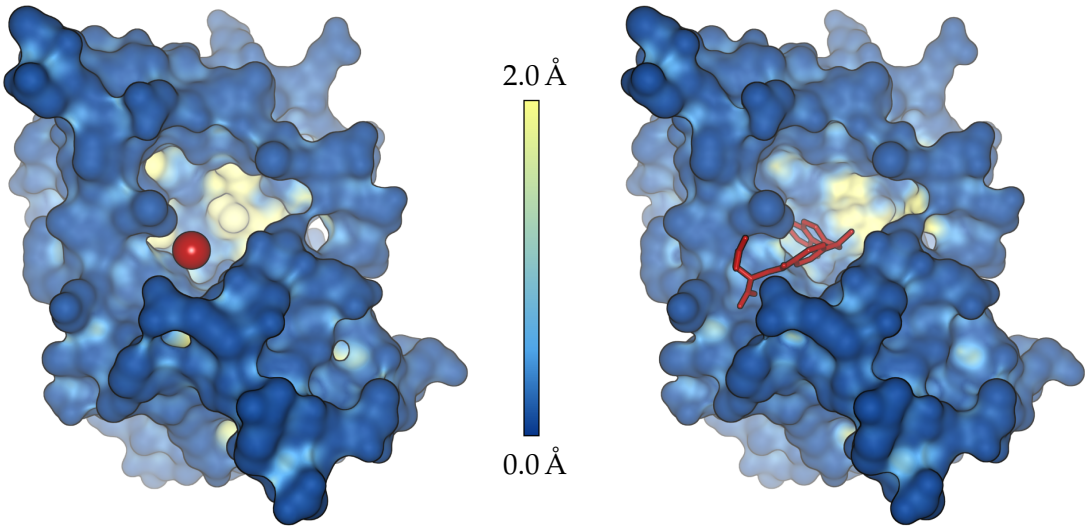


Figure 9.10: *Difference between the SES (left) and the LES (right) for an enzyme (pdb:4DFR). The LES was computed for methotrexate (red). Each surface point is colored according to its minimal distance to the other surface.*

clustering is the most expensive part for the computation of the orientations. It grows polynomially with o. However, compared to the rest of the algorithm, in practice the computation of the orientations is negligible.

The maximal memory requirements for the algorithm are given by the following data structures. The global distance field with a space complexity of $O(s)$, the local distance field with $O(\alpha \cdot s)$, the grid data structure of the receptor molecule with $O(n)$, the KD-tree with a worst case complexity of $O(s)$, and the ligand orientations at all sample points with a complexity of $O(s \cdot o \cdot c)$. For the intersection tests on the GPU, the grid of the receptor molecule and a set of spheres are required. For the distance updates on the GPU, the global and local distance fields and a set of update positions given by sample indices are required. Based on several performance tests, for each call of an OpenCL kernel, the number of spheres for the intersection tests was set to 65 536 and the number of positions for the distance updates to 1 024.

The most amount of memory is required to store the valid ligand orientations at each sample position (Table 9.1, bottom). For the nanotransporter and morphine, this structure requires approximately 50 MB. For 1HPV and amprenavir with a grid spacing of 0.25 Å, approximately 3 GB are required. However, this memory is only temporarily needed if cavities are computed. The largest memory consumption on the GPU is due to the global distance field. For the nanotransporter and morphine, this field requires approximately 25 MB. For 1HPV and amprenavir with a grid spacing of 0.25 Å, approximately 66 MB are required. Detailed memory requirements are given in Table 9.1, bottom.

processor	receptor (#atoms)	ligand (#atoms)	g (in Å)	o	phase I IT	phase I DU	phase II IT	phase II DU	LES	boundary detection	core computation	cavity surface	overall
CPU	1GRM (272)	water (3)	0.25	200	0.2	0.7	32.8	114.7	**148.4**	11.5	0.1	14.1	**174**
GPU	1GRM (272)	water (3)	0.25	200	0.1	0.2	4.6	16.2	**21.1**	0.6	0.1	1.5	**23**
CPU*	1OGZ (944)	equilenine (20)	0.5	200	0.4	5.6	265.4	1 057.1	**1 328.5**	16.3	0.1	3.5	**1 348**
CPU	1OGZ (944)	equilenine (20)	0.5	200	0.4	5.6	255.4	383.1	**644.4**	15.7	0.1	3.6	**664**
GPU	1OGZ (944)	equilenine (20)	0.5	200	0.1	0.4	29.5	50.9	**80.9**	0.7	0.1	0.5	**82**
CPU	nanotrans. (16 487)	water (3)	0.5	200	0.4	1.6	87.7	297.7	**387.3**	125.2	1.6	128.0	**642**
CPU	nanotrans. (16 487)	sulfate (5)	0.5	200	0.5	3.3	159.2	557.1	**720.1**	122.7	0.9	236.1	**1 080**
CPU	nanotrans. (16 487)	morphine (40)	0.5	200	2.8	13.1	2 175.6	762.4	**2 955.1**	131.7	0.4	533.0	**3 620**
GPU	nanotrans. (16 487)	water (3)	0.5	200	0.4	1.2	15.8	51.9	**69.3**	4.5	1.6	11.7	**87**
GPU	nanotrans. (16 487)	sulfate (5)	0.5	200	0.4	1.2	18.2	58.5	**78.3**	3.9	0.9	20.4	**104**
GPU	nanotrans. (16 487)	morphine (40)	0.5	200	0.8	0.9	367.1	139.2	**508.1**	4.2	0.4	82.4	**595**
GPU	1HPV (1 516)	amprenavir (35)	1.0	4 000	0.1	0.1	345.2	152.7	**498.1**	0.3	0.1	0.1	**499**
GPU	1HPV (1 516)	amprenavir (35)	0.5	4 000	0.3	0.9	2 436.7	1 970.2	**4 408.1**	1.7	0.1	0.2	**4 410**
GPU	1HPV (1 516)	amprenavir (35)	0.25	4 000	1.2	21.9	18 540.9	66 939.2	**85 503.2**	11.8	0.5	6.3	**85 522**

System: Intel Xeon X5650 E5540 2.66 GHz with an Nvidia GeForce GTX 680.

receptor (#atoms)	ligand (#atoms)	g (in Å)	o	global distance field (in MB)	local distance field (in KB)	grid (in KB)	kd-tree (in MB)	valid ligand states (in MB)	LES surface mesh (in MB)
1GRM (272)	water (3)	0.25	200	4.5	13.2	50.8	2.1	5.0	**2.1**
1OGZ (944)	equilenine (20)	0.5	200	5.3	76.9	127.9	1.4	9.0	**1.1**
nanotrans. (16 482)	water (3)	0.5	200	19.3	2.8	2 381.3	8.7	15.7	**13.1**
nanotrans. (16 482)	sulfate (5)	0.5	200	20.6	5.2	2 381.3	10.3	13.3	**11.8**
nanotrans. (16 482)	morphine (40)	0.5	200	24.9	47.5	2 381.3	12.5	49.7	**8.2**
1HPV (1 516)	amprenavir (35)	1.0	4 000	1.1	26.8	220.1	0.3	46.9	**0.4**
1HPV (1 516)	amprenavir (35)	0.5	4 000	8.5	197.9	220.1	2.2	383.9	**1.5**
1HPV (1 516)	amprenavir (35)	0.25	4 000	66.2	1 519.6	220.1	16.3	3 106.9	**6.3**

Table 9.1: *Running times in seconds (top) and memory requirements (bottom) for LES and cavity computation for different receptors, ligands, grid spacings (g) and number of orientations (o) both on CPU (*without KD-tree) and GPU. Phases I and II are split into the two most time-consuming steps: intersection tests (IT) and distance updates (DU). Note that for 1HPV, four conformations of amprenavir were used with 1 000 orientations each. Note that the storage of valid ligand states is not necessary for the LES computation, but temporarily it is used to extract the cavities. The LES surface mesh was extracted from the global distance field by using the marching cubes algorithm.*

In phase I all sample positions are detected that have to be investigated for each ligand conformation and orientation. This preprocessing accelerates the algorithm, because many of the sample positions can be ignored in phase II. The amount of acceleration depends mainly on the size of the ligand and the ratio between this size and the size of the receptor, but also on the geometrical complexity of the receptor. For water, which has a bounding radius of 1.52 Å, approximately 90 % of the sample positions can be ignored in phase II. For morphine, with a bounding radius of 5.44 Å, still around 75 % of the sample positions can be ignored and for fentanyl the bounding radius of which is 8.47 Å, approximately 68 % can be ignored. For all tested ligands, on average the number of sample points was higher than 60 %. Note that the percentages for each ligand were averaged over a couple of receptors of different size, ranging from 272 atoms to 58 870 atoms.

Detailed timings for all parts of the algorithm are given in Table 9.1, top. In all examples of common receptor-ligand systems the computations could be accelerated by a factor between 6 and 10 using the GPU compared to the CPU with KD-tree. On the CPU, the use of the KD-tree accelerated the computation by a factor of at least two. Furthermore, the intersection tests can be slower than the distance updates, although the complexity of the distance updates is worse (see, for example, 1HPV or the nanotransporter with morphine in Table 9.1).

While the SES can be computed interactively, the computation of the LES usually takes several minutes up to hours depending on the grid spacing and the number of orientations (Table 9.1, top). On the other hand, the LES better reflects the actual accessibility of a certain ligand, which can be seen in Figure 9.1, 9.6, and 9.10.

9.6 Feedback by Domain Scientists

Apart from an evaluation in terms of computational complexity and performance, a small survey was carried out about the usability of the proposed new molecular surface by performing a *structured interview* about potential advantages and disadvantages of the LES. To do so, four senior experts from different labs were interviewed, working in molecular dynamics simulation and dealing with a wide range of applications. In the following the most important statements were summerized.

Advantages: All collaborators agreed that the LES provides valuable ligand-specific information. In particular, the LES can help identify binding sites that have so far been unknown from experimental data. Furthermore, the LES allows one to discard cavities that are poor candidates as hosts for a ligand. Thus, more expensive methods computing the binding free energies or molecular dynamics simulations using force fields can be applied more effectively. If a protein can bind different ligands, the LES might also enable the identification of different binding sites for different ligands. The pure

visualization of the LES is of interest, because in many applications of molecular modeling, the chemical intuition of the observer is an important addition to fully automatic methods. This intuition is even further supported by the identified cavities.

Disadvantages: If both the protein and the ligand are highly flexible, the computation of the LES is expensive. In this case it might be favorable to use the SES to identify binding sites and cavities and compute the LES only for a few time steps or locally around the binding site.

9.7 Discussion

The most crucial parameter of the algorithm is the number of orientations. With the plot in Figure 9.8 of the surface area and the enclosed volume, an intuition of the quality of the result depending on the number of orientations is given. It is clear that both curves will converge, but the plots indicate how fast this convergence will be. What can be clearly seen is that the largest changes occur between 10 and 1 000 orientations. This suggests that with fewer than 1 000 orientations the LES can be very well approximated. For most tested data sets 200 or even fewer orientations were enough to reproduce most known cavities. In rare cases, however, it might be necessary to use more than this number or even more than 1 000 orientations. Hence, if the running time is of minor importance, $500 - 1\,000$ orientations seem suitable.

More important in terms of running time is the choice of the grid spacing. The running time grows quadratically with the number of grid points, which again grows inversely cubically with the grid spacing. For grid-based cavity analysis, commonly a minimal grid spacing of 0.5 Å is used. While a grid spacing of 0.25 Å is clearly favorable in terms of precision, it requires approximately a 28-fold running time (Table 9.1). Furthermore, in all experiments, a grid spacing of 0.5 Å was sufficient to find the known binding sites. Hence, a grid spacing of 0.5 Å seems to be suitable.

If the ligand binding site is very tight and the flexibility of the ligand is high, it might be necessary to use a large number of conformations. Since the running time depends linearly on the number of conformations, this is expensive. In this case, it might be possible to reduce the complexity by taking into account the redundancy in the conformations.

9.8 Potential Extensions

As described above, with a minor extension of the algorithm, the whole cavity structure of the molecule with respect to the ligand can be computed with almost no run-time overhead. In addition to the positions, also possible orientations and conformations of the ligand are obtained. This information might be exploited to steer molecular docking simulations, where the ligand binding

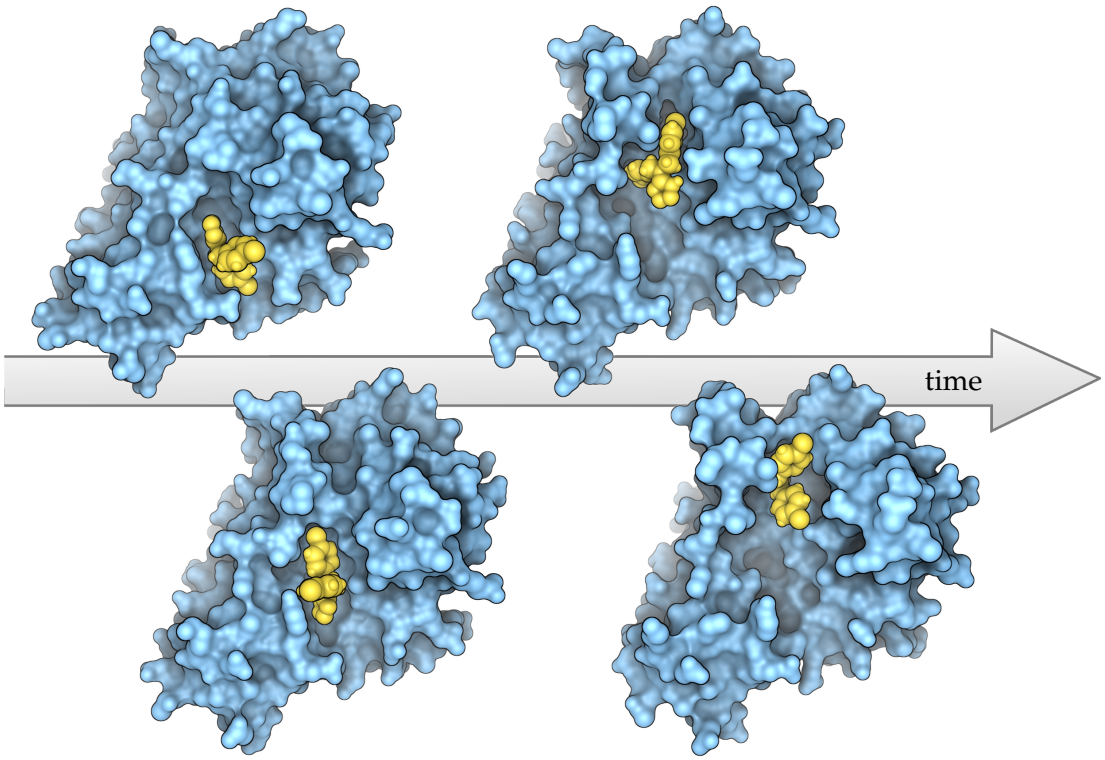


Figure 9.11: *LES (blue) depicted for four selected time steps of a simulation trajectory of β-lactamase (data courtesy of Gregory L. Bowman and Philipp L. Geissler [25]). During the simulation, the active site of the β-lactamase opens. The sequence of images shows the computed positions of the CBT ligand (yellow) closest to the active site.*

path into the active site is detected. On the other side one can use these information to compute molecular paths from dynamic trajectories of the receptor molecule as can be experimentally seen in Figure 9.11.

It is obvious that with the calculation of the LES a new computational challenge is associated. In the following some ideas are proposed on how the computation could be improved and extended.

First, instead of a fixed user-defined number of orientations, it would be favorable to avoid a fixed number by measuring the error rate between increasing numbers of orientations. For this, the algorithm might start with a few orientations and subsequently add more orientations while at the same time measuring the rate of change of the volume. As long as the change is larger than a predefined fraction, the algorithm would continue adding more orientations. This strategy, however, requires a suitable way to successively pick new orientations from a large set of orientations such that the distances between neighbored orientations remains similar. It is not obvious how such a selection can be done most efficiently.

Currently, each sample position with at least one valid ligand state belongs to exactly one cavity core. However, it is possible that the ligand cannot continuously change from one valid state to another valid state at the same

sample position without intersecting the receptor molecule. Hence, the cavity cores might have to be split at such positions, which also means that a sampling position can belong to different cavities and, thus, that the cores can overlap. In order to handle such situations, it is necessary to define valid changes of orientations and conformations within a sample position.

Despite the algorithmic optimizations, the running time of the algorithm can still be long, particularly for large molecules and a small grid spacing. In terms of cavity analysis, one is often only interested in cavities existing in certain regions. Thus, it might be sensible to restrict the cavity computation to user-defined regions. Furthermore, a combination of the LES approach with one that approximates the ligand by a sphere, such as the molecular path computation described in Chapter 6, might significantly speed up the computation.

Additionally, one could investigate whether the use of tetrahedral meshes with the same number of grid points as for hexahedral meshes improve the surface quality and leads to a faster convergence.

The high memory requirements to store the valid and invalid ligand states could be further reduced by applying compression algorithms to the bit arrays. Especially, algorithms that come with a fast decompression are of high interest to handle further tasks that depend on this data. Two promising techniques are the LZO [159] and the LZ4 [158] approaches.

The focus of this chapter lies on the definition and computation of a purely geometry-based representation of the accessibility of a receptor with respect to a ligand molecule. The definition of the LES could be further extended by considering physico-chemical properties, like the binding affinity of the ligand. By considering only those positions and orientations for which the binding affinity is high, the LES might even better reflect the true accessibility of the ligand.

10 Conclusion and Outlook

10.1 Conclusion

In this thesis, improvements for the visualization of molecular surfaces as well as for the detection and analyzes of cavities were presented. For the smooth surfaces, that is the solvent excluded surface (SES) and the molecular skin surface (MSS), the computation of the surface and its rendering were accelerated in order to apply these models to molecular dynamics data of typical proteins and enzymes. This approach is still one of the fastest ways to render the SES and MSS with a very high visual quality. Concerning the van der Waals surface, the focus was on the size of the molecular data. The presented technique allows one to interactively visualize static biological data sets with up to several billion atoms. It was the first approach to visualize cellular data and atom probe tomography samples on atomic resolution bridging five orders of magnitude in length scale. In the field of cavity analysis, a Voronoi-based technique was presented that comes with a very high geometrical accuracy while keeping the computation fast enough to apply it to molecular dynamics data sets. To investigate topological changes of the cavities, a new tracing was proposed. In addition to the computation, the cavity analysis includes several visualization techniques. Especially the novel lighting of cavities and the dynamic cavity visualization techniques are of high interest for biophysicists. Finally, a new molecular surface model was presented, called ligand excluded surface (LES). In contrast to the SES, the LES considers the full van der Waals surface of the ligand instead of approximating it by a single probe sphere. Thus, it better reflects the accessibility for a certain ligand. Additionally, the proposed algorithm for the computation of the LES detects the cavity structure. With all the proposed techniques, molecular visualization based

on hard spheres was improved in several directions, such as speed, size, or accuracy.

10.2 Outlook

Even though substantial improvements concerning speed were achieved in this thesis, it certainly is possible to further accelerate the proposed molecular visualization and analysis techniques, either by improving the existing algorithms or by developing new faster techniques. This is important to quickly investigate molecular data in real-time but also to deal with the increasing size of the data. Especially in the molecular dynamics field, scientists are working more and more on multi-scale simulations and analyzes for both, time and size. This requires corresponding visualization tools to investigate the data. Parallelization plays an important role for these developments in order to get the full performance of new hardware systems. While acceleration of existing techniques and development of multi-scale visualizations is a demanding and interesting research field, there is another important direction in molecular visualization that seems to be even more challenging. The goal of this direction is to improve and develop visualization techniques that better reflect the physical reality. A first step into this direction was done with the definition and computation of the ligand excluded surface. However, the whole visualization of molecules based on hard atom spheres indeed is questionable for some applications, especially if chemical binding processes are investigated. For such purposes, visualization models based on quantum mechanics are required. This field has been rarely investigated in the past and often just scalar fields are derived from the simulation data sets that are then visualized using classical isosurface rendering or volume rendering techniques. It will be a demanding and interesting task to develop tools that bring forward molecular investigations based on quantum mechanics.

Bibliography

[1] N. Akkiraju and H. Edelsbrunner. Triangulating the surface of a molecule. *Discrete Appl. Math.*, 71(1-3):5–22, 1996.

[2] J. Amanatides and A. Woo. A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics '87*, pages 1–10, 1987.

[3] J. An, M. Totrov, and R. Abagyan. Pocketome via comprehensive identification and classification of ligand binding envelopes. *Mol. Cell. Proteomics*, 4(6):752–761, 2005.

[4] M. Andersson, E. Malmerberg, S. Westenhoff, and G. Katona et al. Structural Dynamics of Light-Driven Proton Pumps Original Research Article Structure. *Structure*, 17(9):1265–1275, 2009.

[5] W. Arndt, T. M. Asbury, W. J. Zheng, M. Mitman, and J. Tang. Genome3D: A viewer-model framework for integrating and visualizing multi-scale epigenomic information within a three-dimensional genome. In *BIBM Workshops*, pages 936–938, 2011.

[6] J. Arvo. A Simple Method for Box-sphere Intersection Testing. In *Graphics Gems*, pages 335–339. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

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

[8] F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013.

[9] C. Bajaj, A. Gillette, and S. Goswami. Topology Based Selection and Curation of Level Sets. In *Topology-Based Methods in Visualization II*, Mathematics and Visualization, pages 45–58. Springer Berlin Heidelberg, 2009.

[10] C. Bajaj, H. Y. Lee, R. Merkert, and V. Pascucci. NURBS based B-rep models for macromolecules and their properties. In *SMA '97: Proceedings of the fourth ACM symposium on Solid modeling and applications*, pages 217–228, New York, NY, USA, 1997. ACM.

[11] C. Bajaj, V. Pascucci, A. Shamir, R. Holt, and A. Netravali. Multiresolution molecular shapes, 1999.

[12] C. Bajaj, V. Pascucci, A. Shamir, R. J. Holt, and A. N. Netravali. Dynamic maintenance and visualization of molecular surfaces. *Discrete Appl. Math.*, 127(1):23–51, 2003.

[13] S. S. Batsanov. Van der Waals Radii of Elements. *Inorganic Materials*, 37(9):871–885, 2001.

[14] S. S. Batsanov. Thermodynamic determination of van der Waals radii of metals. *Journal of Molecular Structure*, 990(1):63–66, 2011.

[15] L. Bavoil, M. Sainz, and R. Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 talks*, SIGGRAPH '08, New York, NY, USA, 2008. ACM.

[16] J. Berg, J. Tymoczko, and L. Stryer. *Biochemistry, Fifth Edition*. W. H. Freeman, 2002.

[17] J. F. Blinn. Models of Light Reflection for Computer Synthesized Pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198, July 1977.

[18] J. F. Blinn. A Generalization of Algebraic Surface Drawing. *ACM Trans. Graph.*, 1(3):235–256, July 1982.

[19] J. Bloomenthal and B. Wyvill. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[20] N. Bohr. On the constitution of atoms and molecules - parts I, II. *Phil. Mag.*, 26:1–25, 476–502, 1913.

[21] J.-D. Boissonnat and C. Delage. Convex Hull and Voronoi Diagram of Additively Weighted Points. In *ESA*, volume 3669 of *Lecture Notes in Computer Science*, pages 367–378. Springer, 2005.

[22] A.-N. Bondar, S. Fischer, and J. Smith. Water Pathways in the Bacteriorhodopsin Proton Pump. *J. Membr. Biol.*, 239:73–84, 2011.

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

[24] D. Borland. Ambient Occlusion Opacity Mapping for Visualization of Internal Molecular Structure. *Journal of WSCG*, 19(1):17–23, 2011.

[25] G. R. Bowman and P. L. Geissler. Equilibrium fluctuations of a single folded protein reveal a multitude of potential cryptic allosteric sites. *Proceedings of the National Academy of Sciences of the United States of America*, 109(29):11681–6, 2012.

[26] G. P. Brady Jr. and P. F. W. Stouten. Fast prediction and visualization of protein binding pockets with PASS. *Journal of Computer-Aided Molecular Design*, 14(4):383–401, 2000.

[27] W. L. Bragg. The Arrangement of Atoms in Crystals. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 40(236):169–189, 1920.

[28] J. Brezovsky, E. Chovancova, A. Gora, A. Pavelka, L. Biedermannova, and J. Damborsky. Software tools for identification, visualization and analysis of protein tunnels and channels. *Biotechnology Advances*, 31(1):38 – 49, 2013.

[29] V. E. Brimkov and R. P. Barneva. *Digital Geometry Algorithms: Theoretical Foundations and Applications to Computational Imaging*. Springer Publishing Company, Incorporated, 2012.

[30] M. Buehler. Nanomaterials: Strength in numbers. *Nature Nanotechnology*, 5(3):172–174, 2010.

[31] A. Bujotzek and M. Weber. Efficient Simulation of Ligand-Receptor Binding Processes Using the Conformation Dynamics Approach. *Journal of Bioinformatics and Computational Biology*, 7(5):811–831, 2009.

[32] M. P. Calace, T. Maiwald, and J. M. Thornton. PoreWalker: A novel tool for the identification and characterization of channels in transmembrane proteins from their three-dimensional structure. *PLoS Computational Biology*, 5(7), 2009.

[33] J. A. Capra, R. A. Laskowski, J. M. Thornton, M. Singh, and T. A. Funkhouser. Predicting Protein Ligand Binding Sites by Combining Evolutionary Sequence Conservation and 3D Structure. *PLoS Comput Biol*, 5(12):1–18, 2009.

[34] J. A. Capra and M. Singh. Predicting functionally important residues from sequence conservation. *Bioinformatics*, 23(15):1875–1882, 2007.

[35] S. L. Chan and E. O. Purisima. Molecular surface generation using marching tetrahedra. *Journal of Computational Chemistry*, 19(11):1268–1277, 1998.

[36] M. Chavent, B. Lévy, and B. Maigret. MetaMol: High quality visualization of Molecular Skin Surface. *Journal of Molecular Graphics and Modelling*, 27(2):1391–1398, 2008.

[37] H.-L. Cheng, T. K. Dey, H. Edelsbrunner, and J. Sullivan. Dynamic skin triangulation. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 47–56, Philadelphia, PA, USA, 2001.

[38] H.-L. Cheng and X. Shi. Guaranteed Quality Triangulation of Molecular Skin Surfaces. In *Proceedings of IEEE Visualization*, pages 481–488, 2004.

[39] H.-L. Cheng and X. Shi. Quality Mesh Generation for Molecular Skin Surfaces Using Restricted Union of Balls. In *Proceedings of IEEE Visualization*, pages 399–405, 2005.

[40] H.-L. Cheng and X. Shi. Quality Mesh Generation for Molecular Skin Surfaces Using Restricted Union of Balls. *Computational Geometry*, 42(3):196–206, 2009.

[41] Y. Cho, D. Kim, H. Lee, J. Park, and D.-S. Kim. Reduction of the Search Space in the Edge-Tracing Algorithm for the Voronoi Diagram of 3D Balls. In *Computational Science and Its Applications - ICCSA 2006*, volume 3980 of *Lecture Notes in Computer Science*, pages 111–120. Springer, Berlin Heidelberg, 2006.

[42] E. Chovancova, A. Pavelka, P. Benes, O. Strnad, J. Brezovsky, B. Kozlikova, A. Gora, V. Sustr, M. Klvana, P. Medek, L. Biedermannova, J. Sochor, and J. Damborsky. CAVER 3.0: A Tool for the Analysis of Transport Pathways in Dynamic Protein Structures. *PLoS Comput Biol*, 8(10):1–12, 2012.

[43] R. G. Coleman and K. A. Sharp. Travel depth, a new shape descriptor for macromolecules: application to ligand binding. *Journal of molecular biology*, 362(3):441–458, September 2006.

[44] R. G. Coleman and K. A. Sharp. Finding and characterizing tunnels in macromolecules with application to ion channels and pores. *Biophys. J.*, 96:632–645, Oct. 2008.

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

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

[47] J. Cortes, S. Barbe, M. Erard, and T. Simeon. Encoding Molecular Motions in Voxel Maps. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8:557–563, March 2011.

[48] J. Cortes, T. Simon, V. R. de Angulo, D. Guieysse, M. Remaud-Simeon, and V. Tran. A Path Planning Approach for Computing Large-Amplitude Motions of Flexible Molecules, 2005.

[49] J. Dalton. *A New System of Chemical Philosophy, Part I and II*. Birkenstaff, London, 1808 and 1810.

[50] S. Decherchi and W. Rocchia. A general and Robust Ray-Casting-Based Algorithm for Triangulating Surfaces at the Nanoscale. *PLoS ONE*, 8(4):1–15, 04 2013.

[51] B. N. Delaunay. Sur la sphère vide. *Bulletin of Academy of Sciences of the USSR*, pages 793–800, 1934.

[52] N. Desdouits, M. Nilges, and A. Blondel. Principal Component Analysis reveals correlation of cavities evolution and functional motions in proteins. *J. Mol. Graph. Modell.*, 55:13–24, 2015.

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

[54] H. Edelsbrunner. Deformable Smooth Surface Design. *Discrete & Computational Geometry*, 21(1):87–115, 1999.

[55] H. Edelsbrunner, M. Face, P. Fu, and J. Liang. Measuring proteins and voids in proteins. In *Proc. 28th Ann. Hawaii Int. Conf. System Sciences*, pages 256–264, 1995.

[56] H. Edelsbrunner, M. Facello, and J. Liang. On the definition and the construction of pockets in macromolecules. *Biocomputing: Proceedings of the 1996 Pacific symposium*, pages 272 – 281, 1996.

[57] H. Edelsbrunner, M. Facello, and J. Liang. On the definition and the construction of pockets in macromolecules. *Discrete Applied Mathematics*, 88(1-3):83 – 102, 1998.

[58] H. Edelsbrunner and E. P. Mücke. Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms. *ACM Trans. Graph. (TOG)*, 9(1):66–104, 1990.

[59] H. Edelsbrunner and E. P. Mücke. Three-dimensional alpha shapes. *ACM Trans. Graph.*, 13:43–72, January 1994.

[60] F. Eisenhaber, P. Lijnzaad, P. Argos, C. Sander, and M. Scharf. The double cubic lattice method: Efficient approaches to numerical integration of surface area and volume and to dot surface contouring of molecular assemblies. *Journal of Computational Chemistry*, 16(3):273–284, 1995.

[61] T. Exner, M. Keil, G. Möckel, and J. Brickmann. Identification of Substrate Channels and Protein Cavities. *Journal of Molecular Modeling*, 4:340–343, 1998.

[62] M. Falk, M. Krone, and T. Ertl. Atomistic visualization of mesoscopic whole-cell simulations. In *Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 123–130, 2012.

[63] M. Falk, M. Krone, and T. Ertl. Atomistic Visualization of Mesoscopic Whole-Cell Simulations Using Ray-Casted Instancing. *Computer Graphics Forum*, 32(8):195–206, 2013.

[64] G. E. Farin. *Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide*. Academic Press, Inc., Orlando, FL, USA, 4th edition, 1996.

[65] K. Fischer. *Smallest enclosing balls of balls: Combinatorial structure & algorithms*. PhD thesis, ETH Zürich, 2005.

[66] R. Fraedrich, J. Schneider, and R. Westermann. Exploring the Millennium Run - Scalable Rendering of Large-Scale Cosmological Datasets. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1251–1258, 2009.

[67] D. Frenkel and B. Smit. *Understanding molecular simulation: from algorithms to applications*, volume 1. Academic press, 2001.

[68] S. Frey, T. Schlömer, S. Grottel, C. Dachsbacher, O. Deussen, and T. Ertl. Loose Capacity-Constrained Representatives for the Qualitative Visual Analysis in Molecular Dynamics. In *PacificVis*, pages 51–58, 2011.

[69] D. Frishman and P. Argos. Knowledge-based protein secondary structure assignment. *Proteins*, 23(4):566–579, Dec. 1995.

[70] M. L. Gavrilova and J. Rokne. Updating the topology of the dynamic Voronoi diagram for spheres in Euclidean d-dimensional space. *Comput. Aided Geom. Des.*, 20:231–242, July 2003.

[71] J. Giard, P. Rondao Alface, J.-L. Gala, and B. Macq. Fast Surface-Based Travel Depth Estimation Algorithm for Macromolecule Surface Shape Description. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 8(1):59–68, Jan. 2011.

[72] K. Gibson and H. Scheraga. Exact calculation of the volume and surface area of fused hard-sphere molecules with unequal atomic radii. *Molecular Physics*, 62(5):1247–1265, 1987.

[73] 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.

[74] J. A. Grant and B. T. Pickup. A Gaussian Description of Molecular Shape. *The Journal of Physical Chemistry*, 99(11):3503–3510, 1995.

[75] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 231–238, New York, NY, USA, 1993. ACM.

[76] J. Greer and B. Bush. Macromolecular Shape and Surface Maps by Solvent Exclusion. *Proceedings of the National Academy of Sciences USA*, 75:303–307, 1978.

[77] G. Grigoryan and P. Rheingans. Point-Based Probabilistic Surfaces to Show Surface Uncertainty. *IEEE Transactions on Visualization and Computer Graphics*, 10(5):564–573, Sept. 2004.

[78] S. Grottel, M. Krone, K. Scharnowski, and T. Ertl. Object-space ambient occlusion for molecular dynamics. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*, pages 209–216, Feb 2012.

[79] S. Grottel, G. Reina, C. Dachsbacher, and T. Ertl. Coherent Culling and Shading for Large Molecular Dynamics Visualization. In *Computer Graphics Forum (Proceedings of EUROVIS 2010)*, volume 29, pages 953–962, 2010.

[80] S. Grottel, G. Reina, and T. Ertl. Optimized Data Transfer for Time-dependent, GPU-based Glyphs. In *Proceedings of IEEE Pacific Visualization Symposium*, pages 65–72, 2009.

[81] S. Grudinin, G. Büldt, V. Gordeliy, and A. Baumgaertner. Water Molecules and Hydrogen-Bonded Networks in Bacteriorhodopsin – Molecular Dynamics Simulations of the Ground State and the M-Intermediate. *Biophys. J.*, 88(5):3252–3261, 2005.

[82] S. Gumhold. Splatting Illuminated Ellipsoids with Depth Correction. In *Proceedings of Vision, Modeling, and Visualization (VMV)*, pages 245–252, 2003.

[83] M. Hadwiger, C. Sigg, H. Scharsach, K. Bühler, and M. H. Gross. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[84] M. Haranczyk and J. A. Sethian. Navigating molecular worms inside chemical labyrinths. *Proceedings of the National Academy of Sciences*, 106(51):21472–21477, 2009.

[85] S. Hargreaves. Deferred shading. Game Developers Conference (GDC) talks, 2004.

[86] J. C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[87] M. Heinig and D. Frishman. STRIDE: a web server for secondary structure assignment from known atomic coordinates of proteins. *Nucleic Acids Research*, 32(suppl 2):W500–W502, 2004.

[88] W. Heisenberg. Quantenmechanik. *Naturwiss.*, 14(45):989–994, 1926.

[89] M. Hendlich, F. Rippmann, and G. Barnickel. LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins. *Journal of Molecular Graphics and Modelling*, 15(6):359 – 363, 1997.

[90] D. Herbison-Evans. *Graphics Gems V*, chapter Solving Quartics and Cubics for Graphics, pages 1–15. Academic Press, 1995.

[91] M. Hernandez, D. Ghersi, and R. Sanchez. SITEHOUND-web: a server for ligand binding site identification in protein structures. *Nucleic Acids Research*, 37(suppl 2):W413–W416, 2009.

[92] J. Herzfeld and J. Lansing. Magnetic resonance studies of the bacteriorhodopsin pump cycle. *Annu. Rev. Biophys. Biomol. Struct.*, 31, 2002.

[93] B. K. Ho and F. Gruswitz. HOLLOW: Generating accurate representations of channel and interior surfaces in molecular structures. *BMC structural biology*, 8(1):49+, Nov. 2008.

[94] H. Hoppe. Progressive Meshes. In *SIGGRAPH96*, pages 99–108. ACM Press/ACM SIGGRAPH, 1996.

[95] J. Howard and A. Hyman. Growth, fluctuation and switching at microtubule plus ends. *Nature Reviews Molecular Cell Biology*, 10(8):569–74, 2009.

[96] B. Huang. MetaPocket: a meta approach to improve protein ligand binding site prediction. *Omics*, 13(4):325–330, 2009.

[97] B. Huang and M. Schroeder. LIGSITE^{csc}: predicting ligand binding sites using the Connolly surface and degree of conservation. *BMC Structural Biology*, 6(1):19, 2006.

[98] C. Huygens. *Traitıé de la lumière*. Pierre van der Aa, Leiden, 1690.

[99] E. Jardón-Valadez, A.-N. Bondar, and D. J. Tobias. Coupling of retinal, protein, and water dynamics in squid rhodopsin. *Biophys. J.*, 99(7):2200–2207, 2010.

[100] C. Jones and K.-L. Ma. Visualizing Flow Trajectories Using Locality-based Rendering and Warped Curve Plots. *IEEE Transactions on Visualization and Computer Graphics*, 16:1587–1594, November 2010.

[101] D. Juba and A. Varshney. Parallel, stochastic measurement of molecular surface area. *Journal of Molecular Graphics and Modelling*, 27(1):82 – 87, 2008.

[102] W. Kabsch and C. Sander. Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–2637, 1983.

[103] D. Kauker, M. Krone, A. Panagiotidis, G. Reina, and T. Ertl. Rendering Molecular Surfaces using Order-Independent Transparency. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV)*, volume 13, pages 33–40, 2013.

[104] T. Kawabata. Detection of multiscale pockets on protein surfaces using mathematical morphology. *Proteins: Structure, Function, and Bioinformatics*, 78(5):1195–1211, 2010.

[105] T. F. Kelly and M. K. Miller. Atom probe tomography. *Review of Scientific Instruments*, 78(3), 2007.

[106] I. Kepleri. *Strena, Seu de Nive Sexangula*. Francofurti ad Moenum, 1611.

[107] B. Kim, J. E. Lee, Y. J. Kim, and K.-J. Kim. GPU Accelerated Finding of Channels and Tunnels for a Protein Molecule. *Int. J. Parallel. Prog.*, 44(1):87–108, 2016.

[108] D. Kim and D.-S. Kim. Region-expansion for the Voronoi diagram of 3D spheres. *Comput. Aided Des.*, 38(5):417–430, 2006.

[109] 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.

[110] D.-S. Kim, Y. Cho, J.-K. Kim, and K. Sugihara. Tunnels and Voids in Molecules via Voronoi Diagrams and Beta-Complexes. In *Transactions on Computational Science XX*, volume 8110 of *Lecture Notes in Computer Science*, pages 92–111. Springer Berlin Heidelberg, 2013.

[111] D.-S. Kim, Y. Cho, and K. Sugihara. Quasi-worlds and quasi-operators on quasi-triangulations. *Comput. Aided Des.*, 42:874–888, October 2010.

[112] 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.

[113] J.-K. Kim, Y. Cho, M. Lee, R. A. Laskowski, S. E. Ryu, K. Sugihara, and D.-S. Kim. BetaCavityWeb: a webserver for molecular voids and channels. *Nucl. Acids Res.*, page gkv360, 2015.

[114] T. Klein and T. Ertl. Illustrating Magnetic Field Lines using a Discrete Particle Model. In *Proceedings of Vision, Modeling, and Visualization (VMV)*, pages 387–394, 2004.

[115] R. Klette and F. Wu. Multigrid convergence of surface approximations. Technical report, CITR, The University of Auckland, New Zealand, 1998.

[116] G. J. Kleywegt and T. A. Jones. Detection, delineation, measurement and display of cavities in macromolecular structures. *Acta Crystallographica Section D*, 50(2):178–185, Mar. 1994.

[117] B. Kozlikova, M. Krone, N. Lindow, M. Falk, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege. Visualization of Biomolecular Structures: State of the Art. In *Eurographics Conference on Visualization (EuroVis) - STARs*. The Eurographics Association, 2015.

[118] H. A. Kramers and H. Holst. *The atom and the Bohr theory of its structure*. Gyldendal, London, 1923.

[119] M. Krone, K. Bidmon, and T. Ertl. GPU-based Visualisation of Protein Secondary Structure. In *TPCG*, pages 115–122. Eurographics Association, 2008.

[120] M. Krone, K. Bidmon, and T. Ertl. Interactive Visualization of Molecular Surface Dynamics. *IEEE Trans. Vis. Comput. Graphics*, 15(6):1391–1398, 2009.

[121] M. Krone, C. Dachsbacher, and T. Ertl. Parallel Computation and Interactive Visualization of Time-varying Solvent Excluded Surfaces. In *International Conference On Bioinformatics and Computational Biology*, volume 2010, pages 402–405. ACM, 2010.

[122] M. Krone, M. Falk, S. Rehm, J. Pleiss, and T. Ertl. Interactive Exploration of Protein Cavities. *Comput. Graph. Forum*, 30(3):673–682, 2011.

[123] M. Krone, S. Grottel, and T. Ertl. Parallel Contour-Buildup Algorithm for the Molecular Surface. In *Proc. IEEE Symposium on Biological Data Visualization (BioVis 2011)*, pages 17–22, 2011.

[124] M. Krone, D. Kauker, G. Reina, and T. Ertl. Visual Analysis of Dynamic Protein Cavities and Binding Sites. In *Pacific Visualization Symposium (PacificVis), 2014 IEEE*, pages 301–305, March 2014.

[125] M. Krone, B. Kozlíková, N. Lindow, M. Baaden, D. Baum, J. Parulek, H.-C. Hege, and I. Viola. Visual Analysis of Biomolecular Cavities: State of the Art. *Computer Graphics Forum*, 35(3):527–551, 2016.

[126] M. Krone, G. Reina, C. Schulz, T. Kulschewski, J. Pleiss, and T. Ertl. Interactive Extraction and Tracking of Biomolecular Surface Features. *Computer Graphics Forum*, 32(3pt3):331–340, 2013.

[127] N. Kruithof and G. Vegter. Meshing skin surfaces with certified topology. *Computational Geometry*, 36(3):166–182, 2007.

[128] S. Laine and T. Karras. Efficient Sparse Voxel Octrees. *IEEE Trans. Vis. Comput. Graph.*, 17(8):1048–1059, 2011.

[129] R. Lakes. Materials with structural hierarchy. *Nature*, 361(6412):511–515, 1993.

[130] T. Larsson, T. Akenine-Möller, and E. Lengyel. On Faster Sphere-Box Overlap Testing. *Journal of Graphics, GPU, and Game Tools*, 12(1):3–8, 2007.

[131] R. A. Laskowski. SURFNET: A program for visualizing molecular surfaces, cavities, and intermolecular interactions. *Journal of Molecular Graphics*, 13(5):323 – 330, 1995.

[132] P. Laug and H. Borouchaki. Molecular Surface Modeling and Meshing. *Eng. Comput. (Lond.)*, 18(3):199–210, 2002.

[133] A. T. R. Laurie and R. M. Jackson. Q-SiteFinder: an energy-based method for the prediction of protein-ligand binding sites. *Bioinformatics*, 21(9):1908–1916, 2005.

[134] S. M. Lavalle. Rapidly-exploring random trees: A new tool for path planning, 1998.

[135] V. Le Guilloux, P. Schmidtke, and P. Tuffery. Fpocket: an open source platform for ligand pocket detection. *BMC bioinformatics*, 10(1):168, 2009.

[136] M. Le Muzic, L. Autin, J. Parulek, and I. Viola. CellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. In *Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 61–70. The Eurographics Association, Sept. 2015.

[137] M. Le Muzic, P. Mindek, J. Sorger, L. Autin, D. S. Goodsell, and I. Viola. Visibility Equalizer Cutaway Visualization of Mesoscopic Biological Models. *Computer Graphics Forum*, 35(3):161–170, 2016.

[138] M. Le Muzic, J. Parulek, A.-K. Stavrum, and I. Viola. Illustrative Visualization of Molecular Reactions using Omniscient Intelligence and Passive Agents. *Computer Graphics Forum*, 33(3):141–150, June 2014.

[139] B. Lee and F. M. Richards. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379–380, 1971.

[140] A. Leis, B. Rockel, L. Andrees, and W. Baumeister. Visualizing cells at the nanoscale. *Trends in Biochemical Sciences*, 34(2):60–70, 2009.

[141] D. G. Levitt and L. J. Banaszak. POCKET: A Computer Graphics Method for Identifying and Displaying Protein Cavities and Their Surrounding Amino Acids. *J. Mol. Graph.*, 10(4):229–234, Dec. 1992.

[142] X. Li, W. Wang, R. Martin, and A. Bowyer. Using low-discrepancy sequences and the crofton formula to compute surface areas of geometric models. *Computer-Aided Design*, 35(9):771–782, 2003.

[143] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, S. Subramaniam, et al. Analytical shape computation of macromolecules: I. Molecular area and volume through alpha shape. *Proteins Structure Function and Genetics*, 33(1):1–17, 1998.

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

[145] J. Liang, H. Edelsbrunner, S. Pamidghantam, and S. Subramaniam. Analytical method for molecular shapes: Area, volume, cavities, interface and pockets. In *BIOPHYSICAL JOURNAL*, volume 70 (2), pages A377–A377, 1996.

[146] J. Liang, H. Edelsbrunner, and C. Woodward. Anatomy of protein pockets and cavities: measurement of binding site geometry and implications for ligand design. *Protein science: a publication of the Protein Society*, 7(9):1884–1897, Sept. 1998.

[147] N. Lindow, D. Baum, A.-N. Bondar, and H.-C. Hege. Dynamic Channels in Biomolecular Systems: Path Analysis and Visualization. In *Proceedings of IEEE Symposium on Biological Data Visualization (biovis'12)*, pages 99 – 106, 2012.

[148] N. Lindow, D. Baum, A.-N. Bondar, and H.-C. Hege. Exploring cavity dynamics in biomolecular systems. *BMC Bioinformatics*, 14, 2013.

[149] N. Lindow, D. Baum, and H.-C. Hege. Voronoi-Based Extraction and Visualization of Molecular Paths. *IEEE Transactions on Visualization and Computer Graphics*, 17:2025–2034, 2011.

[150] N. Lindow, D. Baum, and H.-C. Hege. Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale. *Comp. Graph. Forum*, 31:1325–1334, 2012.

[151] N. Lindow, D. Baum, and H.-C. Hege. Ligand Excluded Surface: A New Type of Molecular Surface. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2486–2495, Dec 2014.

[152] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege. Accelerated Visualization of Dynamic Molecular Surfaces. *Comput. Graph. Forum*, 29(3):943–952, 2010.

[153] C. Loop and J. Blinn. Real-time GPU rendering of piecewise algebraic surfaces. *ACM Transactions on Graphics*, 25(3):664–670, 2006.

[154] W. E. Lorensen and H. E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM Siggraph Computer Graphics (Proc. of SIGGRAPH 87)*, 21(4):163–169, 1987.

[155] 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.

[156] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[157] T. Luft, C. Colditz, and O. Deussen. Image enhancement by unsharp masking the depth buffer. *ACM Transactions on Graphics*, 25(3):1206–1213, 2006.

[158] LZ4. http://www.lz4.org/.

[159] LZO. http://www.oberhumer.com/opensource/lzo/.

[160] M. H. Maeda and K. Kinoshita. Development of new indices to evaluate protein-protein interfaces: Assembling space volume, assembling space distance, and global shape descriptor. *Journal of Molecular Graphics and Modelling*, 27(6):706 – 711, 2009.

[161] M. Maňák and I. Kolingerová. Fast Discovery of Voronoi Vertices in the Construction of Voronoi Diagram of 3D Balls. In *Proceedings of the 2010 International Symposium on Voronoi Diagrams in Science and Engineering*, ISVD '10, pages 95–104, Washington, DC, USA, 2010. IEEE Computer Society.

[162] M. McGuire, M. Mara, and D. Luebke. Scalable Ambient Obscurance. In *High-Performance Graphics 2012*, June 2012.

[163] P. Medek, P. Beneš, and J. Sochor. Computation of tunnels in protein molecules using Delaunay triangulation. *Journal of WSCG, University of West Bohemia, Pilsen*, 15(1-3):107–114, 2007.

[164] 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*, pages 1676–1692, 2006.

[165] MegaMol. http://vis-web.informatik.uni-stuttgart.de/trac/megamol.

[166] J. Meyer-Spradow, T. Ropinski, J. Vahrenhold, and K. Hinrichs. K.H.: Illustrating Dynamics of Time-Varying Volume Datasets in Static Images. In *Proceedings of Vision, Modeling and Visualization 2006*, pages 333–340, 2006.

[167] M. Mittring. Finding Next Gen: CryEngine 2. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 97–121, New York, NY, USA, 2007. ACM.

[168] D. R. Musser. Introspective Sorting and Selection Algorithms. *Softw. Pract. Exper.*, 27(8):983–993, Aug. 1997.

[169] H. Niederreiter. Low-discrepancy and low-dispersion sequences. *Journal of Number Theory*, 30(1):51 – 70, 1988.

[170] K. Nomura, Y.-C. Chen, W. Weiqiang, R. K. Kalia, A. Nakano, P. Vashishta, and L. H. Yang. Interaction and coalescence of nanovoids and dynamic fracture in silica glass: multimillion-to-billion atom molecular dynamics simulations. *Journal of Physics D: Applied Physics*, 42(21):1–12, 2009.

[171] OpenGL. http://www.opengl.org/.

[172] K. Olechnovic, M. Margelevicius, and C. Venclovas. Voroprot: an interactive tool for the analysis and visualization of complex geometric features of protein structure. *Bioinformatics (Oxford, England)*, 27(5):723–724, March 2011.

[173] S. H. Oliveira, F. A. Ferraz, R. V. Honorato, J. Xavier-Neto, T. J. Sobreira, and P. S. de Oliveira. KVFinder: steered identification of protein cavities as a PyMOL plugin. *BMC Bioinformatics*, 15(1):1–8, 2014.

[174] OpenCL. http://www.khronos.org/opencl/.

[175] OpenMP. http://www.openmp.org/.

[176] J. Parsons, J. B. Holmes, J. M. Rojas, J. Tsai, and C. E. M. Strauss. Practical conversion from torsion space to Cartesian space for in silico protein synthesis. *Journal of Computational Chemistry*, 26(10):1063–1068, 2005.

[177] J. Parulek and A. Brambilla. Fast Blending Scheme for Molecular Surface Representation. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2653–2662, 2013.

[178] J. Parulek, C. Turkay, N. Reuter, and I. Viola. Implicit Surfaces for Interactive Graph Based Cavity Analysis of Molecular Simulations. In *2nd IEEE Symposium on Biological Data Visualization*, BioVis 2012, 2012.

[179] J. Parulek, C. Turkay, N. Reuter, and I. Viola. Visual cavity analysis in molecular simulations. *BMC Bioinformatics*, 14(19):1–15, 2013.

[180] J. Parulek and I. Viola. Implicit Representation of Molecular Surfaces. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis 2012)*, pages 217–224, March 2012.

[181] L. Pauling. *The Nature of the Chemical Bond and the Structure of Molecules and Crystals: An Introduction to Modern Structural Chemistry*. 429 p., Cornell University Press, Ithaca, NY, 1939.

[182] L. Pauling and R. B. Corey. Configurations of Polypeptide Chains With Favored Orientations Around Single Bonds Two New Pleated Sheets. *Proceedings of the National Academy of Sciences*, 37(11):729–740, 1951.

[183] L. Pauling, R. B. Corey, and H. R. Branson. The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain. *Proceedings of the National Academy of Sciences*, 37(4):205–211, 1951.

[184] Protein Data Bank. http://www.pdb.org.

[185] L. H. Pearl and A. Honegger. Generation of molecular surfaces for graphic display. *Journal of Molecular Graphics*, 1(1):9–12, 1983.

[186] G. Perrot, B. Cheng, K. D. Gibson, J. Vila, K. A. Palmer, A. Nayeem, B. Maigret, and H. A. Scheraga. MSEED: a program for the rapid analytical determination of accessible surface areas and their derivatives. *Journal of Computational Chemistry*, 13(1):1–11, 1992.

[187] M. Petitjean. On the analytical calculation of van der Waals surfaces and volumes: Some numerical aspects. *Journal of Computational Chemistry*, 15(5):507–523, 1994.

[188] M. Petřek, P. Kosinová, J. Koca, and M. Otyepka. MOLE: A Voronoi Diagram-Based Explorer of Molecular Channels, Pores, and Tunnels. *Structure*, 15(11):1357 – 1363, 2007.

[189] M. Petřek, M. Otyepka, P. Banas, P. Kosinová, J. Koca, and J. Damborsky. CAVER: a new tool to explore routes from protein clefts, pockets and cavities. *BMC Bioinformatics*, 7(1):316+, June 2006.

[190] M. Pharr and G. Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2010.

[191] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale. NAMD: Biomolecular Simulation on Thousands of Processors. In *Proceedings of SC 2002*, pages 1–18, 2002.

[192] M. Phillips, I. Georgiev, A. Dehof, S. Nickels, L. Marsalek, H.-P. Lenhof, A. Hildebrandt, and P. Slusallek. Measuring properties of molecular surfaces using ray casting. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–7, April 2010.

[193] T. D. Pollard and G. G. Borisy. Cellular motility driven by assembly and disassembly of actin filaments. *Cell*, 112(4):453–65, 2003.

[194] K. Pöthkow, B. Weber, and H.-C. Hege. Probabilistic Marching Cubes. *Computer Graphics Forum*, 30(3):931 – 940, 2011.

[195] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[196] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl. GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics*, 29(7):845–854, 2013.

[197] PyMOL. http://pymol.org.

[198] G. N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. Stereochemistry of polypeptide chain configurations. *Journal of molecular biology*, 7:95–99, July 1963.

[199] V. Ramakrishnan. Ribosome Structure and the Mechanism of Translation. *Cell*, 108:557–572, 2002.

[200] M. Raunest and C. Kandt. dxTuber: Detecting Protein Cavities, Tunnels and Clefts Based on Protein and Solvent Dynamics. *Journal of Molecular Graphics and Modelling*, 2010.

[201] F. M. Richards. Areas, Volumes, Packing, and Protein Structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977.

[202] J. S. Richardson. The anatomy and taxonomy of protein structure. *Advances in protein chemistry*, 34:167–339, 1981.

[203] A. Rigort, D. Günther, R. Hegerl, D. Baum, B. Weber, S. Prohaska, O. Medalia, W. Baumeister, and H.-C. Hege. Automated segmentation of electron tomograms for a quantitative description of actin filament networks. *Journal of Structural Biology*, 2011.

[204] R. S. Rowland and R. Taylor. Intermolecular Nonbonded Contact Distances in Organic Crystal Structures: Comparison with Distances Expected from van der Waals Radii. *J. Phys. Chem.*, 100(18):7384–7391, May 1996.

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

[206] J. Ryu, R. Park, J. Seo, C.-M. Kim, H.-C. Lee, and D.-S. Kim. Real-Time Triangulation of Molecular Surfaces. In *ICCSA (1)*, pages 55–67, 2007.

[207] T. Saito and T. Takahashi. Comprehensible rendering of 3-D shapes. *SIGGRAPH Comput. Graph.*, 24:197–206, September 1990.

[208] M. F. Sanner and A. J. Olson. Real time surface reconstruction for moving molecular fragments. In *Proceedings of the Pacific Symposium on Biocomputing, Maui*, pages 385–396, 1997.

[209] 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.

[210] T. Schlick. *Molecular modeling and simulation: an interdisciplinary guide: an interdisciplinary guide*, volume 21. Springer Science & Business Media, 2010.

[211] P. Schmidtke, A. Bidon-Chanal, F. J. Luque, and X. Barril. MDpocket: open-source cavity detection and characterization on molecular dynamics trajectories. *Bioinformatics*, 27(23):3276–3285, 2011.

[212] E. Schrödinger. An undulatory theory of the mechanics of atoms and molecules. *Phys. Rev.*, 28(6):1049, 1926.

[213] D. Sehnal, R. Svobodová Vareková, K. Berka, L. Pravda, V. Navrátilová, P. Banáš, C.-M. Ionescu, M. Otyepka, and J. Koca. MOLE 2.0: advanced approach for analysis of biomacromolecular channels. *Journal of Cheminformatics*, 5(1), 2013.

[214] A. Sharma, R. K. Kalia, A. Nakano, and P. Vashishta. Scalable and portable visualization of large atomistic datasets. *Computer Physics Communications*, 163:53–64, 2004.

[215] D. Siersma. Voronoi Diagrams and Morse Theory of the Distance Function. In *Geometry in Present Day Science, World Scientific*, pages 187–208. World Scientific, 1999.

[216] C. Sigg, T. Weyrich, M. Botsch, and M. Gross. GPU-based ray-casting of quadratic surfaces. In *Eurographics Symposium on Point-Based Graphics*, pages 59–65, 2006.

[217] J. M. Singh and P. J. Narayanan. Real-Time Ray Tracing of Implicit Surfaces on the GPU. *IEEE Transactions on Visualization and Computer Graphics*, 16:248–260, 2010.

[218] J. C. Slater. Atomic radii in crystals. *The Journal of Chemical Physics*, 41:3199, 1964.

[219] O. S. Smart, J. G. Neduvelil, X. Wang, B. A. Wallace, and M. S. Sansom. HOLE: a program for the analysis of the pore dimensions of ion channel structural models. *Journal of Molecular Graphics*, 14(6), Dec. 1996.

[220] D. K. Smith. Bibliography on molecular and crystal structure models. Technical report, US National Bureau of Standards, 1960.

[221] A. Sommerfeld. Zur Quantentheorie der Spektrallinien. *Ann. Phys.*, 356(17):1–94, 1916.

[222] R. Sridharamurthy, H. Doraiswamy, S. Patel, R. Varadarajan, and V. Natarajan. Extraction of robust voids and pockets in proteins. In *EuroVis-Short Papers*, pages 67–71, 2013.

[223] D. Stalling, M. Westerhoff, and H.-C. Hege. Amira: A highly interactive system for visual data analysis. In *The Visualization Handbook*, pages 749 – 767. Elsevier, 2005.

[224] M. Tarini, P. Cignoni, and C. Montani. Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12:1237–1244, September 2006.

[225] M. Till and G. Ullmann. McVol - A program for calculating protein volumes and identifying cavities by a Monte Carlo algorithm. *Journal of Molecular Modeling*, 16(3):419–429, 2010.

[226] R. Toledo and B. Lévy. Visualization of Industrial Structures with Implicit GPU Primitives. In *Proceedings of the International Symposium on Visual Computing (ISVC)) - Lecture Notes in Computer Science*, pages 139–150, Berlin, Heidelberg, 2008. Springer-Verlag.

[227] R. Toledo, B. Lévy, and J.-C. Paul. Iterative Methods for Visualization of Implicit Surfaces on GPU. In *Proceedings of the International Symposium on Visual Computing (ISVC) - Lecture Notes in Computer Science*, 2007.

[228] Y.-F. W. Tolga Can, Chao-I Chen. Efficient molecular surface generation using level-set methods. *Journal of Molecular Graphics and Modelling*, 31(4):442–454, 2006.

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

[230] L. Ulrik. *Proteins and Enzymes. (Lane Medical Lectures.).* Stanford University Publications. University Series. Medical Sciences. vol. 6. Stanford University Press, 1952.

[231] M. van Heel, B. Gowen, R. Matadeen, E. V. Orlova, R. Finn, T. Pape, D. Cohen, H. Stark, R. Schmidt, M. Schatz, et al. Single-particle electron cryo-microscopy: towards atomic resolution. *Q. Rev. Biophys.*, 33(04):307–369, 2000.

[232] A. Varshney, F. P. B. Jr., J. William, and W. V. Wright. Linearly Scalable Computation of Smooth Molecular Surfaces. In *IEEE Computer Graphics and Applications*, volume 14, pages 19–25, 1994.

[233] VIPERdb. http://viperdb.scripps.edu.

[234] VMD. http://www.ks.uiuc.edu/Research/vmd.

[235] R. Voorintholt, M. Kosters, G. Vegter, G. Vriend, and W. Hol. A very fast program for visualizing protein surfaces, channels and cavities. *Journal of Molecular Graphics*, 7(4):243 – 245, 1989.

[236] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *J. Reine Angew. Math.*, 134:198–287, 1908.

[237] N. R. Voss and M. Gerstein. 3V: cavity, channel and cleft volume calculator and extractor. *Nucleic Acids Research*, 38(suppl 2):W555–W562, 2010.

[238] B. Weber, G. Greenan, S. Prohaska, D. Baum, H.-C. Hege, T. Müller-Reichert, A. A. Hyman, and J.-M. Verbavatz. Automated tracing of microtubules in electron tomograms of plastic embedded samples of caenorhabditis elegans embryos. *Journal of Structural Biology*, 2011.

[239] M. Weisel, E. Proschak, and G. Schneider. PocketPicker: analysis of ligand binding-sites with shape descriptors. *Chemistry Central Journal*, 1(1):7+, 2007.

[240] A. F. Wells. *Structural inorganic chemistry*. Oxford University Press, 2012.

[241] H.-M. Will. Computation of Additively Weighted Voronoi Cells for Applications in Molecular Biology. *PhD thesis, Swiss Federal Institute of Technology, Zürich*, 1999.

[242] T. Wischgoll, J. Meyer, B. Kaimovitz, Y. Lanir, and G. S. Kassab. A Novel Method for Visualization of Entire Coronary Arterial Tree. *Annals of Biomedical Engineering*, 35(5):694–710, 2007.

[243] M. M. Woolfson. *An introduction to X-ray crystallography*. Cambridge University Press, 1997.

[244] K. Wüthrich. *NMR of proteins and nucleic acids*. John Wiley & Sons, New York, 1986.

[245] G. Wyvill, C. McPheeters, and B. Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.

[246] E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov. MolAxis: Efficient and accurate identification of channels in macromolecules. *Proteins: Structure, Function, and Bioinformatics*, 73(1):72–86, 2008.

[247] J. Yu, Y. Zhou, I. Tanaka, and M. Yao. Roll: a new algorithm for the detection of protein pockets and cavities with a rolling probe sphere. *Bioinformatics*, 26(1):46–52, Jan. 2010.

[248] Z. Yu. A List-Based Method for Fast Generation of Molecular Surfaces. *Annual International Conference of the IEEE EMBS*, 31, 2009.

[249] Z. Zhang, Y. Li, B. Lin, M. Schroeder, and B. Huang. Identification of cavities on protein surface using multiple computational approaches for drug binding site prediction. *Bioinformatics*, 2011.

[250] W. Zhao, G. Xu, and C. Bajaj. An algebraic spline model of molecular surfaces. In *Proceedings of the Symposium on Solid and Physical Modeling (SPM)*, pages 297–302, New York, NY, USA, 2007. ACM.

[251] S. Zhukov, A. Inoes, and G. Kronin. An Ambient Light Illumination Model. In *Rendering Techniques '98*, Eurographics, pages 45–56. Springer-Verlag Wien New York, 1998.

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Dritte haben von mir weder unmittelbar noch mittelbar geldwerte Leistungen für Arbeiten erhalten, die im Zusammenhang mit dem Inhalt der vorgelegten Dissertation stehen.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form als Dissertation eingereicht und ist als Ganzes auch noch nicht veröffentlicht.

Berlin, 2017

Norbert Lindow