Interactive Visualization of Molecular Dynamics Simulation Data

Naif Ghazi Alharbi

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Doctor of Philosophy

Swansea University
Prifysgol Abertawe

Department of Computer Science
Swansea University

April 26, 2020

Declaration

This work has not been previously accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed [redacted] (candidate)

Date 20/4/2021

Statement 1

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed [redacted] (candidate)

Date 20/4/2021

Statement 2

I hereby give my consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed [redacted] (candidate)

Date 20/4/2021

Abstract

Molecular Dynamics Simulations (MD) plays an essential role in the field of computational biology. The simulations produce extensive high-dimensional, spatio-temporal data describing the motion of atoms and molecules. A central challenge in the field is the extraction and visualization of useful behavioral patterns from these simulations. Throughout this thesis, I collaborated with a computational biologist who works on Molecular Dynamics (MD) Simulation data. For the sake of exploration, I was provided with a large and complex membrane simulation. I contributed solutions to his data challenges by developing a set of novel visualization tools to help him get a better understanding of his simulation data. I employed both scientific and information visualization, and applied concepts of abstraction and dimensions projection in the proposed solutions. The first solution enables the user to interactively filter and highlight dynamic and complex trajectory constituted by motions of molecules. The molecular dynamic trajectories are identified based on path length, edge length, curvature, and normalized curvature, and their combinations. The tool exploits new interactive visualization techniques and provides a combination of 2D-3D path rendering in a dual dimension representation to highlight differences arising from the 2D projection on a plane. The second solution introduces a novel abstract interaction space for Protein-Lipid interaction. The proposed solution addresses the challenge of visualizing complex, time-dependent interactions between protein and lipid molecules. It also proposes a fast GPU-based implementation that maps lipid-constituents involved in the interaction onto the abstract protein interaction space. I also introduced two abstract level-of-detail (LoD) representations with six levels of detail for lipid molecules and protein interaction. Finally, I proposed a novel framework consisting of four linked views: A time-dependent 3D view, a novel hybrid view, a clustering timeline, and a details-on-demand window. The framework exploits abstraction and projection to enable the user to study the molecular interaction and the behavior of the protein-protein interaction

and clusters. I introduced a selection of visual designs to convey the behavior of protein-lipid interaction and protein-protein interaction through a unified coordinate system. Abstraction is used to present proteins in hybrid 2D space, and a projected tiled space is used to present both Protein-Lipid Interaction (PLI) and Protein-Protein Interaction (PPI) at the particle level in a heat-map style visual design. Glyphs are used to represent PPI at the molecular level. I coupled visually separable visual designs in a unified coordinate space. The result lets the user study both PLI and PPI separately, or together in a unified visual analysis framework.

Acknowledgements

Undertaking this Ph.D. has been a truly life-changing experience for me, and it would not have been possible without the support and guidance that I received from many people. Firstly, I would like to thank The Almighty God for His help and guidance, and for choosing me to be supervised by Dr. Robert S. Laramee. I believe that being supervised by Dr. Laramee is one of the bounties of Almighty God. I would like to give a very big thank-you to my supervisor Dr. Laramee for all the support and encouragement he gave me during my Ph.D. He has been so generous with his time and efforts—supporting above and beyond what is typical of a Ph.D. supervisor. Without his guidance and constant feedback, this Ph.D. would not have been achievable. I would also like to thank my second supervisor Dr. Adeline Paiement, who was always ready to answer my questions and provide me with advice regarding my research topic. Many thanks also to Dr. Matthieu Chavent for his support and guidance throughout our projects. I would also like to thank Mohammed Alharbi, Xavier Martinez, Alexander Rose, and Marc Baaden for their help and valuable feedback. A special thanks to Michael Krone for his continuous feedback. I would like to thank the Ministry of Education of Saudi Arabia and the Saudi Cultural Bureau in London for their financial support. The community of Ph.D. students in our research group has been such a valuable support network. In particular, I would also like to thank Dr. Sean Walton, Dr. Richard Roberts, Liam McNabb, Dylan Rees, Dave Greten, and Rhodri Fabbro who have proofread my work throughout the project. An additional thanks to Dr. Joss Whittle for his advice and guidance on GPU techniques. Lastly, I would like to express my deepest gratitude to my family for their endless support throughout my life. I am incredibly thankful for my mother, wife, and children for supporting me spiritually throughout writing this thesis and being supportive in all that I do.

List of Publications

This thesis is based on the following publications:

1. Naif Alharbi, Robert S Laramee, and Matthieu Chavant, MolePathFinder: Interactive Multi-Dimensional Path Filtering of Molecular Dynamics Simulation Data, The Computer Graphics and Visual Computing (CGVC) Conference 2016, pages 9-16, 15-16 September 2016, Bournemouth University, UK

2. Naif Alharbi, Mohammed Alharbi, Xavier Martinez, Michael Krone, Alexander Rose, Marc Baaden, Robert S. Laramee, Matthieu Chavent, Molecular Visualization of Computational Biology Data: A Survey of Surveys in EuroVis Short Papers 2017 , pages 133-137

3. Naif Alharbi, Micheal Krone, Matthieu Chavant, Robert S Laramee, VAPLI: Novel Visual Abstraction for Protein-Lipid Interactions, IEEE SciVis Short Papers, IEEE VIS 2018, 21-26 October 2018, Berlin, Germany

4. Naif Alharbi, Micheal Krone, Matthieu Chavant, Robert S Laramee, LoD LPI: Level of Detail for Visualizing Time-Dependent, Protein-Lipid Interaction, International Conference on Information Visualization and Applications (IVAPP) 2019, 25-27 February 2019, Prague, Czech Republic

5. Naif Alharbi, Micheal Krone, Matthieu Chavant, Robert S Laramee, Hybrid Visualization of Protein-Lipid and Protein-Protein Interaction, the Eurographics Workshop on Visual Computing in Biology and Medicine (VCBM) 2019, 4-6 September 2019, Brno, Czech Republic

Supplementary Material

Each chapter includes a supplementary material. I highly recommend that the reader views these videos before reading their corresponding chapter—they are designed to provide an overview of the software features and demonstrate their utility. These videos and the source code are available online here:

Chapter	Material type	URL
Chapter 2	Video	https://github.com/NaifAlhar6i/Chapter2
Chapter 3	Video	https://github.com/NaifAlhar6i/Chapter3
Chapter 4	Video	https://github.com/NaifAlhar6i/Chapter4
Chapter 5	Video	https://github.com/NaifAlhar6i/Chapter5
Chapter 6	Video	https://github.com/NaifAlhar6i/Chapter6
Chapter 7	Source Code	https://github.com/NaifAlhar6i/Chapter7

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Motivation

Contents

> *"If a man will begin with certainties, he shall end in doubts; but if he will be content to begin with doubts he shall end in certainties." -Sir Francis Bacon*[1]

Proteins are extremely complex macro-molecules that are essential to biochemical pathways in every living organism. Standalone or as a component of multi-unit combinations, it facilitates a wide variety of operations, like the replication of DNA, catalysis of chemical processes, or the movement of molecules through cells. The capacity of a protein to associate with many other molecules plays a critical role in such reactions.

Since more in-depth comprehension of protein reactivity is correlated with advances in medicine, agriculture, or pharmaceuticals, the research of protein association trends has been at the forefront of biochemical research for a long time. Sadly, the nature of molecular structure and the

[1] Sir Francis Bacon (1561-1626) was an English philosopher and statesman who served as Attorney General and as Lord Chancellor of England

need for costly and time-consuming in vitro studies are progressing slowly in this field. Many computational techniques tend to support this work by modeling in-silicon trials and thereby lowering the price of actual researches. However, these techniques can produce a vast number of data that must then be checked by domain experts. For instance, the movements of millions of atoms over a given period can be simulated using molecular dynamic simulations. Most of the occasions, it is challenging to evaluate these data and identify key patterns in it by merely watching a long simulation. Another example is the protein-protein docking simulation, which predicts possible associations with two or more proteins. Here, the output of computational modeling frequently comprises of hundreds to thousands of possible combinations that should be evaluated separately by domain professionals to assess the most biochemically important. A proper visual depiction is an extremely valuable part of the process of interpreting and discovering the effects of the statistical methods. This allows domain experts to consider the spatial features and interactions as well as the relative differences of the molecules reacting with each other.

Visual analysis and visualization methods have, therefore, become an essential part of proteomic science, both as support during studies as well as abandonment and interpretation of tests. The primary purpose of these techniques is to accelerate up the research procedure by often dynamically capturing and communicating the key features of the data in this kind of way that the formerly unobservable trends and interactions become more visible. While several approaches have been identified in molecular visualization studies in recent years, there are still aspects and issues that have not yet been tackled. The purpose of this study is to bring another piece to the puzzle by proposing new approaches that fill the gaps in the process of visual discovery of protein and protein-ligand interactions.

The field of visualization focuses on the creation of images that communicate the necessary information about the underlying data. It plays a crucial role in enhancing our ability to understand broad and complicated data. Computational biology, to study living cells and the activity of molecules, researchers depend on simulation, such as the simulation of Molecular Dynamics (MD) and the sampling of Monte Carlo (MC). The simulation includes the development of molecular structures in the forms of sequential position-snapshots that reflect a particle trajectory.

Molecular dynamics visualization is used to get insight into the simulation data, thus the molecular dynamics visualization has received a great deal of attention over the past decades. Numerous representations were presented for the depiction of MD data from simple molecules to complex macro-molecular systems [53, 45]. Protein molecules, for example, are often represented by surfaces [170, 3, 93] while small molecules can be depicted as licorice, ball-and-stick or Van der Waals spheres [28]. See Figure 1.1.

In-silico simulations of chemical reactions, like molecular docking, minimize the costs and time required for in-vitro research. However, the quality of the data created always demands more study. It is probably up to the domain specialist to determine the adequacy of the data and to make conclusions from all of this. Without right equipment, it can also be a challenging and time-consuming task. Nevertheless, visualization metaphors that endorse specific research activities and their configurations in an integrated visual analytics framework can dramatically speed up the analysis process and help uncover fascinating trends and relationships in the data.

While I was planning to write this introduction, I was very interested in employing the title of the thesis during the process. The title involves essential terminology that expresses the theme of the thesis and denotes the identity of the final product. I decided to start by analyzing the title of the thesis with a simple description of its terminology in such a way that the introduction provides the reader with the fundamental and essential background of the topic.

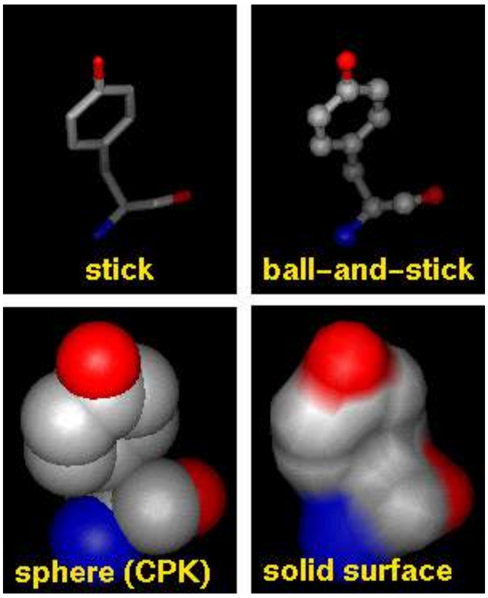


Figure 1.1: Different representations for a molecule. Image is generated via VMD [25]

The title consists of two combined terminologies: ”interactive visualization” and ”molecular dynamics simulation”. The term ”interactive visualization” links with Human-computer interaction and computer visualization while ”molecular dynamics simulation” comes from scientific computing. The three disciplines are classified under computer applications. This thesis applied concepts and methods from the discipline of Human-computer interaction and computer graphics to solve problems posed by the scientific computing discipline. The following sections address some questions concerning these terminologies and their relationship to the thesis.

1.1 Molecular Dynamics Simulation

Here I address three questions; **what is Molecular Dynamics (MD) simulation, why it is used, and how it poses challenges.** Molecular dynamics simulation is a computational method used to simulate the physical movements of atoms and molecules depending on time and force field chosen under different conditions, such as temperature, pressure, and force (Samui [29]). In computational biology, from which I obtain my data, scientists rely on simulation to study living cells and the behavior of molecules. AMBER [30], CHARMM [31], and GROMACS [32] are examples of MD software, which are widely used in the field to produce MD simulation data. They produce raw data that describe the dynamics and properties of every individual entity in the molecular system. The MD software can be used for modeling dynamic protein motion to study the workings of macromolecular systems at both temporal and spatial scales that are otherwise hard to access empirically (Karplus *et al.* [33]).

Nowadays, advances in simulation enable the study of large models constituted by millions or billions of atoms (Perilla *et al* [34]). With current simulation methods, one can analyze membrane organelles, virus envelopes, and large patches of bacterial membranes (Chavent *et al.* [35]). These new models are, indeed, especially difficult to analyze due to the diversity of molecules (particularly the lipids) and the huge quantity of data. Furthermore, molecular dynamics simulations represent time-dependent data, which increases the complexity and poses space-time challenges. Thus, computational biologists are continually facing challenges to finding programs that can help them to understand their MD data. This is the primary motivation behind this thesis, and visualization is a primary candidate for helping computational biologists address their challenges and obtaining insight into their simulations.

1.2 Interactive Visualization

This section provides a brief overview of visualization and illustrates how interactivity can augment visualization.

The field of visualization is focused on creating images that convey salient information about underlying data and processes (Hansen and Johnson [36]). Visualization can be broadly classified into *Scientific* and *Information* Visualization (Keim *et al.* [37]). *Scientific* Visualization is defined as: "Data that describes a physical phenomenon is defined as scientific data. Examples of this are fluid flow, living organisms, and data from the natural world" (McNabb and Laramee [38]). It is used to depict data that represent 3D geometries, data that can be understood as scalar, time-dependent vector or tensor fields [37]. *Information* Visualization is defined as "the communication of abstract data through the use of interactive visual interfaces" (Keim *et al.* [39]). In information visualization, no explicit spatial references are necessarily given, and data may comprise hundreds of dimensions. Some visualization designs may need to straddle both the scientific visualization and information visualization arenas (Rhyne *et al.* [40]).

In the context of molecular dynamics, scientific and information visualization have widely been used to convey underlying information about the simulation data. VMD [25], UCSF Chimera [41], and PyMol [42] are examples of software that are used in visualizing molecular simulation data. Scientific visualization, often, is used to provide 3D imagery of atoms, molecules, and complex systems. Molecules at the atomic level usually are visualized by simple shapes. Spheres are used to depict atoms, and sticks depict bonds between them. If atomic details are not desired, molecules can be represented by an approximated surface. In order to represent a complex system, such as the membrane, advanced visualization techniques are used to enhance the visualization or boost performance. While scientific visualization mainly focuses on depicting simulation with limited dimensions (i.e. 3D or 4D space-time + limited properties), information visualization utilizes abstraction and employs novel visualization techniques such as parallel coordinates, treemaps, glyphs, and heat-map visual design representations. The information visualization techniques convey abstract information and provide the user with a special understanding of the system's behavior.

Interactive visualization systems combine visualization with interactivity. The interactivity of a system, in general, means that the system works incrementally and reversible in such a way every change is immediately sent to the system and dynamic feedback is generated (Brodbeck *et al.* [43]). A good visualization design can be guided by the Visual Information Seeking Mantra “overview first, zoom and filter, then details-on-demand” (Shneiderman [44]). This kind of visualization encourages exploration and allows users to experience the data spaces.

1.3 Thesis Pipeline

The thesis consists of eight, whilst individual and significant in their own rights, complementary chapters. Four of these chapters, besides the literature review, represent the core of the thesis and its intent. Each chapter works in effort, whilst complementing the others respectively, to illustrate and support a coherent plan for new visualization methods. See Figure 1.2. Chapter 3 begins the main body of this thesis. As a starting point, it plays a key role in exploring and understanding the underlying data. This data is fundamental in setting a sound foundation upon which subsequent analysis can be established. The chapter seeks to understand said data, then, by utilizing different visualization approaches in visualizing molecules trajectories. It focuses on visualizing time-dependent trajectory MD data at the atomic level from which the user is able to render the entire set of trajectories or a sub-set of the data. As Chapter 3 focuses on exploring the trajectories of molecules and their properties, Chapter 4 digs deeper and more exclusively into studying the molecules trajectories by focusing on the interaction

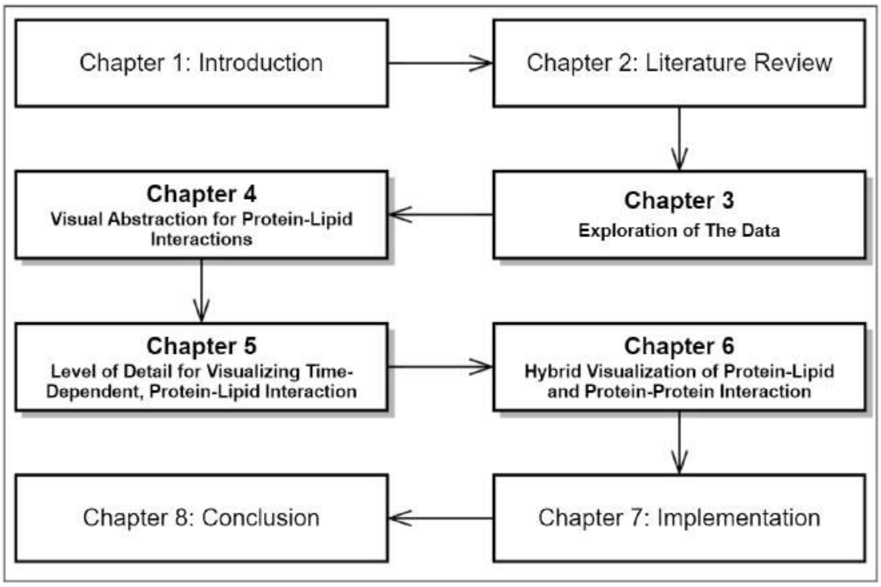

Chapter 1: Introduction
Chapter 2: Literature Review
Chapter 3
Exploration of The Data
Chapter 4
Visual Abstraction for Protein-Lipid Interactions
Chapter 5
Level of Detail for Visualizing Time-Dependent, Protein-Lipid Interaction
Chapter 6
Hybrid Visualization of Protein-Lipid and Protein-Protein Interaction
Chapter 7: Implementation
Chapter 8: Conclusion

Figure 1.2: Thesis pipeline.

around proteins molecules. Namely, the area surrounding protein molecules at which the most molecule interaction take place. The chapter offers a more comprehensive study into the trajectories and provides a visual abstraction for protein-lipid interaction. Chapter 4 presents a novel, abstract space that simplifies proteins molecules for the sake of studying the interaction between proteins and the surrounded particles. Moving on in a complementary manner, Chapter 5 addresses some limitations in Chapter 4. It does so by utilizing a dynamic Level-Of-Detail (LoD) technique. The level of detail for visualising time-dependent, protein-lipid interaction is also regarded as a further expansion from Chapter 4. Finally, Chapter 6 introduces a hybrid visualization tool for molecules interactions by implying the solutions proposed by previous Chapters.

1.4 Thesis Overview and Contributions

This thesis addresses a number of challenges that are posed by molecular dynamics simulation. It applies the concept of "interactive visualization" to produce novel solutions that help computational biologists to obtain a better understanding of their molecular dynamics simulation. The solutions produced were developed under the supervision of an experienced visualization scientist (Dr. Robert S. Laramee) and were guided by an experienced scientist in computational biology (Dr. Matthieu Chavent).

Each chapter in the thesis is dedicated either to addressing a specific problem that unsolved or to improve a previous solution. Under each heading below, the summary of a chapter including challenges, objectives and contributions. For continuity, these summaries are arranged to start with corresponding chapter headings.

Chapter 2: Literature Review This chapter consists of two types of review: a broad survey of surveys (SoS) and an in-depth survey. The SoS summarizes 11 molecular visualization surveys while the traditional survey summarizes a number of molecular visualization research papers concerning molecular dynamics visualization.

Challenges and Objectives: The literature review chapter is an essential phase in the Ph.D. process. It helps researchers to build up their scientific background and to define the direction of their research. The main challenge is that the molecular dynamics visualization articles are published in two separated fields: computational biology, and the computer graphics and visualization fields. These fields have witnessed rapid technological advances in the last decade.

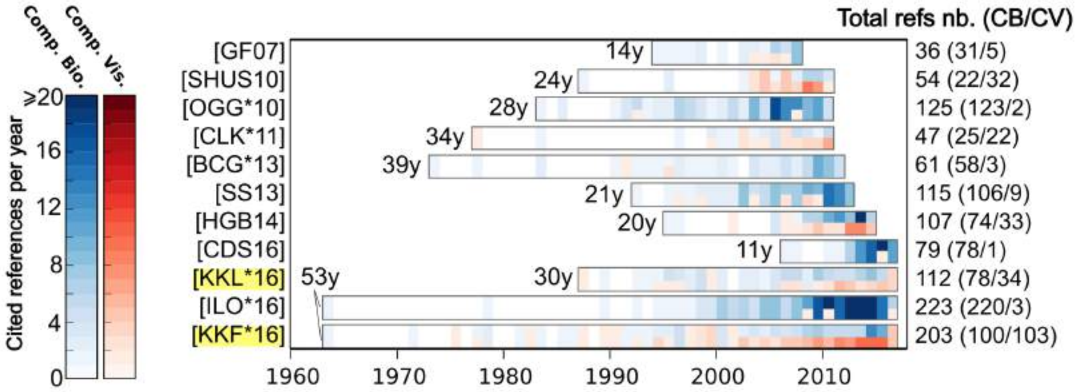


Figure 1.3: List of surveys presented in this article indicating their time span, number of cited references per year, total number of references, and the ratio of papers coming from the Computational Biology (CB) and Computer Visualization (CV) fields respectively. If a paper refers to both types (CB and CV) of references for a same year, the cell is divided in two rows of different color.

Thus, coping with the large number of scientific articles from both fields is a challenging task. Furthermore, there remains a gap between the two communities of visualization and computational biology, resulting in additional challenges to bridge the divide. The SoS attempts to address these challenges by presenting unified state-of-the-art reviews from both communities. While the SoS provides a broad overview of the topic, the traditional survey provides a deeper understanding of the summarized molecular dynamic visualization articles.

Contributions:

- The first survey of surveys in molecular dynamics visualization.
- Identifying solved and unsolved problems in the molecular dynamic visualization research literature.

Chapter 3: Interactive Multi-Dimensional Path Filtering of Molecular Dynamics Simulation Data This chapter describes the first project in the thesis (MolPathFinder). MolPathFinder focuses on visualizing time-dependent trajectory MD data. It provides the user with 2D and 3D representations by utilizing a number of different visualization techniques.

Challenges and Objectives: The analysis of lipids dynamical behavior in membrane models plays a significant role in understanding how lipids and proteins can interact with each other,

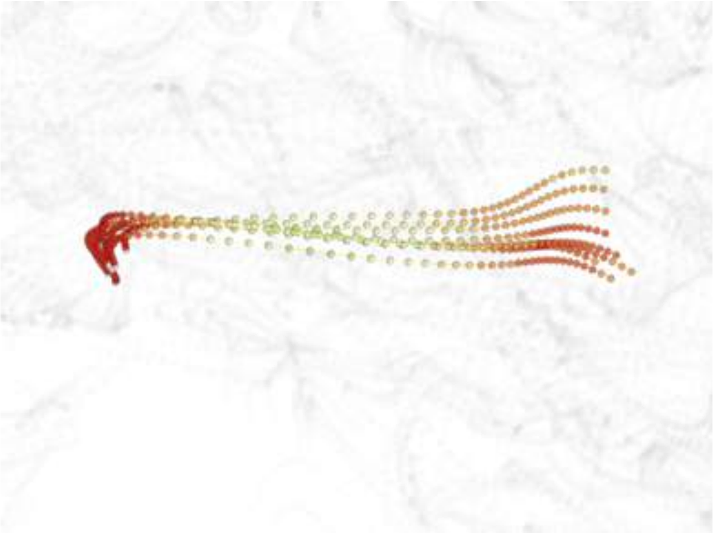


Figure 1.4: The color of atom reflects local edge length in 3D.

and how it can influence their respective dynamic properties. Visualization can convey clear information about MD trajectories from atomic details to a more global scale. However, common molecular visualization tools do not yet have dedicated rendering functions to analyze such lipid paths in a user-friendly way. This chapter aims to fill the gap by providing a tool that enables computational biology scientists to obtain insight into the complex motions of lipid molecules at both scales.

Contributions: I have developed MolPathFinder as a set of tools to help the user interactively extract path features during the course of the Molecular Dynamics simulation. This tool exploits new interactive visualization techniques:

- Novel interactive filtering techniques to help the user to identify trajectories based on path length, edge length, curvature and normalized curvature, and their combinations.
- Combination of 2D-3D path rendering in a dual dimension representation in order to highlight differences arising from the 2D projection on a plane.

Chapter 4: Novel Visual Abstraction for Protein-Lipid Interactions Chapter 4 presents a novel solution for visualizing protein-lipid interaction in a membrane MD simulation. The proposed solution employs abstraction and simplifies the computation and visualization of protein-lipid interaction.

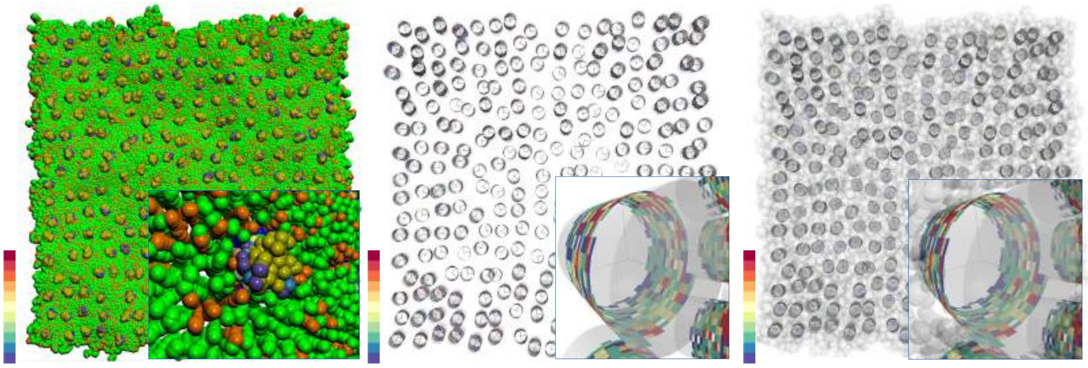


Figure 1.5: Visual Abstraction for Protein-Lipid Interaction (VAPLI). (Left) A naive visualization of protein-lipid interaction (PLI). (right) Focus-and-Context applied to the PLI.

Challenges and Objectives: Proteins are often dynamically modulated by their environment through interactions, i.e. the surrounding membrane. It can be difficult to fully understand the molecular mechanisms of such interactions especially for large systems. Furthermore, these interactions are often transitory and quickly change in a time-dependent manner. These challenges may not be overcome by classical molecular depictions. This chapter aims to design a new type of molecular abstraction focusing on the Protein-Lipid Interactions (PLI).
Contributions: The design and implementation of a Visual Abstraction for PLI (VAPLI) including:

- A novel cylindrical definition and abstraction of the PLI for large MD simulations.
- A fast GPU-based implementation to calculate PLI on the fly.

Chapter 5: Level of Detail for Visualizing Time-Dependent Protein-Lipid Interaction
This chapter is built based on the future work described in Chapter 4. Chapter 5 describes the proposed abstract protein space in the context of membrane protein-lipid interaction. It also describes six levels of detail for both lipid types and the PLI abstract space.
Challenges and Objectives: Protein molecules, for example, are often represented by surfaces. This representation provides an overview of the protein shape on a molecular scale. At the atomistic scale, small molecules including the atoms and bonds are often represented by models such as ball-and-stick. These representations are commonly used to visualize the MD data at the desired scale. However, depending on the intended analysis task, these representations

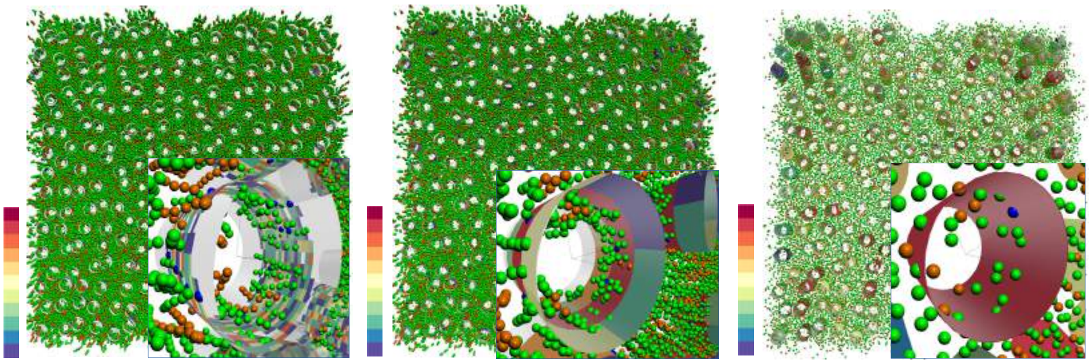


Figure 1.6: Three LoD PLI representations. PLI frequency is mapped to color. From left to right, PLI at high resolution, medium resolution, and low resolution.

might also become part of the challenge due to their complexity. The aim here is to provide an LoD visualization of the PLI over the entire simulation time-span.

Contributions: The design and implementation of an LoD PLI including:

- A definition and abstraction of PLI with six levels of detail for the protein-lipid interaction.
- A smooth transition between the six LoDs utilizing lipid-scale and particle-scale effects.
- A fast GPU-based implementation to represent PLI LoD in the abstract interaction space.

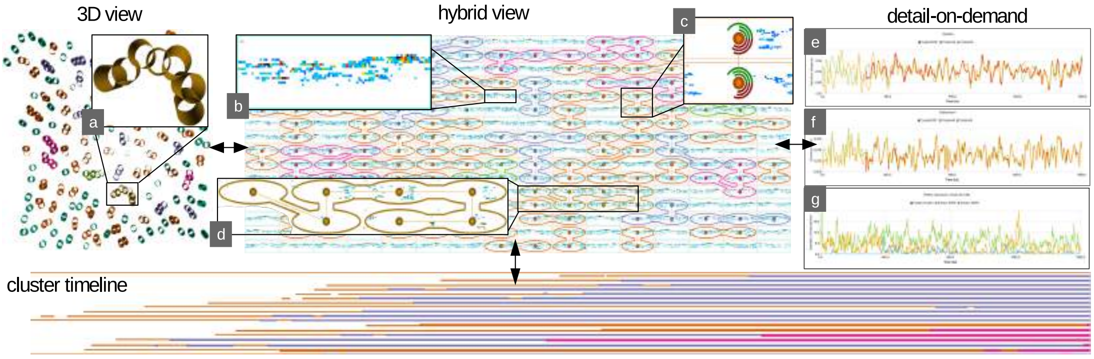


Figure 1.7: The interface of the MD hybrid visualization framework.

Chapter 6: Hybrid Visualization of Protein-Lipid Interaction (PLI) and Protein-Protein Interaction (PPI) This Chapter is built on Chapter 4 and 5. It describes a novel PLI and PPI visualization framework. The framework utilizes four visual designs that enable the user to study a time-dependent membrane simulation. It employs abstraction and space projection to address a number of visualization challenges. It also utilizes a novel hybrid view to enable the user to study PLI and PPI, and the behavior of the PPI and clusters.

Challenges and Objectives: Protein-lipid interactions (PLI) and protein-protein interaction (PPI) occur in the same space-time domain. The simulation's size, length, and complexity increase the challenge of understanding their dynamic behavior. In addition to the view-dependency problem in the 3D visualizations, most of the interaction details are obscured due to particle overlap and occlusion. This chapter aims to address these challenges by utilizing abstraction and projection techniques.

Contributions: A novel framework consisting of four linked views: A time-dependent 3D view, a novel hybrid view, a clustering timeline, and a details-on-demand window including:

- The first tool to both visually separate and combine time-dependent PLI and PPI imagery in a unified visual design.
- A novel, glyph-based representation of PPI that also encodes rotational properties.

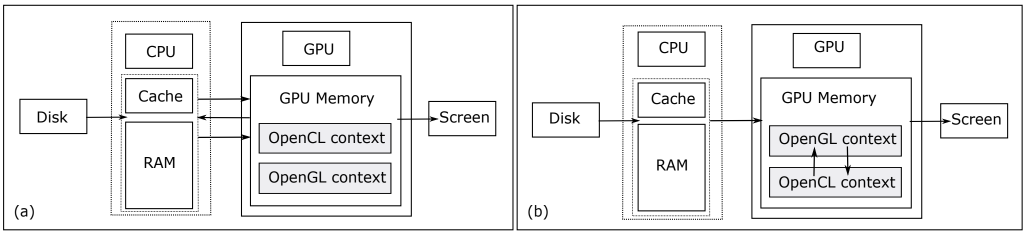


Figure 1.8: A traditional approach vs. my approach that I described to accelerate the visualization.

Chapter 7: Real-Time Rendering of Molecular Dynamics Simulation Data This chapter is inspired by the performance challenges that I encountered in Chapter 3. Chapter 7 illustrates and demonstrates how real-time visualization of molecular dynamics simulation can be achieved. It focuses on the implementation from the performance perspective by exploiting *OpenGL 4.3* and *OpenCL 1.1*. It also presents the performance strategies that I applied in Chapter 4 & 5.

Challenges and Objectives: The size and nature of the MD simulation data, the required processing per frame, and the rendering approaches affect the final rendering frame rate and pose three main challenges: the I/O, the computation, and the rendering challenges. This chapter is proposed to address these challenges by introducing a memory mapping technique and an advanced GPU approach utilizing the OpenCL interoperability feature.
Contribution: A number of strategies to achieve a real-time rendering of molecular dynamics simulation data and the main contribution:

- A memory mapping technique to enhance the I/O performance.

Chapter 8: Conclusion and Future Work This Chapter concludes the thesis and provides potential future research directions.

Chapter 2

Literature Review

Contents

> *"Life can only be understood backwards; but it must be lived forwards." -Soren Kierkegaard* [1]

This chapter provides two types of review: a broad overview of the molecular dynamics visualization literature and an in-depth review of a set of selected articles from the molecular dynamics literature.

[1] Soren Kierkegaard (1813-1855) was a Danish philosopher, theologian, poet, social critic and religious author who is widely considered to be the first existentialist philosopher.

2.1 Introduction and Motivation

In computational biology, visualization is an important means to gain insight into molecular structures and their dynamics. Due to its demanding nature, visualizing molecular data has always been tightly linked to computer hardware development (Levinthal [49]). Originally, papers describing advances in molecular visualization were welcomed by the whole scientific community and published in journals with a broad audience such as *Science* (Langridge *et al* [50]). More recently, scientific fields have become more specialized, resulting in focused scientific communities publishing in dedicated journals. This fragmentation can lead to undesired situations where visualization challenges may be published in one type of journal while the solutions may appear in another. This section aims to reunite the communities by describing both questions posed by the computational biology community and answers provided by (or new challenges for) the visualization community. It provides, for newcomers and experienced researchers, a unique and concise perspective presenting state-of-the-art literature in molecular visualization.

Survey Scope: I have selected 11 survey papers spanning both fields. I focused on literature reviews addressing the rapidly expanding fields of structural biology and molecular modelling with a focus on spatio-temporal simulation data. For readers interested in a broader view of visualizing biological data, I refer to O'Donoghue *et al* [51]. The literature reviews cover selected, related topics: visualization of molecular structures (Goddard and Ferrin [52], Kozlíková *et al.* [53]) and software dedicated to this task (O'Donoghue *et al.* [54]), advances based on Graphics Processing Units (GPUs) (Chavent *et al.* [28], Stone *et al.* [55]), detection and analysis of cavities in proteins (Brezovsky *et al.* [56], Krone *et al.* [57]), time-dependent biological data (Secrier and Schneider [58]), and new challenges in molecular modelling leading to new visualization questions (Chavent *et al.* [35], Im *et al* [59]). As a useful introduction to the links between molecular simulation and visualization, I discussed the review by Hirst *et al.* [60]. As the literature selection covers a large time span, I focused on the last fifty years from the mid-sixties to 2016 (see Figure 2.1).

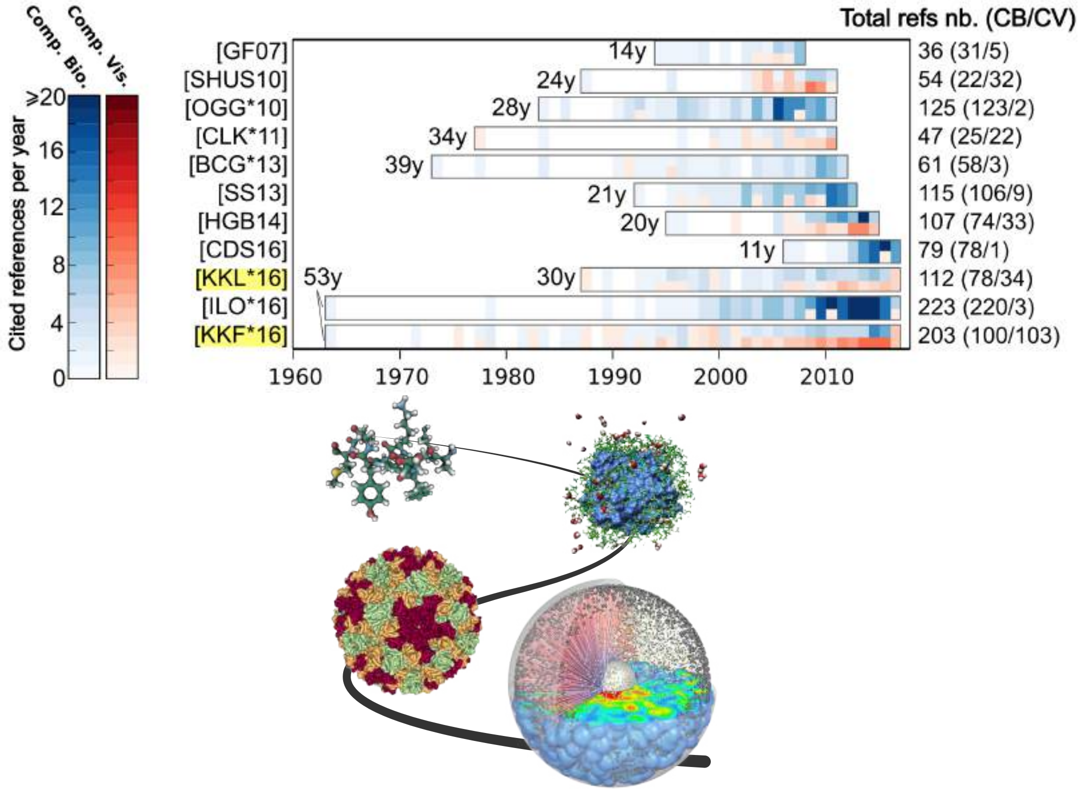


Figure 2.1: List of surveys presented in this section indicating their time span, number of cited references per year, total number of references, and the ratio of papers coming from the Computational Biology (CB) and Computer Visualization (CV) fields respectively. If a paper refers to both types (CB and CV) of references for a same year, the cell is divided in two rows of different color. The collage to the bottom illustrates the scales covered by the visualizations in these surveys, ranging from small molecules over protein complexes to whole cells (screenshots made with UnityMol [1], MegaMol [2, 3], NGL Viewer [4, 5] (norovirus example), and CellVis [6]).

2.2 Survey of Molecular Dynamics Visualization Surveys

In this section I describe each review and group them by main common themes such that closely related surveys are together. Details about references cited and literature time span are depicted in Figure 2.1.

Introduction to Molecular Visualization and Simulation

Hirst *et al.* [60] propose an overview of the recent literature on molecular simulation and visualization. They highlight the increasing importance of Human-Computer Interaction (HCI) and virtual reality in the molecular visualization context. This survey introduces a series of articles dedicated to molecular visualization. This review contains 107 citations covering 20 years of research with about two thirds of the citations referring to computational biology work and one third to computer science papers.

Visualization of Molecular Structures

O'Donoghue *et al.* [54] review visualization methods and tools that enable the community of structural biologists to gain insight into macromolecular structures. This report covers an extensive list of web-based and stand-alone tools and discusses the advantages and disadvantages of the most common molecular structure acquisition techniques. The review covers 28 years of scientific literature containing 125 references, almost exclusively related to works published in the biological and experimental communities.

Goddard *et al.* [52] discuss developments and challenges in visualization of molecular structure to better understand molecular systems such as Depth Perception, Level of Detail (LoD), 2D and abstract representations. This review focuses on 14 years, citing 36 papers, of which 5 are from computer science.

The recent state-of-the-art report by Kozlíková *et al.* [53] proposes an extensive review of visualizing biological data covering a wide range of spatial scale from atoms to cells. The authors pay particular attention to molecular surface rendering with an interesting chronological perspective on visualization of the *solvent excluded surface*. Numerous challenges evoked by Goddard *et al.* [52] are addressed in this review such as LoD or the effective representation of dynamical data. This review covers more than fifty years of scientific research referring to 203 articles. These references are well balanced between computational biology and computer science literature.

Detection and Visualization of Cavities

While the previous selection of surveys discusses how it is possible to render a structure, here I present two reviews highlighting detection, visualization, and analysis of molecular cavities. These cavities are often important for the proper function of a molecule. This task is especially difficult as it needs to visualize voids which have to be well defined and detected.

Brezovsky *et al.* [56] review programs that identify, visualize, and analyse protein voids. As the shape of the void may have an impact on the technique used to detect it, the authors compare different tools to assess which one is the best for a dedicated type of space. The review spans 39 years of literature, presenting a majority of articles published in computational biology journals.

Complementary to Brezovsky *et al.*, Krone *et al.* [57] detail the technical background of the algorithms. Their report also covers visualization methods for cavities. The authors present the definition and the classification of cavities. They classify the methods according to the underlying algorithms or the type of cavity definition. This study constitutes a very comprehensive review, spanning 30 years and citing 112 papers. The ratio of computer science to computational biology related papers is about one third.

GPU Computing

With the developments of programmable graphics cards in the early 2000's, development of new algorithms that harness this relatively new computing power are evolving rapidly.

Chavent *et al.* [28] focus on studies that redesign traditional algorithms to exploit GPUs. This survey covers techniques that display small molecules up to macromolecular assemblies, and discusses visual effects to enhance molecular structure perception. It covers 34 years of research and cites 47 papers almost equally balanced between computer science and computational biology.

Even though it is not completely focused on visualization, I mention a closely related review from Stone *et al.* [55] discussing the development of GPU-computing to accelerate molecular simulations. This work covers 24 years of research and refers to 54 papers predominantly from computer science. Note that some of the previously cited reviews also discuss GPU computing (e.g. Goddard and Ferrin [52], Hirst *et al.* [60], Krone *et al.* [57] and Kozlíková *et al* [53]).

Visualizing Time-dependent Biological Data

Improved rendering efficiency now enables visualization of dynamical systems. Several reviews discuss this topic. O'Donoghue *et al.* [54] present different tools to render molecular motions. Kozlíková *et al.* [53] dedicate a full section to the visualization of molecular dynamics data.

The review by Secrier *et al.* [58] discusses the visualization of biological processes at different time scales. This survey reviews time-dependent biology visualization tools by categorizing them into seven groups based on their time scale: molecular level (nano- to micro-seconds), gene level (micro-seconds/hours), network level (micro-seconds/days), cellular level (hours/days), level of an organism (days/weeks), population level (billions of years) and evolutionary scales (multiple levels). This review covers 21 years and cites 115 references with 9 computer science papers.

Challenges in Computational Biology

Computational biology is evolving very quickly, thus, new challenges appear regularly. Here, I highlight two recent reviews that outline challenges in computational biology. For computer scientists, these reports can inspire future research directions. For computational biologists, these reports cover the latest state-of-the-art.

Chavent *et al.* [35] discuss the advances in molecular simulations of membrane proteins with a focus on protein-lipid interactions and modelling complex membranes at different scales. At the nanoscale resolution, simulations are used to predict and investigate fine lipid-protein interactions. Beyond the nanoscale, it is necessary to model very large and crowded systems requiring significant computing power. Reaching time-scales probed in experiments will require the development of new types of models. This review covers very recent work (the last 11 years), almost exclusively from the computational biology field.

Im *et al.* [59] explore the modelling of biological systems at different scales. They discuss how to move from one scale to another while simultaneously maintaining a high resolution to develop meaningful models. The next big challenge is to reach the cell scale and combine models with experimental data. This survey covers a long time span (up to 53 years) and is constituted by 223 references, mostly from the computational biology field.

2.2.1 From Text to Information: Meta-analysis of the Reviews

I perform a meta-analysis of all eleven surveys based on reference origins (CB or CV), shared references, and extracted keywords. These analyses yield new comparisons and insights not available from simply reading each paper separately.

Methods: To construct Figures 2.1 and 2.2, I extract the references from the Scopus database [61] and analyze them using in-house Python scripts. For Figure 2.1, the references are curated by us to define which category a reference belongs to. Briefly, if the reference was published in an ACM, IEEE or related conference and journal it is categorized as a computer visualization paper, otherwise it was tagged as a *'computational biology'* paper. This category is kept very simple due to the paper format. I also investigated the concordance of important words across the surveys using the Natural Language Toolkit [62] and Python scripts. Figure 2.3 shows a parallel coordinates plot that highlights the most represented words for each survey and a word cloud generated using the script by Müller [63].

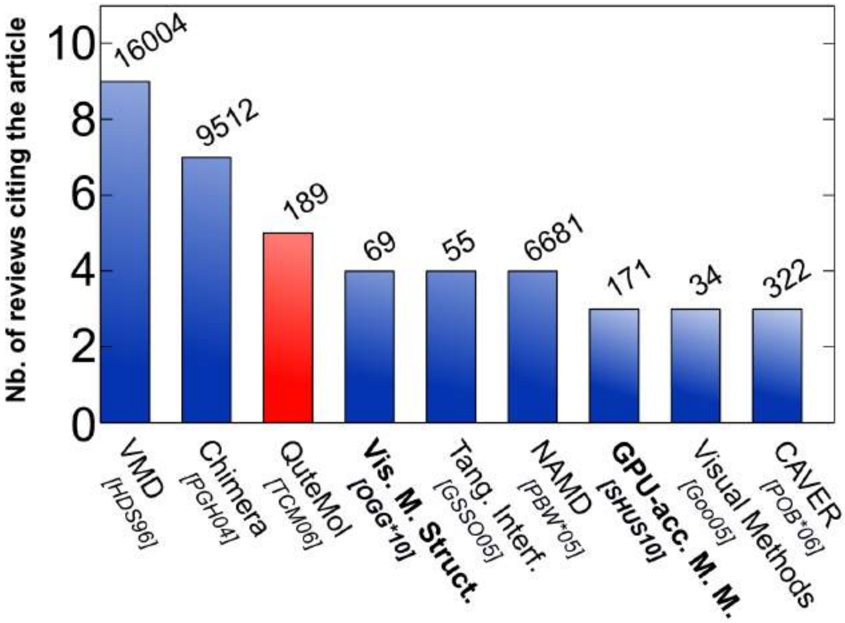


Figure 2.2: The most common references shared by the 11 selected surveys. The two papers cited in bold print are included in my selection. I only displayed papers shared by at least 3 surveys. On top of each bar is the number of citations for each paper. Blue: computational biology papers; Red: scientific visualization paper.

References as a Function of Time

Figure 2.1 shows that the selected reviews focus mainly on the last 25 years, even though some highlighted works were published before 1980. There is an imbalance between references from the CV and CB fields. The latter is clearly more represented. There is of course an intrinsic bias, as selected reviews are more from computational biology (9: [52, 54, 55, 28, 56, 58, 60, 35, 59]) than pure data visualization (2: [53, 57]). Nevertheless, at least three of them ([55, 28, 60]) are focusing on molecular graphics or algorithms development, which counter-balances the ratio to 6:5. Furthermore, even the reviews published in the scientific visualization field cite numerous computational biology papers. To explain this imbalance, I hypothesize that the technical orientation of CV papers and the dissemination through very dedicated conferences may prevent some researchers of being aware of these studies. Recent initiatives such as the VizBi [64] and BioVis [65] conference series may help to highlight work from computer visualization researchers. Another reason may be that, even if some CV papers are published in journals, some papers are only published as conference proceedings and may

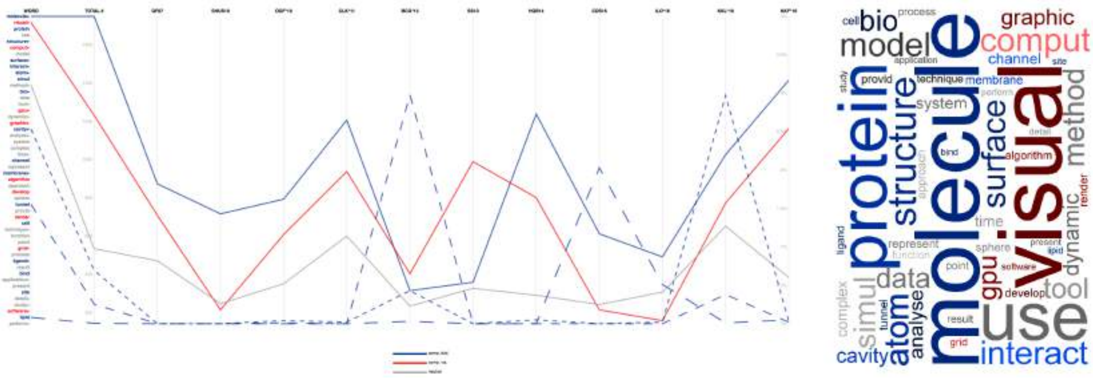


Figure 2.3: Result of the text analysis. Left: Parallel coordinate plot displaying the collective concordance of the most frequent words in each survey. Right: Word cloud based on the collective concordance ranking. The keywords is categorized by experts in both fields (category shown by color; blue: computational biology keywords; red: scientific visualization keywords; grey: neutral keywords).

not be referenced in scientific article databases such as PubMed [66] commonly used by CB researchers. This situation may cause large parts of CV research to be almost invisible to the CB community. Some CB journals also publish methods dedicated to molecular visualization and analysis such as *Journal of Molecular Graphics and Modeling*, *Journal of Computational Chemistry*, *PloS Computational Biology* etc. This topical intersection may create some competition with journals dedicated to computer science.

Shared References

These surveys share several references. Figure 2.2 shows that the most shared references are associated with software (VMD [25], Chimera [41], NAMD [67], and CAVER [68]). Only one reference comes from the CV field: Tarini *et al.* [69] presented an Ambient Occlusion method applied to molecular visualization. This paper also describes a software application, the molecular viewer QuteMol. Thus, making computer graphics research available, even just as prototype, is a key step to highlight CV researchers' work. Another good example is the fast *QuickSurf* molecular surface visualization by Krone *et al.* [70], which was published at a major visualization conference but was also made available in the popular molecular visualization tool VMD. This makes the method widely known in both fields, as can be seen in the number of citations as well as the usage and feedback by CB researchers. Three references shared by the selected reviews are survey papers: [54, 55, 71] with two of them discussed in section 2.2.

The last paper mentioned discusses the combination of molecular visualization and 3D printing [72]. The number of shared references in the selected survey is in very good agreement with the overall number of citations for each paper. I observed one clear outlier: the NAMD program for Molecular Dynamics simulations [67] which is important for creating dynamic models but is out of the scope of these surveys.

Text Analysis I performed a text analysis using the parallel coordinates plot depicted in Figure 2.3. An interactive version of the plot is available as supplementary material to allow interested readers to further investigate the data I collected for my survey. The interactive parallel coordinates plot is a useful way of exploring themes throughout the surveys. The user can exploit mouse motion to observe trends in the collecton of text over time. For example, if we hover the mouse over 'cavity' we can see that it is a popular topic in the surveys, i.e. [56] and [57]. Another example is with the term 'lipid' which reoccurs often in [35, 59] but is never mentioned previously, with the exception of Brezovsky *et al.* [56] , but only twice in the references. This may indicate an emerging important visualization topic. In contrast to the interactive plot that can show correlations or concordances between the individual surveys, the word cloud presented in Figure 2.3 gives a static overview of the most important keywords. This figure highlights biological topics (such as protein, cell, ligand, membrane, molecule, lipid) or a part of it (channel, cavities, atom, structure, tunnel) that can be interpreted as important application fields for CV reserachers. Some words are related to 3D objects (points, grid, surface, sphere) describing the essential graphical primitives used to render molecular objects. Some are potentially related to biological processes (binding, interact) which are important to analyse and visualize.

2.3 Survey of Molecular Dynamics Visualization Articles

The field of molecular visualization has received growing interest in recent years. As a result, a wide range of visualization techniques have been introduced to depict molecular dynamics simulation (MD) data. Fundamentally, the MD visualization literature can be divided into two major branches: 1) static visualization and 2) dynamic visualization. Static visualization is mainly used to depict the different attributes of molecules in a single time-step. On the other hand, dynamics visualization gives insight into behavior obtained from time-dependent molecular dynamics simulation. This section discusses the molecular dynamics visualization articles from both computational biology and computer graphics venues. Table 2.1 classifies

Reference	Visualization Theme		Cavities	Motions	Surfaces	Structures
	2D	3D				
Lindow et al.[7]		✓	✓			
Lindow et al.[8]		✓	✓			
Rozmanov et al. [9]		✓		✓		
Bhattarai and Karki [10]		✓		✓		
Medek et al. [11]		✓	✓			
Bidmon et al [12]		✓	✓			
Krone et al. [3]		✓			✓	
Krone et al. [13]		✓	✓			
Chovancova et al. [14]		✓	✓			
Fioravante et al. [15]		✓		✓		
Sehnal et al. [16]		✓	✓			
Krone et al. [17]		✓	✓			
Dabdoub et al [18]		✓		✓		
Skan̊berg et al. [19]		✓				✓
Blöchliger et al. [20]	✓			✓		
Zhou et al. [21]	✓			✓		
Byška et al. [22]	✓		✓			
Bhatia et al. [23]	✓			✓		

Table 2.1: Classification of the reviewed articles based on the visualization theme and visualization topic.

the reviewed articles based on the visualization theme and visualization topic.

2.3.1 Challenges

The significant growth in the molecular dynamic visualization field results in an enormous amount of research and publications. A researcher spends a significant amount of time on investigating what has already been done in his/her research area to identify the possible new contributions that they might add to the field. However, with the rapid growth of publications of molecular dynamic visualization, it might be quite challenging for the researchers to ensure that a specific idea has not been published before.

In this section, I address these challenges by providing readers with an overview of molecular dynamic visualization articles.

2.3.2 Survey Scope

There is much related work that contributes to the molecular visualization field. However, interactive visualization of time-dependent molecular data is my main area of focus. I do not focus on molecule dynamics literature that has been published out of this survey's scope such as:

- Molecular dynamics simulation without visualization: For example, Hopkins et al. [73] introduce and employ a new modified Mass Repartitioning (MR) schema by which they investigate advantages of utilizing (MR). The investigation shows an increasing in the simulation time step in a stable manner without significantly increasing truncation error.

- Molecular dynamics analysis without visualization: For instance, Roe and Cheatham [74] proposed two complementary tools (PTRAJ and CPPTRAJ) that facilitate processing and analysing time-dependent trajectory of atoms. These tools have been designed to read and display topology files. However, they are not practical for interactive visualization.

- Virtual Reality and hardware: For example, Bara et al. [75] introduce a methodology which is used to transform complex surfaces generated from MD simulations into .stl files that can be printed in any desired orientation. A 3-D printer can be utilised to provide a tangible output of the molecule data.

2.3.3 Literature Search Methodology

In this review I incorporate both manual and systematic search strategies including the IEEE Xplore, ACM digital libraries, EuroVis proceedings, Eurographics digital libraries and VisPubData [76]. First, a systematic search is used to identify the molecular dynamics visualization literature. Then, for each work found, the essentials are extracted using a special methodology (Laramee [77]). This is used to systematically summarize the related papers. In addition, molecular dynamics surveys, references in individual papers and my collaborator's recommendations papers are amongst my resources.

- IEEE Xplore digital library

- ACM digital libraries

- IEEE Transactions on visualization and graphics
- The Annual Eurographics conference
- The Annual BioVis conference
- VisBe
- The Annual EuroVis conference
- The Eurographics digital library

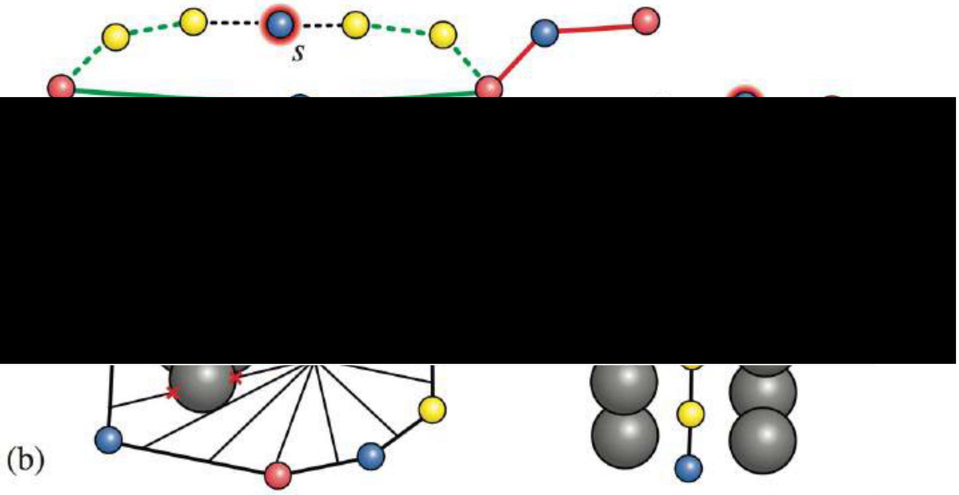
2.3.4 Molecular Dynamic Visualization

I utilized a simple classification to classify the literature. The selected articles are classified based on the main visualization theme: scientific and information visualization following by MD interaction and clustering visualization. An article that employs 2D abstractions to assist the visualization is classified under the information category, otherwise it is classified as a scientific visualization.

2.3.4.1 Scientific Visualization

Lindow et al.[7] propose a novel geometry-based path detection approach based on the computation of Voronoi diagrams. They compute and visualize the paths of all regions of a molecule utilizing many state-of-the-art methods and use lighting effects along paths to draw the attention of the user to the cavities and channels. The authors assume that a molecule is a set of spheres consisting of atoms and their position and Van der Waals radii. By computing the Voronoi configration of spheres from the Van der Waals spheres, the paths of interest are extracted as a subset of the topological structure of the distance transform of the molecule. Finally, they apply filtering to the Voronoi diagram. In comparison to [78], [11] and [79] this work is able to extract the paths from the exact distance transform of the molecule. In terms of Voronoi diagram computation, this work is about two times faster than that presented by Manák and Kolingerová [80].

In another work, Lindow et al.[8] extend their novel geometry-based path detection approach of a static coordinate snapshot [7] to molecular dynamics trajectories. They developed visualization tools that help explore the structure and dynamics of internal cavities in the bacteriorhodopsin proton pump. They denote a molecular path as a 3D curve joined by vertices who's

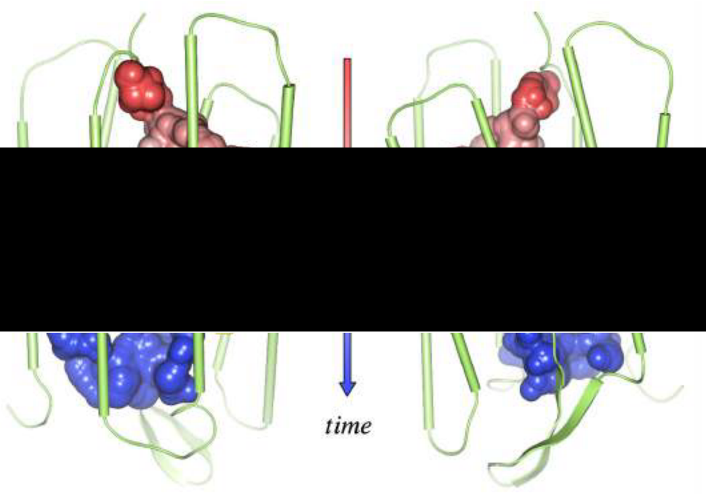


[redacted]

Figure 2.4: Two-dimensional sketch of cycle elimination. Image (a) shows the cycle detection algorithm for cycles of length 2, starting at s clockwise. The red paths fail, the green is the cycle and the dotted path is removed. Image (b) shows a non-empty cycle. If we remove the minimum m in image (c), we get two branches with a common start vertex, so the smaller branch will be removed (dotted). Image courtesy of Lindow et al. [7]

distance to the Van der Walls surface is maximal and above a minimum value. This threshold value approximates the radius of an ion or substrate as it moves within the molecule. The analysis is based on computing paths for all time steps that are connected. Krone et al. [13], and Raunest and Kandt [81] can be used to analyze cavities and channels in time-dependent data. However, they depend on the resolution of a grid data structure whereas Lindow et al.[8] compute the paths in each time step and are represented by an analytic description of the geometry as well as their extensions.

By focussing on spatial atomic densities, Rozmanov et al. [9] present a novel 3D histogram-based visualization method. The result of utilizing the technique is reported and covered in detail. The key is to sample the visualization of spatial information from molecular simulations and map the sampled information on a high-resolution 3D property grid over a given time frame. This histogram grid can be used to store average information about any property of the system. The grid can collect various types of data and represent scalar, vector, or tensor property fields such as density, potential energy, mobility, order parameter, linear momenta, force, or stress tensors. The averaging is accomplished for a spatial location rather than for a specific atomic trajectory. The property grid can be manipulated to produce more complex



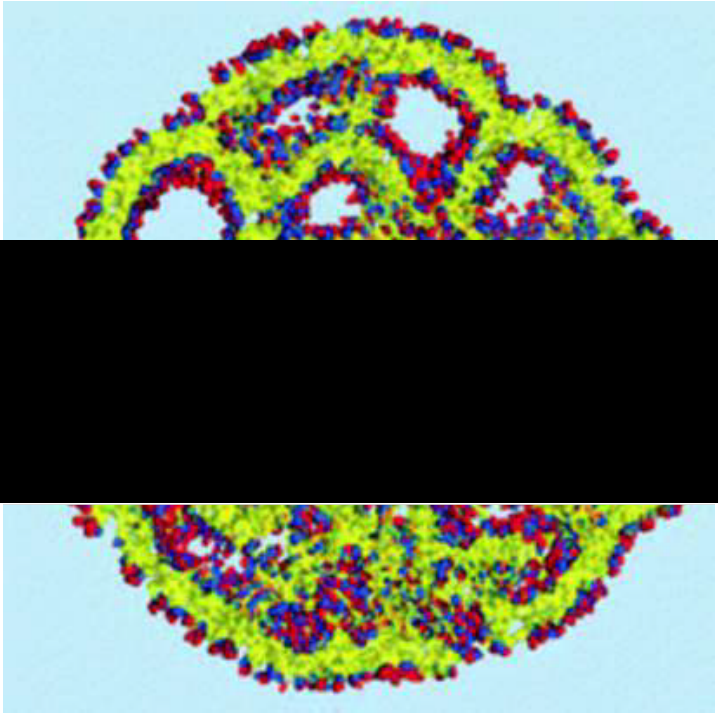
[redacted]

Figure 2.5: Visualization of a dynamic molecular channel. The surface shows the accumulated components of one path component traced over time. The time is shown by pseudo-coloring the extension surface. Here, red represents an early time and blue a later time. Image courtesy of Lindow et al. [8]

fields or directly visualized via appropriate visualization tools. VMD [25] utilizes various illustrative rendering representations which are only applicable to proteins. Razul et al. [82] and Vatamanu and Kusalik [83] use average configurations for the visualization of ordered systems. However, the method is poorly suited for disordered systems. The method can be considered a natural extension and combination of Razul et al. [82], Vatamanu and Kusalik [83] and Svishchev and and Kusalik [84].

Bhattarai and Karki [10] introduce a novel interactive visualization tool for analyzing and visualizing space-time multi-resolution atomistic simulation data. Bhattarai and Karki [10] use two phase rendering approach. First, an overview of all the data set is obtained by applying animation and color map to particles and their path-line. Then, they generate additional information on the fly and analyze and visualize them using a combined graph-theoretic and statistical approach.

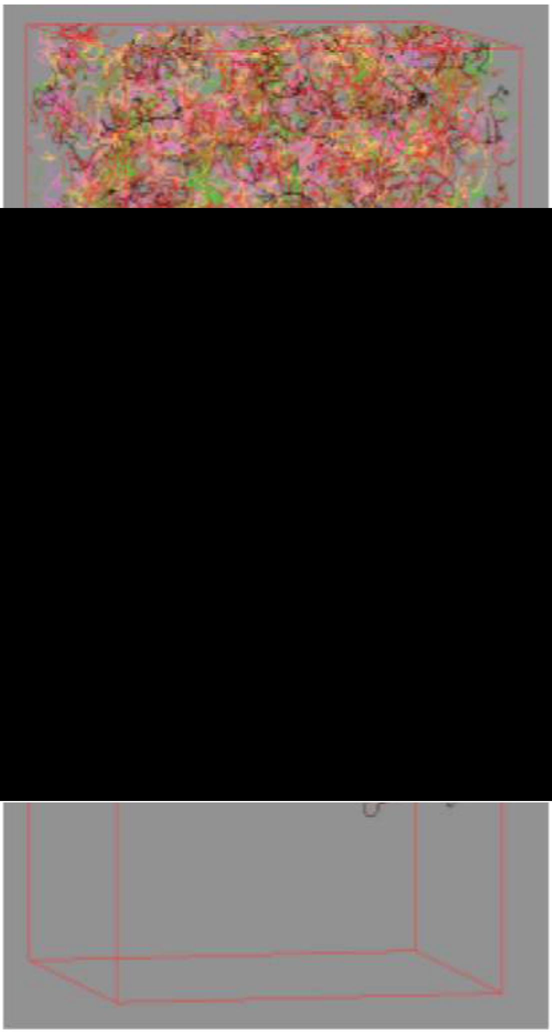
Medek et al. [11] introduce a novel approach for analysing a cavity in protein molecules. The method of Medek et al. [11] is based on the geometry of the protein. The molecule is represented by a set of spheres i.e each atom is represented as a sphere and it has a center point and radius. For each atom, a Voronoi cell is defined as a set of points. The union of all Voronoi cells is a voronoi diagram. Medek et al. [11] utilize duality between a Voronoi



[redacted]

Figure 2.6: The density of the DLin-KC2-DMA cationic lipid in the nano-particle. The densities of the three parts of the lipid molecules were sampled on separate grids and plotted as separate iso-surfaces through 3 nm3 densities: the hydrophilic head groups (red), linker groups (blue), and the hydrophobic tail groups (yellow). Image courtesy of Rozmanov et al [9].

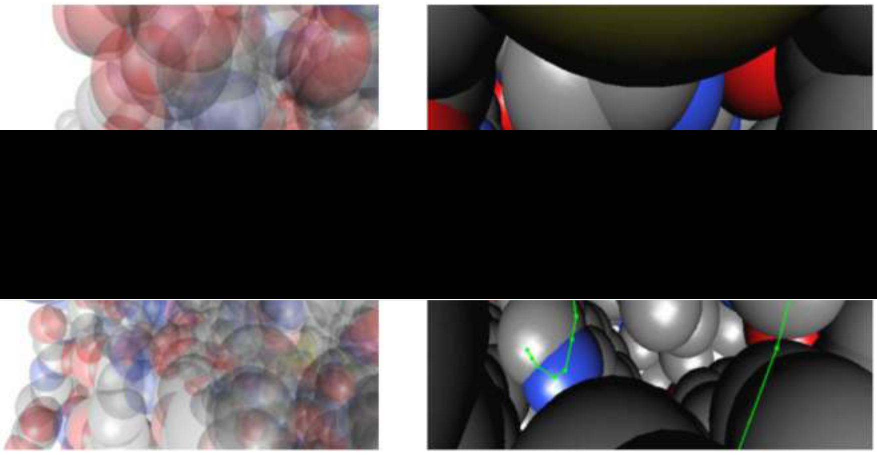
diagram (VD) and a Delaunay Triangulation (DT) to identify tunnels. A VD is used to compute the tunnel by finding the narrowest passage of every tunnel. The narrowest location is be located either via the Voronoi edge intersections or via the voronoi edge endpoint. The DT is interpreted as a weighted graph which is used for computation of the ideal tunnel. CAVER 1.0 [68] computes tunnels by utilizing a space discretization technique. The technique requires frequently sampling the space that contains the molecule and evaluating the samples in the 3D raster (voxels) based on distances from the nearest atom. The greatest minimal distance is used to find tunnels. Liang et al. [85] analyse cavities based on α-shape. Their tool deals with overall cavity analysis and does not require any used input. However, as it is designed as a general solution it can not be used for tunnel computation. In comparison to the previous work, Medek et al. [11] provide a hight quality result within acceptable computational time. However, for identifying tunnels over a long time Medek et al. [11] compute each tunnel separately for each time step.



[redacted]

Figure 2.7: Pathlines of all 64 atoms (top) and the pathline of one selected atom (bottom) of the liquid MgO at 4000 K. The bottom clearly shows that the particle wanders in different regions as time elapses. Image courtesy of Bhattarai and Karki [10].

A novel approach of identifying and visualising the pathways of solvent molecules in and around protein cavities is introduced by Bidmon et al [12]. The visualisation focuses on the velocities and residence times of water molecules following trajectories in the solvent. First, the solvent molecules' pathlines are calculated by using a special purpose tool which is designed for this purpose. The result, including lengths of the extracted pathlines together with the average path lengths and the exits taken when entering and leaving the cavity, is stored offline. Second, before exploring the data, two techniques are used in order to reduce the amount of

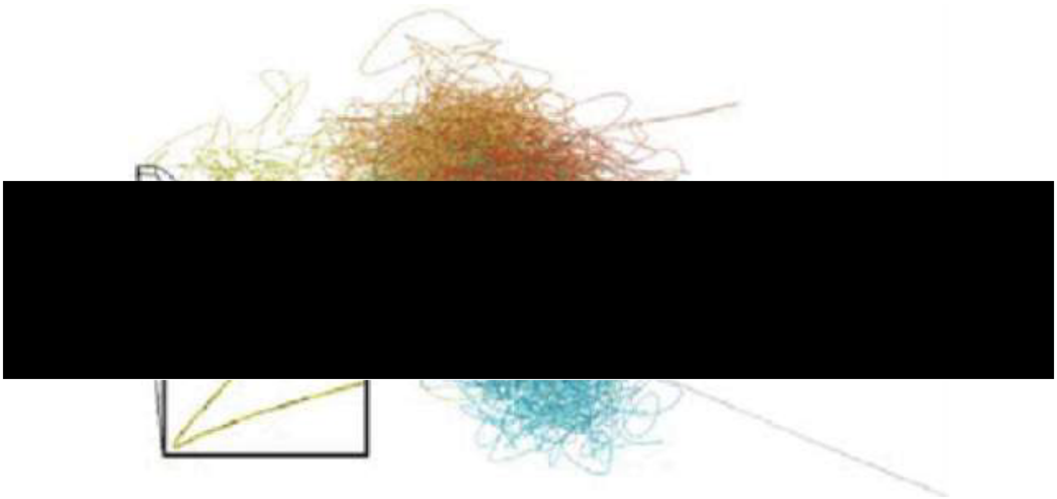


[redacted]

Figure 2.8: Representation of ideal Tunnel (left) and Tunnel centerline by a simple line (right). Image courtesy of Medek et al [11].

extracted data in order to minimize path clutter: 1) Applying a superposition of each frame in the trajectory onto a common reference frame by utilising AMBER [30], 2) Selecting a region of interest (ROI). The whole trajectory of the extracted path is read to get all positions of solvent molecules within the selected area. Then paths are filtered based on the user-chosen frames. Finally Bidmon et al. [12] utilize color mapping to visualize the dynamic properties of paths by using a specialized clustering algorithm with respect to these properties. Bakowies and Van Gunsteren [86] identify a tight hull volume of the cavity and detect water molecules inside. However, water molecules aren‘t tracked and visualized the pathways inside the cavity. Therefore, no information is provided of whether water molecules enter and leave the cavity by the same exit. In Bidmon et al. [12] pathlines are clustered and visualised as tubes to handle the numerous pathlines resulting from a large trajectory.

Krone et al. [3] present a novel interactive visualization approach for molecular surface dynamics. The presented method enables interactive exploring and analysis of large, long time-depenedent tragectories of proteins. In addition, to reduce the visual complexity of the surface, Koren et al. [3] propose a semantic simplification of the raw protein data. The implementation relies on out-of-core visualization techniques and the structure of Van der Waals (VdW) surfaces as defined by Greer and Bush [87]. Krone et al. [3] apply GPU ray casting techniques to the implicit mathematical description of the Solvent Excluded Surface (SES). The SES is described by three primitives (spherical patches, toroidal patches, spherical triangles). OpenGL Shading Language (GLSL) -fragment and vertex- shaders are implemented for the

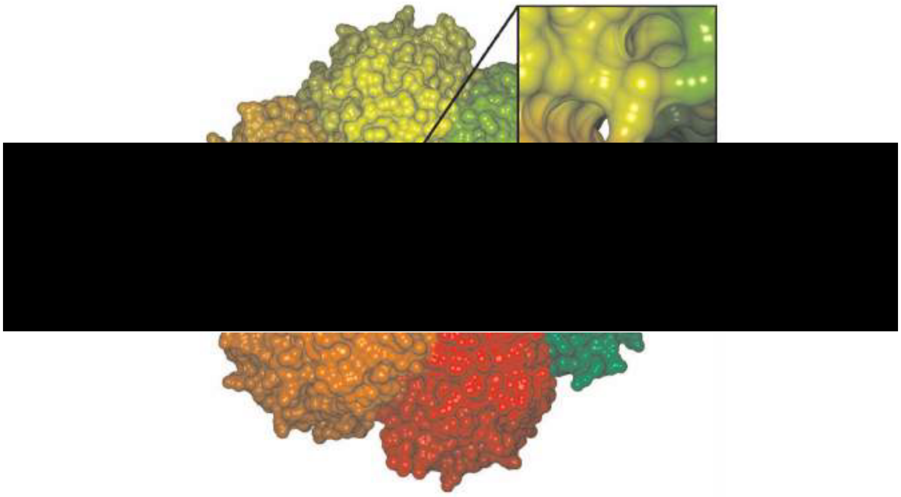


[redacted]

Figure 2.9: Properties mapped to the pathlines: The time within the trajectory is represented by a colour gradient from red (start of the trajectory) over yellow and cyan, to blue (end of the trajectory). An additional luminance ramp encodes the molecule's direction of motion as can be seen in the cutout. The velocity is encoded in the saturation. Fast pathlines (on the right side) are shown in gray. Image courtesy of Bidmon et al. [12]

three primitives in order to be used in ray casting. The work load is distributed between the fragment shader and vertex shader to obtain the optimal rendering time. In comparison to PyMOL [42] and VMD [25], Krone et al. utilise the program MSMS by Sanner et al. [88] which recomputes the SES completely for every frame which in turn result in consuming twice the memory and almost double the computation time compare to Krone et al. [3]. Despite the fact that VMD [25] and Chimera [41] compute the SES on-the-fly every frame, each frame takes several seconds to render thus impeding trajectory analysis. Krone et al. [3] visualize the SES of the trajectory in a cache-friendly manner which enables the analysis of trajectories.

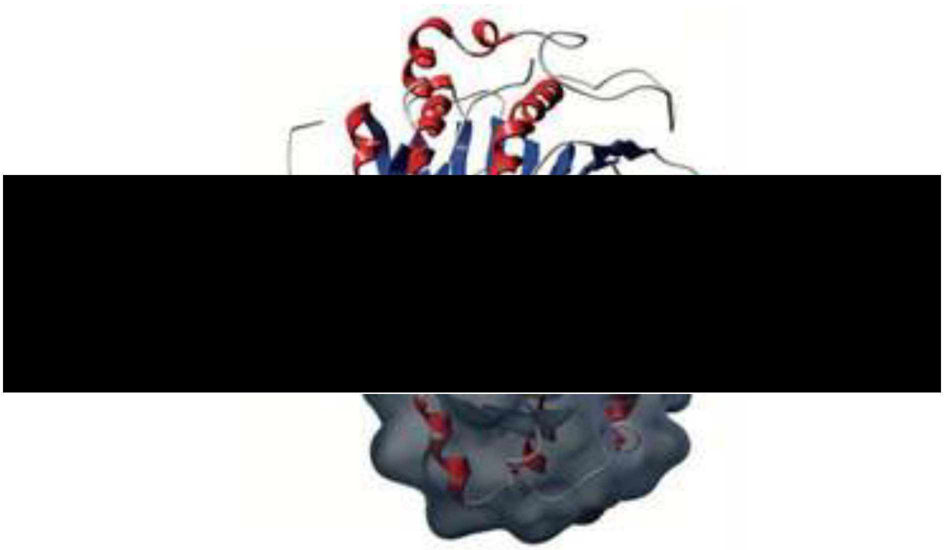
Krone et al. [13] introduce a novel interactive tool for analysing and visualizing protein cavities in molecular dynamics. Three diffrent views are provided by Krone et al. [13]: A 3D visualization view which shows the protein, a clipping plane view, and a graph that is used to plot the size of the focus cavity. A semi-transparent isosurface of a protein is displayed by the 3D visualization. An illustrative rendering model is utilized to simplify the representation of the secondary structure of protein. The position of individual atom is shown by a Ball-and-Stick model. A clipping plane is applied to the molecule to select a cavity by clicking on the hole in the 2D plane. A 3D flood fill segmentation is used to extract the selected cavity. A 2D graph supports the analysis. The time step of the trajectory is represented by x-axis of the graph



[redacted]

Figure 2.10: A coupling protein (PDB-ID: 1GKI) which is used for performance measurements, color-coded by amino acid chain. Image courtesy of Krone et al. [3]

while the qualitative size of the given cavity is mapped to the y-axis. Krone et al. [13] use a fast density segmentation to extract a cavity. They use partial segmentation rather than segmenting the whole volume. CAVER (Summary on page 46), MOLE [78] , PocketPicker [89], LIGSITE [90] and McVol [91] are used for the detection of cavities in proteins. However, they do not support the interactive visualization of dynamic data. Krone et al. [3], [92] and Lindow et al. [93] utilize GPU ray casting and efficient parallelization to render the SES. However, the classical SES results in visual distraction when rendering the surface semi-transparently. Cipriano and Gleicher [94] proposed a mesh-based simplification of the SES. Giard and Macq [95] and Dias [96] use the Marching Cubes (MC) algorithm to extract isosurfaces. However, their implementation is not fast enough for interactive use. Can et al. [97] extract a molecular surface from the volume by using Level Sets. Phillips et al. [98] extract all cavities of the protein by using direct volume rendering and perform an exhaustive 3D flood fill segmentation. However, these approaches are not designed to be used with dynamic data sets. Chovancova et al. [14] introduce a new version of CAVER 2.0 [99] and propose a novel algorithm for the calculation and clustering of trajectories. The new version (CAVER 3.0) helps analyze tunnels and channels in large collections of protein structures and provides information of individual transport pathways and their time evolution. In order to differentiate between pathways based on their characteristics, Chovancova et al. [14] introduce the following terminology: A **Channel** is



[redacted]

Figure 2.11: Illustration of a segmentation result (circled yellow). The selected cavity inside the isomerase is visualized by yellow spheres for each voxel. Image courtesy of Krone et al. [13]

a pathway that leads throughout the protein structure with both sides open, the surrounding solvent and without any interruption by an internal cavity. A **Tunnel** is a pathway that either connects a protein surface with an internal cavity or connects more than one internal cavity. Chovancova et al. [14] employ a Voronoi diagram by using fixed size spheres to approximate each atom individually in the protein analyzed structure which results in reducing calculation error. Paths' geometrical distance is used to evaluate similarity between paths. For flexibility, the CAVER 3.0 algorithm involves three independent steps. First, trajectories in each given protein structure are identified. The identification of pathways is obtained by constructing a Voronoi diagram with some optimization by utilizing Delaunay triangulation of vertices and edges of the Voronoi diagram. A transport pathways search algorithm is used to identify pathways and redundant pathways are removed. Then, pathways are clustered by utilizing the average-link algorithm based on pairwise distances of the pathways. Finally, the priority of the resulting pathway clusters is derived and used to rank them. The output data is provided in image and .CSV file format. HOLE 2.2 [100], CAVER 1.0 [68], MOLE 1.2 [78] and MolAxes [79] and [101] are used to identify pathways in molecular dynamics simulations. However, they are very limited in studying dynamic systems. Even though HOLE 2.2 [100] can be used for the the analysis of ensemble structure, its application is restricted only to the analysis of a

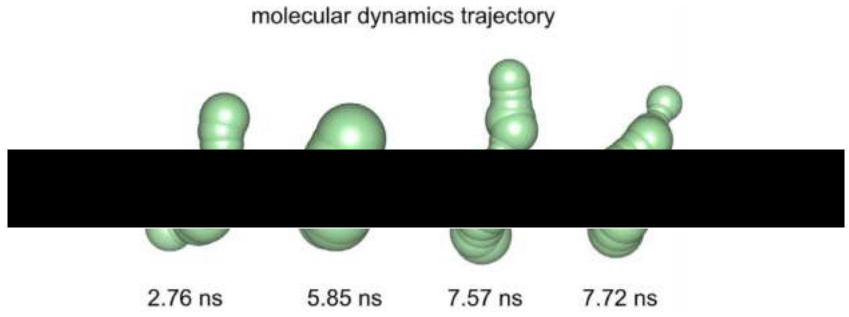
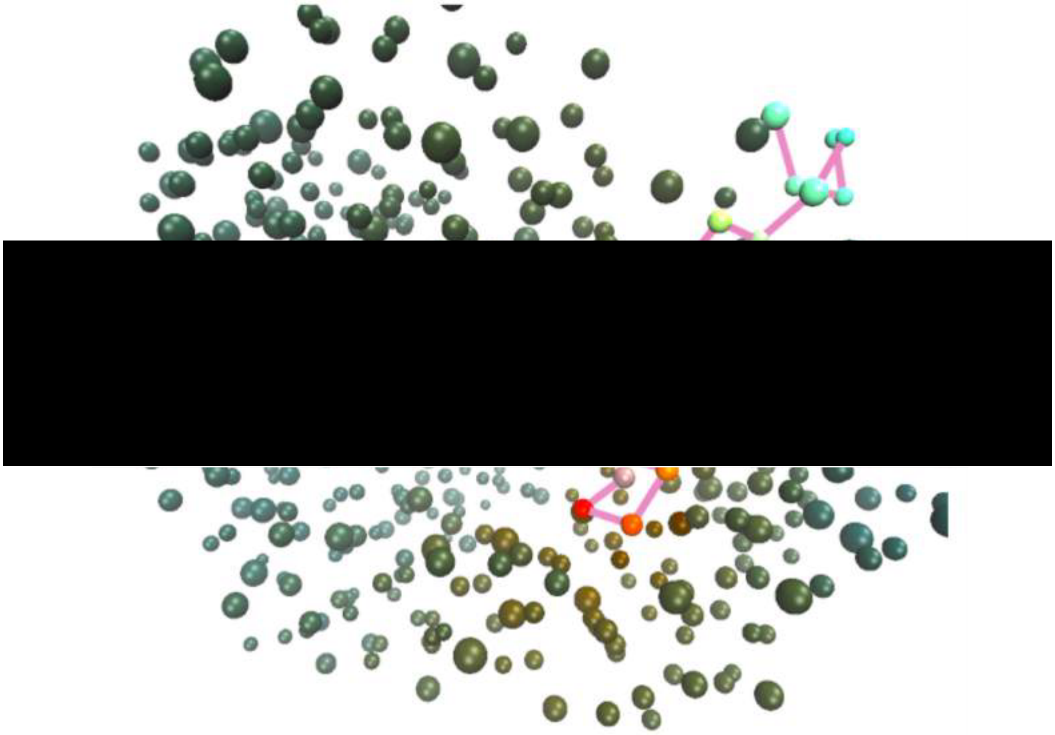

[redacted]

Figure 2.12: The p1 tunnel identified in the 2.76 ns, 5.85 ns, 7.57 ns and 7.72 ns snapshots of the MD trajectory of DhaA. Image courtesy of Chovancova et al. [14]

single channel. On the other hand, Chovancova et al. [14] supports the analysis of tunnels and channels in large ensembles of protein structures.

Fioravante et al. [15] propose novel visualization techniques for investigating molecular dynamics motional correlation. Residues are represented as spheres. In addition to the overall view of the molecule, four visualization techniques are introduced by utilizing covariance matrices (dimension reduction, ray tracing and geometric deformation). The first technique utilizes dimension reduction matrix and defines a mathematical function, which is called importance metric, to divide residues into two sets of important and unimportant residues. The important set includes residues that have high levels of correlation or anti-correlation selected by user. The technique visualizes the important subset of residues and color maps residues based on the relationship between a given residue and the rest of molecule. The second technique utilizes ray casting to provide a general overview by visualizing the interaction between residues with each other. The surface color of each sphere is updated based on the other residue covariance values in the domain. The third technique uses a geometric deformation matrix to visualize the global correlation for all residues sphers. Cones' size and direction are used to visualize the direction of correlation and/or anti-correlation. The fourth technique is to visualize an allosteric path. Fioravante et al. [15] define an allosteric path to be a successive grouping of residues whose motion is highly correlated. The allosteric path is visualized by connecting the spheres with lines. Bryden et al. [102] use 3D glyphs to visualize the coarse motion of a subset of the element. However, the method doesn't provide visualization of how allosteric impacts movement through the molecule. Barlowe et al.[103] employ feature identification and comparison on two-dimensional plots to study the motion of a molecule. However, the data doesn't apply to the three-dimensional location of the molecule. Fioravante et al. [15] visualize the



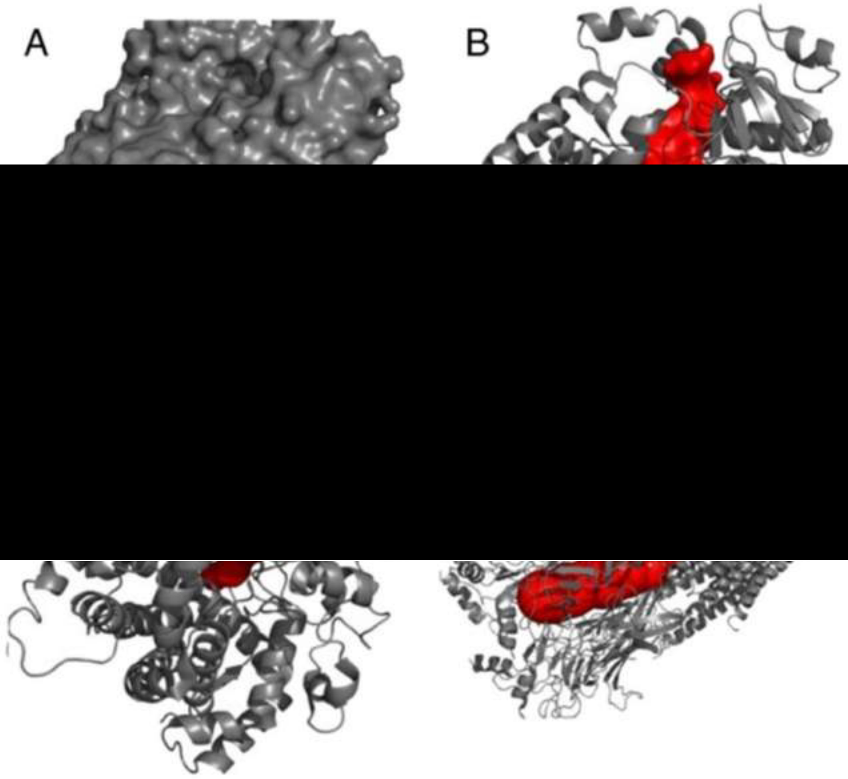
[redacted]

Figure 2.13: An allosteric path. Image courtesy of Fioravante et al [15].

allosteric path and provide user with a 3D depiction of the data.

Sehnal et al. [16] introduce a novel software tool for identification of channels and pores. They proposed a fast and precise algorithm for finding channels. In order to identify channels, the algorithm performs seven steps: First it computes the Delaunay triangulation of the atomic centers in order to construct a Voronoi diagram as the dual of the Delaunay triangulation. Then, the molecular surface is approximated by removing all tetrahedrons that are extracted at the interface between the molecule and the external environment. Furthermore, boundary tetrahedrons which are too large to fit a sphere with a given probe radius or too small to fit a sphere with interior radius are removed. If components contain at least one tetrahedron on the molecular surface then they are called cavity diagrams. After that, possible start and end points of channels are identified by either an automatic or user defined approach. The shortest paths between all pairs of start and end points in the same cavity diagram are used for computing channels. During the computation, a sequence of tetrahedrons is used to represent channels and a natural cubic spline of the circumsphere centers of the tetrahedrons is used to approximate the channel centerline. Finally, channels are filtered with two criteria. The first one is to remove channels that have a bottleneck with a small radius and the other one is to discard channels that are very similar to each other. Utilizing this novel algorithm results in increased speed, accuracy and robustness of channel and pore identification. Furthermore it is capable of correctly identifying channel start points and computing pores by merging channels. POCKET [104], LIGSITE [[105],[90]], dxTuber [81], HOLLOW [106], 3V [107], CAVER 1.0 [68] and CHUNNEL [108] utilize grid-based approaches to identify pockets of tunnels. FAST [109] and SURFNET [110] use sphere-filling methods. HOLE [100] and PoreWalker [111] implement slice and optimization methods. MOLE 1.x [78], MolAxis [[79], [101]], CAVER 3.0 (summary on page 46) utilize Voronoi diagrams. However, Sehnal et al. [16] provide an improved method for channel identification. Their method utilizes Delaunay triangulation and a Voronoi diagram and increases speed, accuracy and robustness by performing several preprocessing steps. In addition, Sehnal et al. [16] provide two unique methods which have not been implemented in the previous tools i.e cavity computation and computation of physicochemical properties.

Krone et al. [17] introduce a novel real-time and interactive visualisation tool for analysing dynamic simulation data of protein behavior with a focus on the analysis of cavities and binding sites. The tool provides users with both automatic and manual filtering options. In this work, ambient occlusion is utilised to extract and visualize the conformation of a protein. The cavity extraction process relies on classifying cavities and extracting their centerlines. Information about the possible binding sites of the protein-connections are obtained from the cavities. In addition, the user may add and map chemical information about binding sites to the visualisation. Finally, the authors use brushing and linking to explore the extracted information. In terms of the visualisation tools, PyMOL [42], [112], VMD [25] and Chimera [41] are some of the most popular free visualisation tools. However, they are too slow for interactive cavity extracttion. In addition, in some cases they do not offer sufficient functionality (such as analysing and classifying cavities) whereas Krone et al. provide these functionalities with sufficient speed. POCASA [113] offer 3D visualizations to show the analysis results for static data. Lindow et al. [8] and MD-pocke [114] implement similar approaches based on computing Voronoi diagrams for all time steps to extract cavities. However, the cost of the computation of Voronoi diagrams is high and time-consuming. Parulek et al. [115] utilise the distance to the molecular surface and its gradient to extract cavities. Even though the results for all time steps are shown

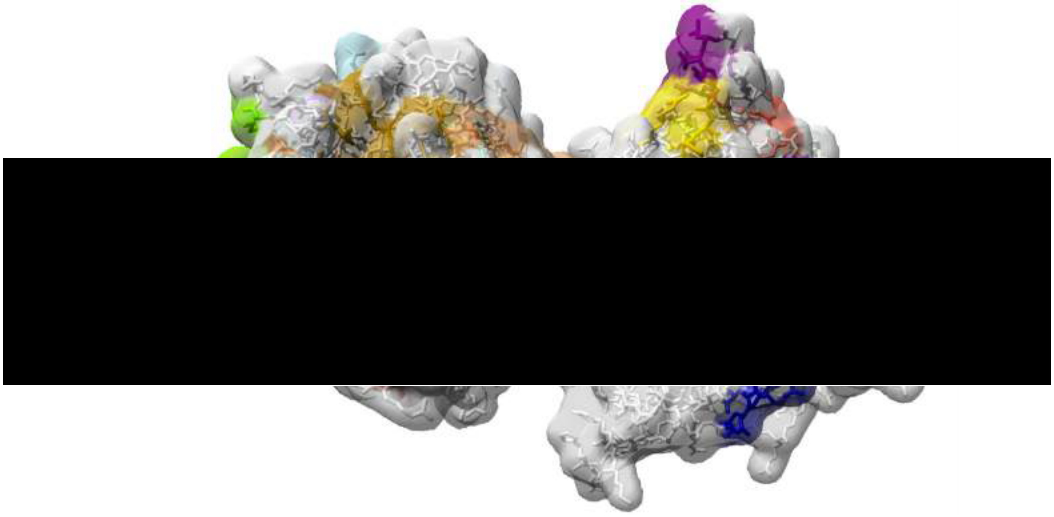


[redacted]

Figure 2.14: Classification of biomacromolecular ”empty spaces”: A) pockets, B) cavities, C) channels (or tunnels), and D) pores. Image courtesy of Sehnal et al. [16]

in a scatter plot, the extracted cavities aren’t correlated. In comparison to these approaches, Krone et al. rely on improving their previous work [116] which is suitable for dynamic data and offers high performance especially on the GPU.

Dabdoub et al [18] introduce an interactive web-based molecular dynamics visualization system. The system employs a Pathline-analogous representation and provides two services, 1) Visualizing molecular motion and 2) Producing output files that can be printed via 3D printers. The system requires a sequence of molecular poses (set of discrete molecular) as input data. The visualization process starts by extracting the backbone alpha carbons from the input data. Then, smoothed pathlines for atoms is obtained as follow, for each backbone alpha carbon the intermediate positions are interpolated by constructing a natural cubic spline that passes through the expected positions of each carbon atom over time. The system provides a variety of temporal resolution in the input series of poses, and a GUI control that enables the user to render only a subset of input poses. Each pathline is color coded based on timestep. Finally,

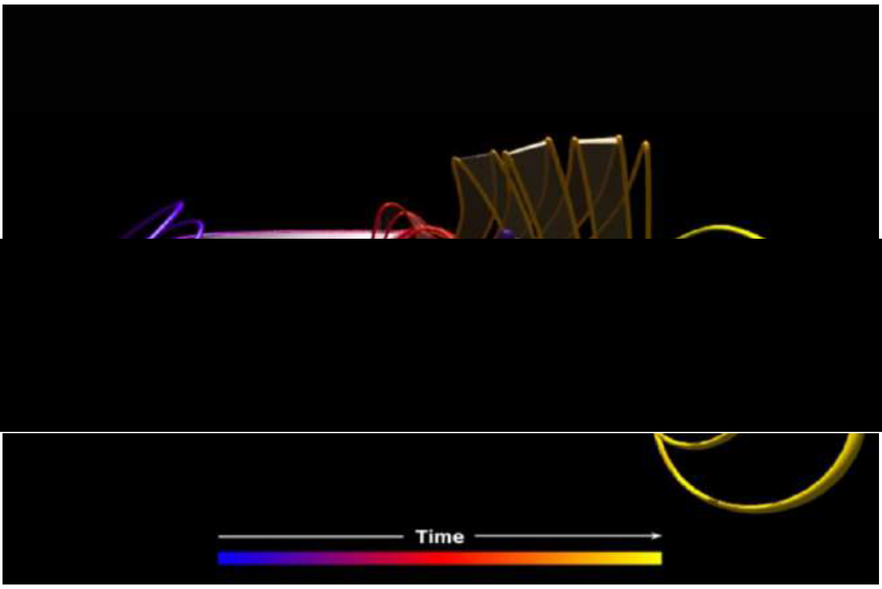


[redacted]

Figure 2.15: Semi-transparent molecular surface of a lipase (PDB-ID: 4FKB) colored by binding site information combined with a stick model. Image courtesy of Krone et al. [17]

the spline is rendered as a piecewise-continuous cylinder in both on-screen and the printable output file. Schmidt et al. [117] apply Computational Fluid Dynamics (CFD) visualization methods to generate isosurfaces from conformational densities of metastable molecules. Bhattarai and Karki [118], Bidmon et al. [12], Falk et al. [119] and Joshi and Rehingans [120] use pathline representation. However, none of these deal with the composite trajectory of the backbone structure for a single molecule.

A novel method for visualizing atomic molecular structures is proposed by Skånberg et al. [19]. The proposed approach involves a novel visualization algorithm for space filling representations of molecular structures and utilizes mutual interreflections as a new visual communication channel for visualizing pairwise interaction strengths. Furthermore, Skånberg et al. introduce symbolic regression into the visualization field as an approach to analytically capture multidimensional functions. Skånberg et al.[19] is based on capturing the diffuse interreflections between atoms, in a physically-based manner. However, to reduce the computational complexity of handling the interreflection of multiple atoms they consider two isolated atoms (The reflections resulting from several neighboring structures can be reconstructed by modulating the reflection contributions of pairwise combinations of these structures). The diffuse interreflections between atoms is captured via a derived function. Then, to achieve interactive frame rates, they employ symbolic regression to find an analytic expression that can be imple-



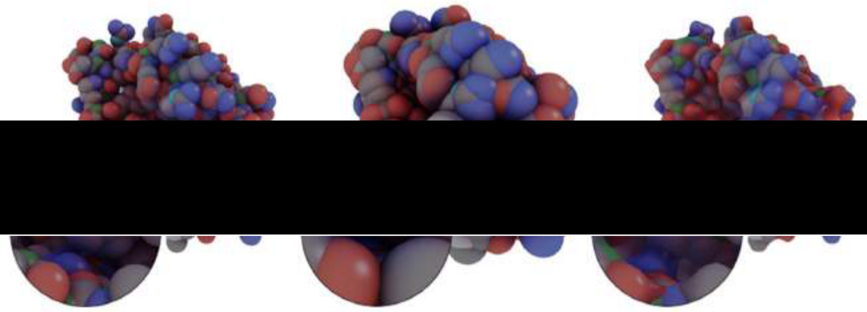
[redacted]

Figure 2.16: Visualization of six alpha carbons from the middle of the protein structure. These visualizations illustrate the contrast between the seemingly simple swinging motion in the composite image and the clearly more complex motion revealed by the pathline rendering. Image courtesy of Dabdoub et al [18].

mented on the GPU. Kajiya [121] describes light transport in a geometric scene which is the original inspiration behind this work. However, Skaňberg et al. eliminate the radiance emitted from a single surface point by introducing environmental lighting as the only source of energy in the scene instead and modify the path tracer to compute the radiance distribution on the surface of the atom rather than the image plane. In terms of the fixed-radius neighbors search technique used in this work, Skaňberg et al. [19] adopt Hoetzlien's [122] technique in order to be able to fetch the neighboring cells in constant time. However, it requires liner time with respect to the number of atoms per cell.

2.3.4.2 Information Visualization

Blöchliger et al. [20] build an instantaneous graph of the entire time-dependent molecular dynamics simulation. The proteins and biomolecules are clustered in the graph representation such that the graph can be queried for the most interesting simulation states and kinetic pathways. For the entire dataset, all timesteps are represented as a complete graph. The graph vertices correlate with each timestep and edge weights are given by the pairwise distances between



[redacted]

(a) vdW (b) SAS (c) SES

Figure 2.17: Application of the proposed technique to the 3RH8 molecule for (a) van der Waals (vdW), (b) solvent-accessible surface (SAS) and (c) solvent-excluded surface (SES) representations. While vdW and SAS are directly supported, SES requires an additional interpolation. Image courtesy of Skan̊berg et al [19].

timesteps. The minimum spanning tree is computed. From an arbitrary starting timestep, the snapshot linked by the shortest edge is added to a sequence of timesteps, the so-called progress index. This sequence proceeds through regions of high sampling density one after another. The progress index is independent of input order and it can be interpreted kinetically and structurally to provide a comprehensive representation of all states visited by the input trajectory. Blöchliger et al. [123] introduce an adaptable method that hierarchically groups timesteps of a complex system into kinetically distinct sets. Blöchliger et al. [20] apply this method to two molecular dynamics trajectories of a protein.

Zhou et al. [21] introduce a novel method (Sigma-r) that provides a simple measure of the global dynamics of a macromolecule. A Sigma-r plot is a plot of the average standard deviation of interatomic distance (σ) as a function of interatomic distance (r). The approach starts by looping through the trajectory file. First, for each time step, all interatomic distances are calculated and accumulated. Then the standard deviation of each interatomic distance is calculated. During the first step, the value of the interatomic distance is sorted from shortest to longst. Then, the standard deviations are averaged within intervals with step size Δr in order to determine the average α value for this interatomic distance interval. Zhou et al. [21] observe that α tends to increase with interatomic distance and most of the sigma-r plots exhibit a trend that can be modeled with an exponent between 0 - 1.0 similar to Neighbor Model and Rigid Body Hinge Model (Makowski et al[124]). Zen et al. [125] introduce a method to represent the global protein dynamics based on correspondences between low energy modes in enzymes.

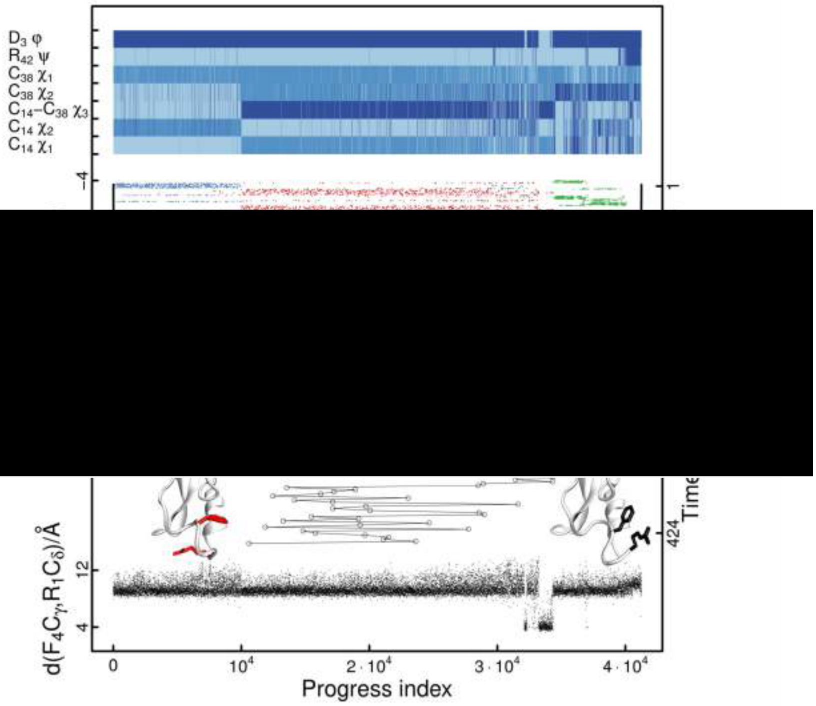
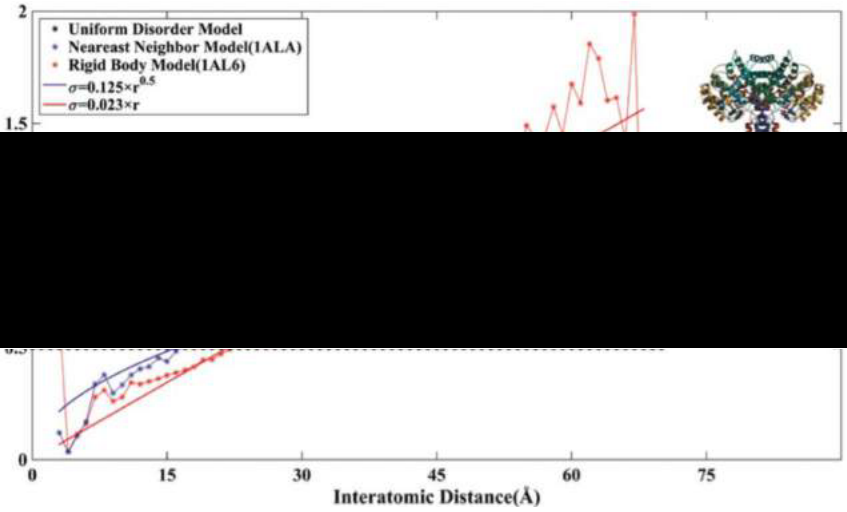

[redacted]

Figure 2.18: (Upper panel) The progress index, of 41250 timesteps from 1.03 ms of MD data. Lower panel) Zooming on a thin time slice of the dynamical trace to visualise a particular transition from the red to the black state. Image courtesy of Blöchliger et al [20].

Munz et al. [126] compare matrices of the standard deviation of all interatomic distances in the protein through a pathway. However, these matrices involve considerable amount of information that can't be used in comparing the dynamic behaviors of different proteins. Zhou et al. [21] reduce the information in a trajectory by utilizing sorting and averaging the standard deviation as a function of the interatomic distance.

Byška et al. [22] introduce a novel interactive approach for exploring protein tunnels. The technique enables a user to explore the tunnel surroundings for individual time steps in detail and utilizes a new quantitative visualization for molecular void spaces. This work consists of two main parts, the first one is the representation of the tunnel width and length and their changes over time by utilizing a function graph which describes the tunnel width along its centerline. The Chovancova et al. [14] algorithm is used to detect all tunnels satisfying the input parameters for each time step. The correspondence between tunnels over time is obtained by performing a clustering on the detected tunnels. The second stage is exploring the lining of



[redacted]

Figure 2.19: Example of sigma-r plots for three interatomic movements models, the Nearest Neighbor Model and Rigid Body Model. Image courtesy of Zhou et al [21] .

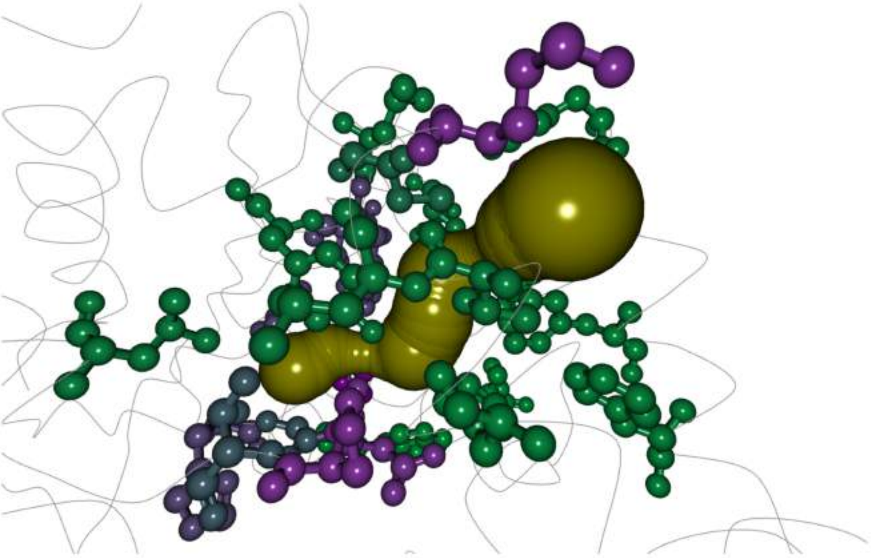
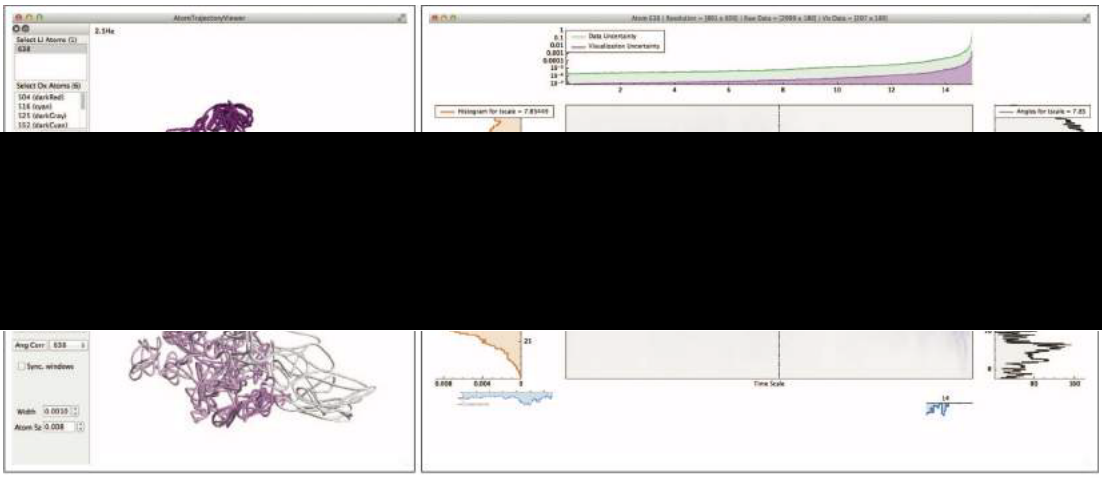


Figure 2.20: 3D visualisation of a tunnel (in green), its surrounded amino acids are colored with respect to their hydrophobicity. Image courtesy of Byška et al [22].

amino acids of the tunnel in detail throughout the whole dynamics. In order to overcome the visual clutter caused by overlaps, a highly abstracted 2D representation for lining amino acids is used rather than a 3D view. The exploration approach starts by calculating a set of amino acids that surround the tunnel. Byška et al. [22] incorporate a user defined value as a threshold and define amino acids as tunnel lining if their distance to the tunnel centerline is smaller than the threshold. Next, mapping is applied by unfloding the tunnel in 2D along its centerline. The mapping involves two steps 1) Computing the influence for each amino acid and for each single time step, 2) Finding the nearest tunnel sphere for each atom of the lining amino acid. Then, for all amino acids and in all snapshots the results are displayed below the TunProfile view. Finally, ranking is used through changing the ordering by giving each amino acid a score which influences its vertical positioning (A higher score means that the amino acid will be depicted at a higher position in the AAExplorer). Petřek et al. [68] use a grid-based approach to detect tunnels. However, the grid resolution plays a critical rule in terms of the precision and the performance. Sehnal et al. [16], Lindow et al. [7], Medek et al. [11], and Petřek et al. [78] improve the grid-based approach by subdividing the protein space using a Voronoi diagram by defining the shape of each cell according to the position of individual atoms. This results in a great improvement to the accuracy. In terms of identifying and tracking a single tunnel, the evolution of a single cavity for all time steps is analyzed and the results are aggregated by Lindow et al [127]. Chovancova et al. [14], and Sehnal et al. [16] cluster the tunnels and compute the evolution of individual tunnels for the entire molecular dynamics simulation. However, the evolution is visualized and represented with a focus on 3D view which results in visual clutter caused by large overlaps. Furthermore, it suffers from less understandability when analyzing large simulations. In terms of data reformation, Kanitsar et al. [128] present a method to generate Curved Planar Reformation (CPR) to deal with the challenge of occlusion by mapping tubular structures such as blood vessels onto a 2D plane. Bidmon et al. [12] analyze solvent pathways and their behavior in molecular dynamics then cluster similar pathways to reveal the principal paths. However, current existing solutions support displaying the information about the lining amino acids for a given time step. Whereas Byška et al. [22] show all lining amino acids throughout the whole dynamics and even explore their properties.

Bhatia et al. [23] propose a novel interactive histogram for molecular dynamics and 3D visualization of trajectories. The tool based on Relative-Angle Distribution taking into account data uncertainty and visualization uncertainty. Bhatia et al. utilize an interactive histogram to visualize the relative-angle distribution with respect to space-time. The angle-distribution is visualized as a 2D image by mapping bins to a single color hue. The time-scale increases left to right, and the angle increases from bottom to top. The histogram is linked with a trajectories visualization tool. The user is able to select multiple or individual atoms to be visualized with



[redacted]

(a) 3D visualization of trajectories.

(b) 2D visualization of relative-angle distribution statistics.

Figure 2.21: Screenshots of Bhatia et al. [23] integrated analysis and visualization tool for interactive exploration of atomic trajectories. Image courtesy of Bhatia et al. [23].

or without chemical bonds. The relative-angle distributions tool visualizes four data attributes: 1) Multiple distributions as a 2D image, 2) A 1D histogram, 3) Uncertainty plots, and 4) Angle vs. time. The relative angle quantifies the direction of motion over successive time intervals Burov et al [129]. Savage and Voth [130] use 2D visualization for multiple 1D histograms for different time-scales. However, Bhatia et al Bhatia et al. [23] utilize interactive techniques and reduce visualization errors.

2.3.5 Visualization of Protein Interaction and Clusters

The field of molecular visualization has received significant attention in the last decades. For an overview, I recommend the following recent survey papers on molecular visualization. O'Donoghue *et al.* [54] review visualization methods and tools that enable the community of life scientists to obtain insight into their molecular data. Gehlenborg *et al.* [131] review stand-alone and web-based applications that are used to visualize PPI with a focus on 2D visualization techniques such as interaction networks, scatter plots, heat maps, dendrograms and graphs. In two recent state-of-the-art reports, Kozlíková *et al.* [53] review and classify visualization techniques developed to render molecular structures and Krone et al. [57] review the visual analysis of biomolecular cavities. Most recently Miao *et al.* [132] provide a high-level survey of multi-scale molecular visualization techniques focusing on application domain

		Ref.	PLI		PPI		Clustering		Vis. Capacity	
			2D	3D	2D	3D	2D	3D	N	1
Molecular Interaction	PLI	Falk *et al.*[133]		T						✓
		Le Muzic *et al.* [134]		T						✓
		Skan̊berg et al. [19]		T						✓
		ZigCell3D [135]	T	T					✓	✓
		Khazanov and Carlson [136]	S							✓
		Hermosilla *et al.* [137]	T	T						✓
		Vzquez *et al* [138]	T	T						✓
		Chapter 4 & 5		T					✓	✓
	PPI	Finn *et al.* [139]			S	S				✓
		PocketQuery [140]				S				✓
		Furmanova *et al.* [141]			S	S				✓
Clustering		Le Muzic *et al.* [142]						T		✓
		Chavent *et al.* [143]					T	T	✓	
		Chapter 6	T	T	T	T	T	T	✓	✓

Table 2.2: A classification of related work based on features. (T) visualizing time-dependent simulation. (S) visualizing static data. (N) indicates a focus on a complex molecular system. (1) indicates a focus on a single molecule.

questions, challenges, and tasks.

I present the selected articles based on, 1) the visualization of two molecular classes of interactions (PLI and PPI) and 2) visualization of protein clustering. Table 2.2 classifies the related work based on the primary focus of each paper.

2.3.5.1 Visualization of Molecular Interactions

Molecular interactions play a fundamental role in many biological processes. In complex systems, molecular interactions often occur either between lipid and protein molecules or between protein molecules. The size and scale of the interacting molecules can be used to differentiate between the two types of interaction. The lipid is the smallest substance involved in an interaction (usually a nanomolecule), while the protein is a macromolecule component. In the molecular interaction visualization literature, the focus is given primarily to direct protein-surface visualization and internal protein visualization such as cavities. These solutions enable the user to visualize molecular interactions on the surface of proteins and to render tunnels and channels on-the-fly. However, depending on the purpose of the visual design and the visualization theme, details of the interaction may be omitted or absent. For example, encoding interaction density onto a protein surface in 3D indicates accessibility through that surface. However, detail concerning the accessible patches on the surface, e.g. the associated particles

and the interaction frequency for each particle, is not usually presented. In such a case, if underlying detail concerning molecular interactions is desired, the user may need to utilize a separate visualization tool. Conversely, if the user finds interesting results from a 2D visualization he/she might be unable to investigate them in 3D using the same tool.
Here I classify the work on visualizing molecular interactions into 1) PLI visualization, and 2) PPI visualization. In each sub-section, the related work is, also, classified based on the main visualization theme (spatial 3D visualization or information visualization).

Protein-lipid Interaction (PLI) Visualization The following papers focus mainly on 3D and 4D visualization. Two approaches have been used to highlight PLI in 3D, 1) glyphs, and 2) color-coding interaction on the surface of individual particles (a sphere) or molecular surfaces. Falk *et al.* [133] develop a visualization to show molecules of interest as well as their trajectory and interactions. Their method enables the user to zoom in and observe protein interaction by following the trajectory of the protein in 3D. The interactions along the trajectory of a protein are highlighted by glyphs. Le Muzic *et al.* [134] design a framework to convey the spatial aspects of molecular interaction. The user can focus on a single molecule such that interactions between the given molecule and other are brought into focus in front of the user in a 3D view. The interacting protein is color mapped to a user-selected attribute. Skånberg et al. [19] visualize pairwise interaction strengths between molecules utilizing mutual interreflections as a new visual communication channel. The interaction strength is encoded on the surface of atoms in 3D view. ZigCell3D [135] provides 3D visualization of molecular systems while the interaction between particles is visually highlighted. The visualization includes an interaction network by which the user can interactively select a molecule of interest in the 3D visualization. The network stores the interaction state and represents the relationship between molecules. Chapter 4 focus on the PLI and PPI interaction space. It proposes a cylindrical geometry surrounding each protein to capture PLI and PPI. The interaction frequency is stored in the tiles of the cylinder. The proposed solutions lack details of the interactions. With the exception of ZigCell3D [135] and Chapter 4, no overview concerning the system is provided.

In the field of information visualization, Khazanov and Carlson [136] visualize the contacts with ligands and their binding sites by modifying the atoms' Van der Waals radii and color. They provide several visualization elements in the form of tables and different types of 2D graphs. Hermosilla *et al.* [137] present a visualization approach addressing protein interaction by considering the interaction forces. The visualization utilizes two linked-views, the first view conveys the 3D structure of the protein and the ligand, while the other depicts the spatial arrangement of the atoms. Vzquez *et al* [138] present a compact 2D visualization of molecular simulations. They utilize 2D information visualization tools with coordinated views to present physical properties of single molecular components and their pairwise interactions. The ligand is shown at the center of the display, and the user can hover the mouse over the secondary structure or the residue to obtain the details.

In summary, these researches provide great detail concerning molecular interaction, however, most of them are limited to the interaction of either a single molecule or a single timestep. My work differs from the others in two aspects. First, my framework enables the user to explore the entire time-dependent simulation of a complex membrane consisting of 256 proteins and more than 18K lipids. Second, it is capable of visualizing two types of interaction, PLI, and PPI. In addition, it provides historical information about the PPI at the molecular level, i.e. clustering. This is indicated in Table 2.2.

Protein-Protein Interaction (PPI) Visualization Finn *et al.* [139] propose three visual designs to investigate PPI, 1) a visualization of a PPI network, 2) a 3D representation of the protein, and 3) a schematic diagram that links to the images of residue-level interactions. PocketQuery [140] is a graphical interface which is designed to investigate the properties of PPI. A receptor protein is displayed by its surface, and a ligand molecule is represented by a stick. The receptor surface is color mapped by the partial charge of the residues while the ligand can be color mapped to different properties such as energy estimates. The above tools are limited to individual molecules and do not support dynamic simulations. Most recently Furmanova *et al.* [141] propose a novel set of visual designs focussing on PPI configuration space. They utilize different interactive techniques summarized in a PPI matrix view to enable the user to visually filter configurations that contain a desired combination of interacting amino acids. A 3D representation of the filtering is depicted in a 3D view. To address overlap, occlusion, and visual clutter they enlarge the distance between the interacting proteins. The tool is limited to exploring the PPI configuration space of two proteins.

The visualization of time-dependent PPI has received less attention than PLI which suggests that the study of PPI is more of an open research area. In general, the previous work is dedicated to studying a particular class of interaction either PLI or PPI. With the exception of ZigCell3D [135] and Chapter 4, the previous work is generally not capable of visualizing interactions of time-dependent (MD) interaction for the entire lifespan of molecular systems.

My tool enables the user to visualize PLI and PPI simultaneously for the time-dependent simulation of a membrane including 256 proteins and 19K lipid molecules. The tool provides an overview of the dynamical system, and also the user can focus on a specific protein molecule.

2.3.5.2 Visualization of Protein Clustering

In the molecular visualization literature, the term ’clustering’ is often associated with visualizing the result of various analysis routines which are categorized in the visual analytics field. In this work, I focus on the spatial-temporal clustering of proteins.

The term ’spatial clustering’ appears more often in the imaging and spectroscopic analysis literature [144, 145, 146, 147, 148] while it is seen less often in molecular dynamics visualization. Le Muzic *et al.* [142] propose an illustrative time-lapse method that slows down the movement of proteins while they are involved in a reaction. The tool is designed to address the challenge of tracking molecules in consecutive time steps. This visualization can be used to identify protein clusters with the ability to zoom in and out of time. The limitation of this work is that it cannot provide a comprehensive view of the system clusters. Most recently Chavent *et al.* [143] provide a set of 2D and 3D visualizations employing VMD [25] and matplotlib [149]. The proposed visualizations are used to study and compare large-scale simulation to experimental data of outer membrane protein behavior. However, the visualizations lack interactivity, hence, two separate tools are used.

Most of the previous work focuses on a single aspect of the molecular visualizations, i.e. either scientific or information visualization. The 3D and 4D visualizations alone might be less informative especially when the underlying information about the interactions is desired. The information visualizations are sometimes the best choice to convey details on demand concerning interaction. However, the information visualizations represent abstract data which is useful to link to its 3D source. A combination of 3D visualization, visual abstraction, and information visualization can enable a better understanding of the molecular system, especially if they are interactively linked. Furthermore, uniting these features in one tool accelerates the process of investigating the system. PLI, PPI, and protein clustering are three different phenomena that occur in a molecular dynamics system. However, in general there is still no means by which the three phenomena are integrated together.

2.4 Conclusion

This section provides the reader with two types of review - a wide review, and an in-depth review - covering a set of related works. In summary, with the increase of larger models obtained by MD simulations (Chavent et al. [35], Perilla et al. [34]), it is necessary to define new representations in order to: 1) simplify and clarify the visualization, and 2) limit the number of graphical primitives to accelerate the rendering. With these constraints in mind, I could start my thesis with implementing a set of tools to display the path of molecules and map dynamical properties onto it. This type of rendering is widely used in Physical sciences Lipşa et al. [156] but is not yet adopted in computational biology. In term of molecule surfaces and structures, numerous representations are available for the depiction of MD data from simple molecules to complex macro-molecular systems [53, 45]. Protein molecules, for example, are often represented by surfaces [170, 3, 93] while small molecules can be depicted as licorice, ball-and-stick or Van der Waals spheres [28]. These representations generally take every particle constituting the molecules into account which gives a very accurate picture of the molecular systems. However, in the case of a very big and crowded environment, such as numerous proteins embedded in a large patch of lipid membrane, these same representations might also become inadequate due to computational complexity or by occluding useful information presented to the user. Therefore, I would like to propose a new type of abstraction in order to gain insight into very large model systems constituted by numerous protein and lipid molecules. Finally, these researches provide great detail concerning molecular dynamics interaction. However, most of them are limited to the interaction of either a single molecule or a single time step. Also, most of the previous work focuses on a single aspect of the molecular visualizations, i.e. either scientific or information visualization. The 3D and 4D visualizations alone might be less informative especially when the underlying information about the interactions is desired. This limitation can be addressed by a tool that employs 2D and 3D time-dependent visualizations and enables the user to explore two types of molecular interactions (PLI and PPI) both separately or together simultaneously.

Chapter 3

Interactive Multi-Dimensional Path Filtering of Molecular Dynamics Simulation Data

Contents

"If I have seen further it is by standing on the shoulders of Giants."-Isaac Newton[1]

[1] Isaac Newton (1642-1726) was an English mathematician, astronomer, theologian, author and physicist who is widely recognised as one of the most influential scientists of all time, and a key figure in the scientific revolution.

3.1 Introduction and Motivation

Nowadays, advances in simulations allow the design of large models constituted by million, or billion of atoms Perilla et al [34]. Furthermore, modelling very large membrane systems is becoming a subject of intense research. With current methods, one can perform membrane organelles, virus envelopes, and large patches of bacterial membranes Chavent et al [35]. These new models are indeed especially difficult to analyze due to the diversity of molecules - particularly the lipids - and the huge quantity of data. Thus, computational biologists are now facing challenges to find programs that can help them to visualize and analyze their Molecular Dynamic (MD) data. Molecular simulations have benefitted from the development of a range of visualization tools: from the well established VMD [25] or Pymol [42] viewers to the more recently developed Megamol [2] or Unitymol [27] programs. These programs take advantage of advances in hardware technologies such as GPUs capabilities Chavent et al [28]. Nevertheless, there are still tremendous efforts needed to convey clear information about MD trajectories from atomic details to a more global scale. In this context, the analysis of lipids dynamical behavior in membrane models is a typical example. At the atomic level, it is important to understand how lipids and proteins can finely interact together and how it can influence their respective dynamical properties Hedger and Sansom [150]. Zooming out, at the level of the whole model, aggregates of lipids move together as swarms. At this scale, the dynamical behavior of these aggregates need to be taken into account Chavent et al [151]. My goal is to gain insights into the complex motions of lipids molecules at both scales. Common molecular viewers do no have yet dedicated rendering functions to analyze such lipid paths in a user-friendly way Chavent et al [151]. Thus, I have developed MolPathFinder as a set of tools to help the user interactively extract paths features during the course of the Molecular Dynamics simulation. This program exploits new interactive visualization techniques:

- Focus and context visualization techniques that exploit the GPU to help guide the users attention to specific regions of interest.
- Novel interactive filtering techniques to help the user to identify trajectories based on path length, edge length, curvature and normalized curvature, and their combinations.
- Combination of 2D-3D path rendering in a dual dimension representation in order to highlight differences arising from the 2D projection on a plane.

I applied my approach on a real test case used to better understand the formation of protein clusters at the surface of bacteria Rassam et al [152]. In this work, the lipid dynamics was not extensively studied due to the lack of available tools. In this Chapter, I will show how one can use MolPathFinder to quickly gain insights about the MD data.

The Chapter is organized as follows: In section 2, I give an overview of tools dedicated to the analysis of MD data at different scales. In section 3, I present the available features of MolPathFinder and how they were designed. In section 4, I briefly describe my first results and the user feedbacks.

3.2 Background

In biology, visualizing data is a key step in order to gain insight into mechanisms and functions of cells and organelles. In numerous scientific fields related to biology, it is now possible to extract a vast amount of data. This creates new challenges for scientists to manage and decipher this deluge of information and requires the development of new tools O'Donoghue et al [51]. Structural biology generates more and more complex and large data: from protein structure to nucleic acid assemblies. Thus, structural biologists are using a wide range of programs to make sense of their data O'Donoghue et al [54]. These programs are often dedicated to render static molecules via a wide range of metaphors like balls-and- sticks, ribbons, or even simple lines Goodsell [71]. From these static structures it is then possible to create dynamic models using the so-called Molecular Dynamics methodology Karplus and Petsko [33]. The visualization of trajectories from molecular dynamics simulations has received significant attention over the last few decades Kozlíková et al [153]. To the list of well established and widely used programs - such as PyMol [42], VMD [25], Chimera [41],YASARA [154], [155] - I can add recent developments like MegaMol [2] or UnityMol [27]. All these programs have numerous functionalities to analyze and display the MD data. They are also optimized to harness recent hardware facilities to efficiently render time dependent datasets and not only static structures (Chavent et al. [28], Hirst et al. [60]).

Nevertheless, these programs rarely propose new ways of visualizing the data. They rely on known metaphors such as Surface, Van der Waals, or secondary-structures representations. With the increase of larger models obtained by MD simulations (Chavent et al. [35], Perilla et al. [34]), it is necessary to define new representations in order to 1) simplify and clarify the visualization, and 2) limit the number of graphical primitives to accelerate the rendering.

With these constraints in mind, I designed MolPathFinder as a set of tools to display the path of molecules and map dynamical properties onto it. This type of rendering is widely used in Physical sciences Lipşa et al. [156] but is not yet adopted in computational biology. Recent works has begun to use this approach to display small molecules movements at the surface of proteins Bidmon et al. [12] and nucleic acids Ertl et al. [157] or in the cavities Lindow et al [7]. Here, I am using path lines as a descriptor to render the large number of lipid molecules and how they can dynamically move together.

My multi-dimensional tools focus on MD trajectories by quantifying their attributes such as length, edge length, curvature and normalized curvature. The tools provide the user with a number of different interactive filtering techniques. In addition to the 2D and 3D overview of the molecular dynamics, color mapping is used to enable the user to visualize the trajectories and to identify their distribution. Different filtering techniques are used to render a sub-set of the trajectories based on their length, edge length, curvature and normalized curvature. I provide the user with focus and context techniques to de-emphasize less interesting data. Finally, all the features of the trajectories are computed in both 2D and 3D. A novel visualization method is used to compare the properties of the trajectories in 2D and 3D.

3.3 Visualization of MD Data with MolPathFinder

User Requirements Nowadays, there is a large range of available tools to create and analyze membrane models Javanainen and Martinez-Seara [158]. Therefore, it is now possible to quickly create very large membrane models. Nevertheless, there are rooms to develop new programs to better visualize and interact with such models. One of the main difficulties is to interactively visualize and explore the data. To do so, it is essential to remove unnecessary details while keeping a meaningful rendering. Thanks to my collaboration with computational biologists, I defined the paths of molecules as a good descriptor to tackle this issue. Commonly used programs, such as VMD, can efficiently display very large systems but, to my knowledge, have no build-in function to easily manipulate and decipher molecular paths (see Figure 3.1). MolPathFinder has been developed and designed to fulfil this requirement. A particular attention was paid to offer a variety of features to be mapped onto the path: from the length to the curvature of the path. This visualization helps the user observe the distribution of paths based on a chosen attribute and provides the user with a clear overview of the local properties of the trajectories. However, color mapping alone does not always scale well with data size. I also used filtering techniques to help the user highlight areas of interests and only focus on relevant features. The filtering techniques will be described in section 3.3.1.2 but first I am describing basic functions required to load data and select different parts of the molecule

Loading the data Molecular dynamics simulations produce different formats. Each format is designed and proposed for a specific purpose. For example, the GROMACS [32] software provides coordinate, trajectory, energy and topology data. My data contains the trajectories of 242,790 atoms over a time span of 51 nanosecond (ns).

GRO File The associated GRO file is utilized to get the atoms details i.e. residue name and atom name and to construct the data hierarchy. Only lipid molecules were present in the GRO file provided by my collaborator. There were 2 types of lipids in this file called POPG and POPE. Each lipid molecule is then constituted by several atoms, for example the POPE residue

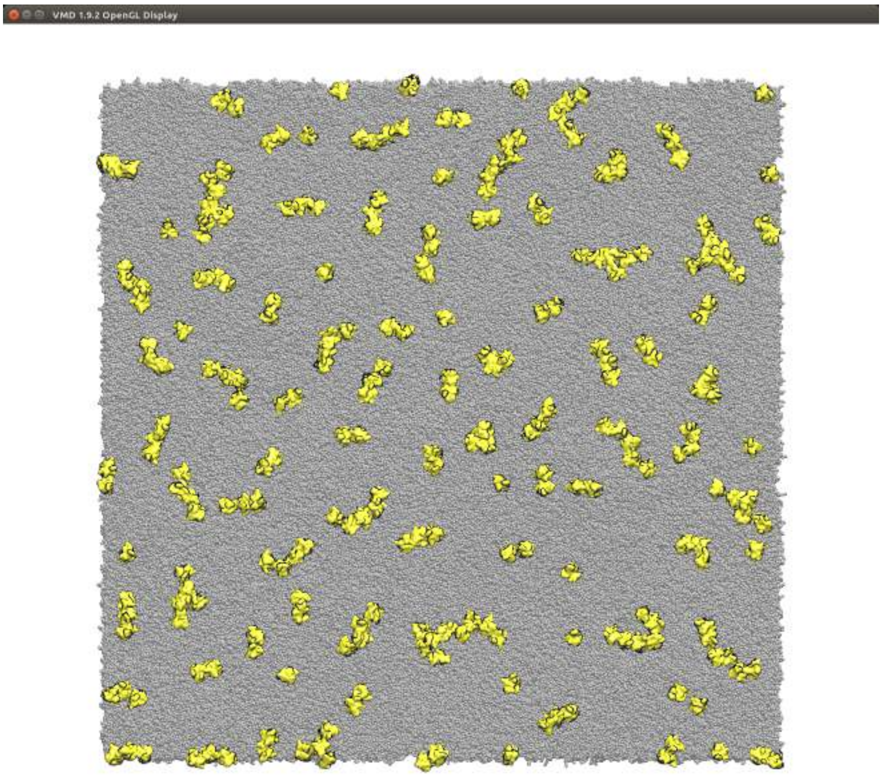


Figure 3.1: Snapshot of the VMD program. The protein data is represented by yellow surfaces and the lipids by gray spheres.

includes 13 atom types C1A, C1B, C2A, C2B, C3A, C4A, C4B, C5B, D3B, GL1, GL2, NH3 and PO4. The hierarchy is used to build the data exploration window (see Molecules explorer and Figure 3.2).

XTC File In order to read the trajectory file, I utilize the GROMACS xdrLibrary which is designed to enable developers to read and write .xtc, .edr and .trr files [159]. The computer simulations define a set of boundary conditions to approximate a large system by using a set of small cells (each cell is called a unit cell). If an atom crosses one side of the unit cell, it enters the opposite side with the same velocity. I process the periodic boundary conditions (PBCs) to introduce spatial domain boundaries and the original coordinate system of the atoms is converted into euclidean space coordinates.

Molecules Explorer I design an interface to facilitate the process of atom-type selection. See Figure 3.2. The data hierarchy is used to present a GUI control that enables the user to select one or more atoms from different residues. Each atom can be colored by a user-chosen color map. One of the advantages of this tool is that it helps the user comparing the behaviors of two or more atoms from different amino acids.

3.3.1 Visualization Techniques and Data Representation

In this subsection I introduce three visualization techniques and briefly discusses the methods that I use to derive the atom trajectory attributes. The atom and path representations, and the color mapping are first described. Then, the focus and context technique in Section 3.3.1.1. The filtering techniques are described in the sections 3.3.1.2 to 3.3.1.7, and the dual space representation representation in section 3.3.1.8.

Atom and Path Representations The user is able to represent the trajectories in three ways. 1) depicting atoms only, 2) depicting paths only and 3) depicting atoms and their paths simultaneously. I provide the user with both static and dynamic representations of the trajectories. Atoms are represented by spheres, and curves are used to represent the atom path. See Figure 3.3.

Color Mapping I utilize color mapping to map local length, edge length, curvature and normalized curvature to a variety of color scales. The nature of my data can exploit a diverging

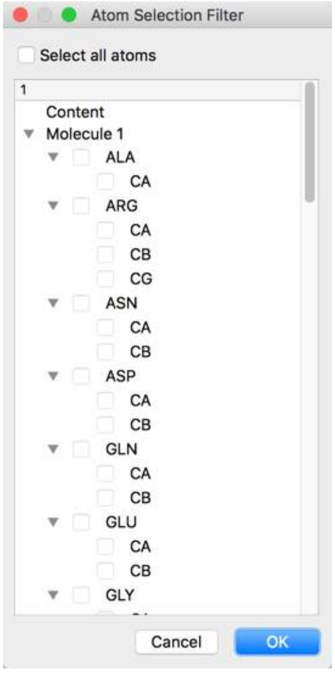


Figure 3.2: Atom selection filter: the molecule hierarchy is utilized to construct the control. The user is able to select different atoms from different residues (amino acids).

color scheme to easily distinquish between the different values. In addition to the sequential color scheme and qualitative color schemes, ColorBrewer [160] provides nine different diverging color schemes. The user can change two coloring options, 1) the scheme control and 2) the class number control. The former is to switch between the color schemes and the latter is to select the number of colors. The color scheme is interactively mapped and applied to the trajectories based on a user-chosen property.

3.3.1.1 Focus + Context

My collaborators provided a very large dataset. Sometimes, the user wants to direct his attention only on a small area of interest. To let him select this type of area I used a Focus + Context technique. I utilize this technique in such a way that it combines with the interactive filtering techniques. The focus data is rendered in its original color whereas a transparent gray is used to render the context. See Figure 3.4.

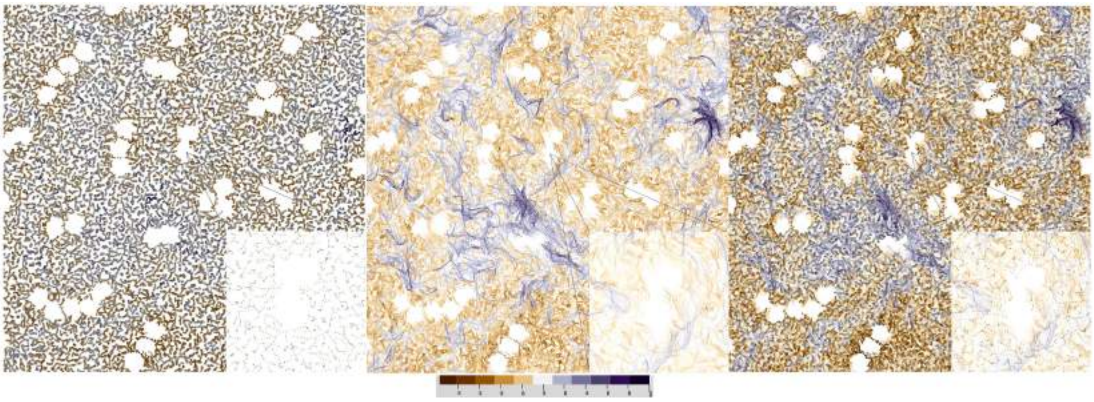


Figure 3.3: Atoms and their path representations. Atoms represented as spheres at time step 0 (left), the atoms' path represented by lines (middle), and the representation of atoms' path and the atoms at time step 51 (right). Color is mapped to total path curvature in 3D.

3.3.1.2 Path Filtering Techniques

Atom trajectories have a number of characteristic properties such as edge length, total length and curvature. Each property conveys different information and requires specific computation. Color mapping is used to represent the local magnitude of these properties. The filtering techniques examine the paths to de-emphasize trajectories that do not match the user selection.

Even though the Focus + Context technique is a very useful approach, it may require the user to select a region of space. However, the different path attributes such as edge length, total length and curvature require special filtering. The user must be able to specify these properties to reduce the rendered data. Based on one or more of the chosen attributes, the path filtering techniques discard paths that do not fulfil the user criteria. The user is provided with range slider controls to specify minimum and maximum ranges. The trajectories are filtered interactively. The filtering techniques work in conjunction with both the color mapping and the focus + context visualization.

3.3.1.3 Filtering Based on Local Edge Length of Paths

In time-dependent trajectory data, the length of path edges reflects the atoms velocity throughout the trajectory. Understanding how the molecules are moving together can be very important to better decipher their aggregation. It is especially true for lipid molecules Apajalahti et al [161]. Filtering the paths in function of their length will clearly highlight groups of molecules

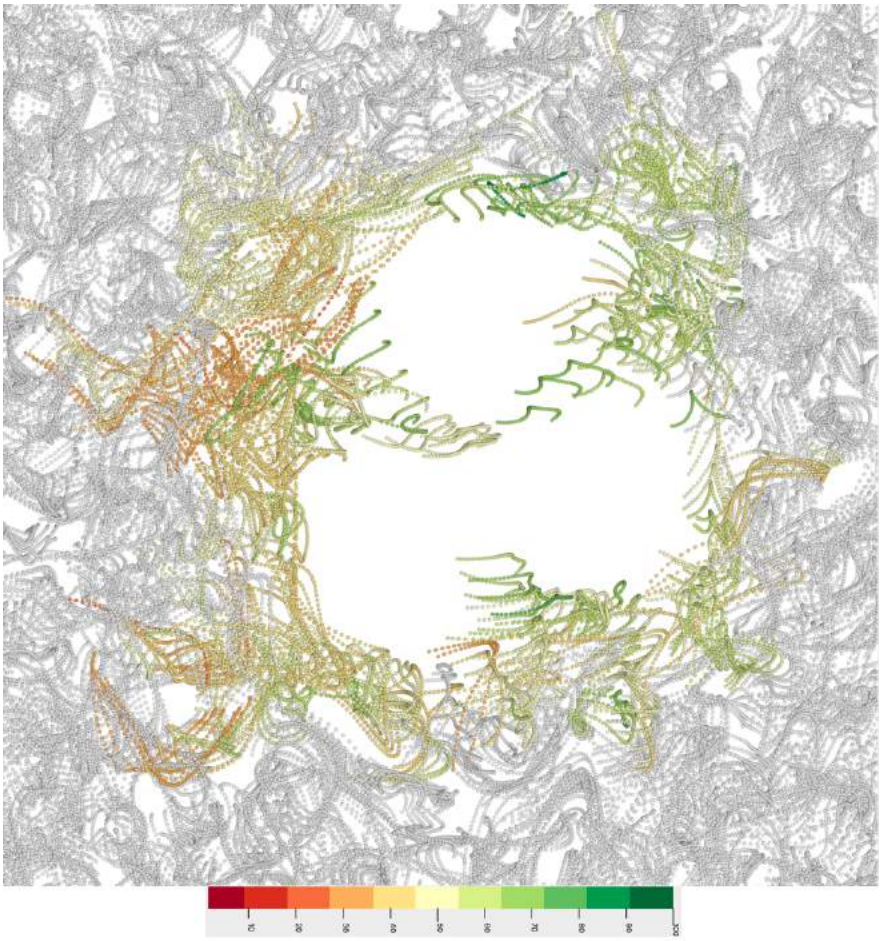


Figure 3.4: A combination of focus + context and color mapping. The focus shows paths surrounding a protein. The total path curvature is coded by color in 3D. The context is de-emphasized by a transparent gray color.

with equivalent dynamical properties hence allowing the user to visually delimit groups of molecules. My trajectory data involves 51 time steps for each atom. Each pair of points in a trajectory constructs an edge. For 2D edges and 3D edges, the edge length is simply computed using the distance between successive points, p_i, p_{i+1}, for each point $p \in \mathbb{R}^3$ in the x, y, z spatial domain. Let T be a trajectory and let p_i and p_{i+1} be two sequential points on trajectory T. The edge $\overline{e}$ is constructed from p_i and p_{i+1}. See Figure 3.5.

$$\overline{e} = p_{i+1} - p_i$$

I find the magnitude $|\overline{e}|$ of edge $\overline{e}$ using:

$$|\bar{e}| = \sqrt{(p_{i+1} - p_i)^2}$$

I store the result of this computation for the local path length. The result for each atom path is normalized by the maximum edge length $|\bar{e}|_{max}$ and stored in the GPU's vertex buffer for color mapping. The current published approaches for visualizing the edge of an atom path fall into two catagoriez 1) a glyph-based approache (using arrows) and 2) a connected edges approach. Chavent et al. [151] combine the two representations into a single depiction. They represent a path using cylinders. The cylinders' position is mapped to time.

I propose a similar approach to Chavent et al. [151] by depicting the path as connected edges combined with varying atom size. This approach helps in both the overview representation and lateral displacement representation. See Figure 3.6.

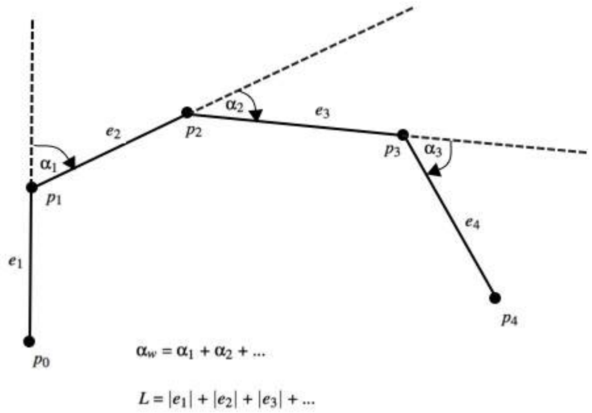

e_2
p_2
α_2
e_3
p_3
α_3
α_1
p_1
e_4
e_1
p_4
p_0
$\alpha_w = \alpha_1 + \alpha_2 + \ldots$
$L = |e_1| + |e_2| + |e_3| + \ldots$

Figure 3.5: The winding-angle α_w is the sum of the angles between the edges e_i and e_{i+1}. The path length L is the sum of the magnitudes $|\bar{e}|$ of the path's edges. Image based on Sadarjoen and Post [24].

3.3.1.4 Filtering Based on Total Path Length

As the local edge length filtering, the total path filtering will help the user to quickly define groups of molecules with the same dynamical features. The edge length of all paths is already computed (see the previous subsection 3.3.1.3). The edge length of each path is accumulated to obtain the total path length. For path P that has n edges $\overline{e}$, the total path length is defined as:

$$L = \sum_{i=1}^{n} |\overline{e}_i|$$

Where $|e|$ is the length of edge $\overline{e}$ and n is the length of a trajectory. See Figure 3.5. The result is normalized by the maximum path length L_{max} and stored in the GPU's vertex buffer. The vertex buffer is used for both color coding and the path filtering techniques.

3.3.1.5 Filtering Based on Total Path Curvature

Membrane curvature plays a key role in several biological functions and can influence the dynamic of proteins embedded in it Quemeneur et al [162]. Computational biologists are creating models to better understand the membrane curvature and how it can be induced by proteins West et al [163]. There are only few tools available to analyze curvature in membrane models Gapsys et al [164]. None of them allow the visualizing of the curvature for each lipid path. Curvature has multiple definitions the i) amount of deviation of an arc from a tangent line; ii) rate of change of the tangent direction; iii) reciprocal of the radius of the osculating circle; iv) an element of area of circular image/element of arclength Goldman [165]. I compute the total path curvature by utilizing the winding-angle method presented by Sadarjoen and Post [24]. However, I use positive rotation for both a counterclockwise and a clockwise-rotating curve. For path P that has $n-1$ angles α the total path curvature of P is derived:

$$\alpha_w = \sum_{i=0}^{n} |\alpha_i|$$

Where $|\alpha|$ is the absolute value of α and α is the angle formed by two edges e. See Figure 3.5. Then the result is normalized by the maximum path curvature $\alpha_{w_{max}}$ before it is stored in the GPU's vertex buffer. The curvature of the trajectories can be represented by mapping the user selected color map to the curvature value. However, the user is able to set the filtering option to reduce the focus data via the range slider control.

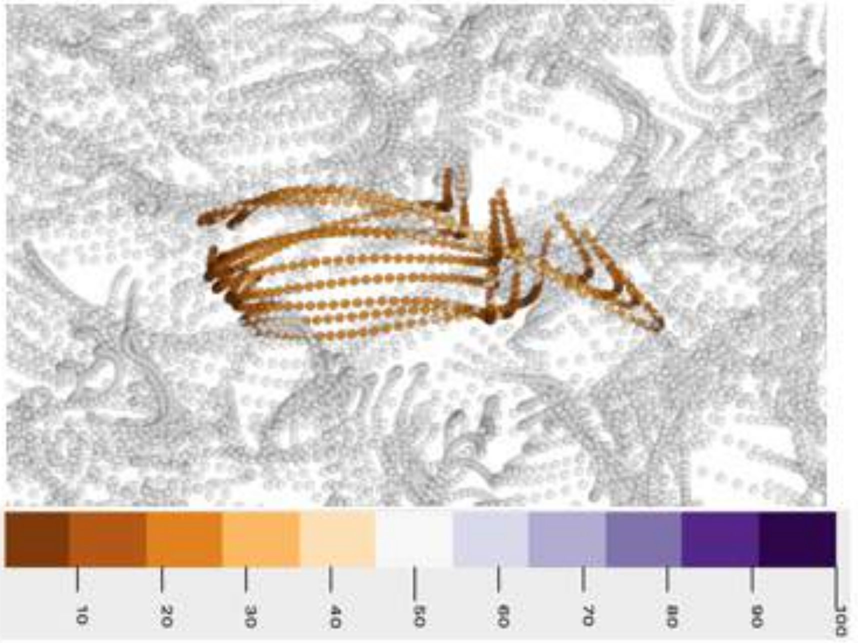


Figure 3.6: The color of atom reflects local edge length in 3D.

3.3.1.6 Filtering Based on Total Normalized Path Curvature

Another approach for investigating the trajectories based on their curvature is by computing the total normalized path curvature. I use the results in section 3.3.1.4 and section 3.3.1.5 to get the total normalized path curvature. For each path P the total normalized path curvature α_N is derived by dividing the path total curvature α_w by the total path length L:

$$\alpha_N = \frac{\alpha_w}{L}$$

The result is stored in the GPU's vertex buffer to be used for both filtering and color mapping.

3.3.1.7 Filtering Based on Depth of Path Starting Point

In order to derive helpful observations based on molecule dynamics in 3D space, I provide the user with a depth filtering technique based on the z component of the trajectories starting point p_0. The user is able to de-emphasize trajectories that are out of depth of interest by defining a minimum and maximum value of z . The depth filtering works alongside the other filtering and visualization techniques. See Figure 3.7. Membranes are deformable systems and can accommodate very curvy shapes. This filtering is useful to highlight areas of equivalent z value and then focus user attention on group of molecules gathered in the bottom or at the top of the membrane shape.

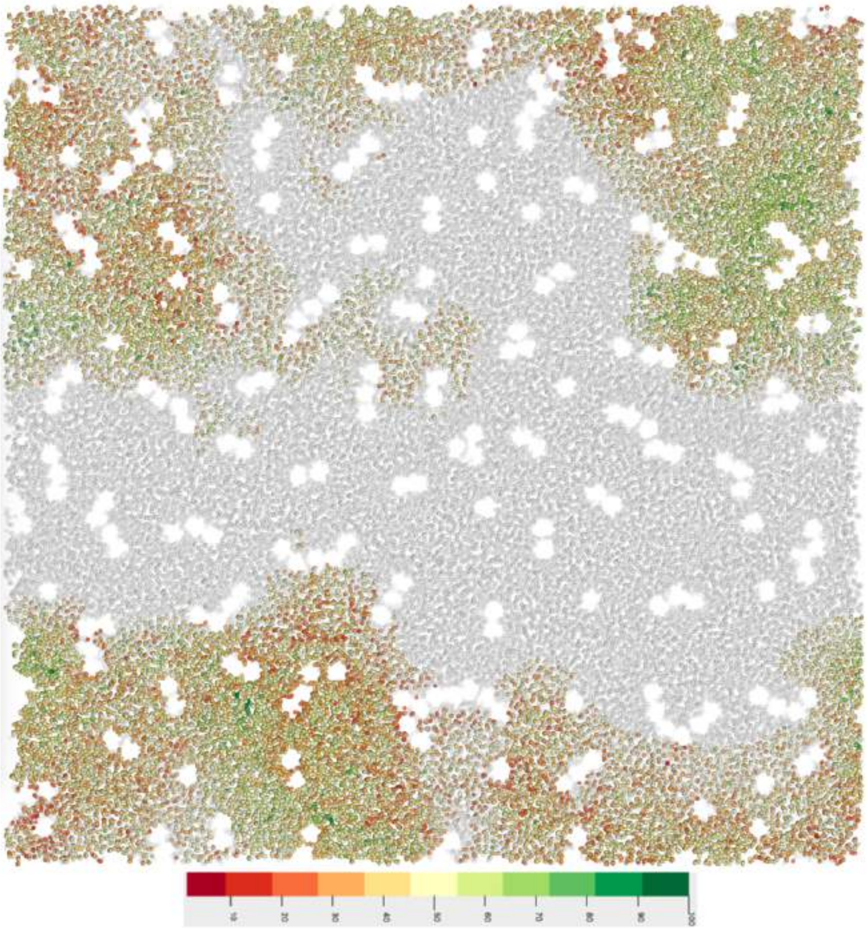


Figure 3.7: The z Depth filtering technique (min z =6.3 nano and max z =7.5 nano). Color is mapped to the total path length of the atom. The gray color indicates atoms outside the user-defined range.

3.3.1.8 Dual Space Representation

Projecting membrane properties onto a plane is a common simplification made by computational biologists to analyze their models (West et al. [163], Falck et al. [166], Lin et al. [167]).

Nevertheless, the membranes are 3D objects which are deformable. Assessing the differences between 2D and 3D rendering may help users choosing the most accurate depiction for the subsequent analyses.

The MolPathFinder provides the user with two spatial representations in order to study the data in 2D and 3D space. I explore a novel interactive 2D and 3D comparison visualization. The user can switch between the 2D and the 3D views. However, the novelty of my dual representation is that the user is able to compare between the properties of the trajectories in 2D and 3D simultaneously (see the supplementary video).

From the overview representation, the user gains insight into the global molecule dynamics based on the magnitude of the attributes of paths in 2D and 3D. See Figure 3.8. The user is able investigate interesting paths and visually compare them. First the trajectories are rendered in the 3D space. Next, by updating the same frame buffer, the trajectories are projected onto a 2D plane. For each point in the 2D path I project the z component onto the 2D plane. The path is projected onto the xy plane in such a way it has the same starting point as its dual in the 3D space. The representation shows the trajectories in the 3D and 2D simultaneously. The projected paths can be rendered on a single sliding plane or on a local plane.

3.4 Experimental Results

My experiments are carried out through analyzing and visualizing a real, time-dependent MD data set. The size of the trajectory file is 76 MB. The trajectories involve the evaluation of 242,790 atoms over 51 ns. The dataset provided by my collaborators was creating to extend previous works to model bacterial membrane (Rassam et al. [152], Goose and Sansom [168]). To do so, my collaborators have designed very large models to address the questions of protein clusters at the mesoscale Chavent et al [35]. The main goal of the dataset presented in this Chapter is to understand the influence of such clusters on the lipid dynamics. The software is tested on Mac OS X El Capitan with a 4 GHz Intel Core i7 processor, 16 GB 1600 MHz DDR3 memory, and an AMD Radeon R9 M295X 4096 MB graphics card.

The Open Graphics Library 4.1 (OpenGL) and GL Shading Language 410 (GLSL) are utilized for computing and rendering the trajectory data. I use two shader programs (compute and render) to separate the computation from the rendering process. The former is used to update the color buffers based on the user requirements and the computational result. The second is used to render the data exploiting the updated color buffers. The frame rate is calculated

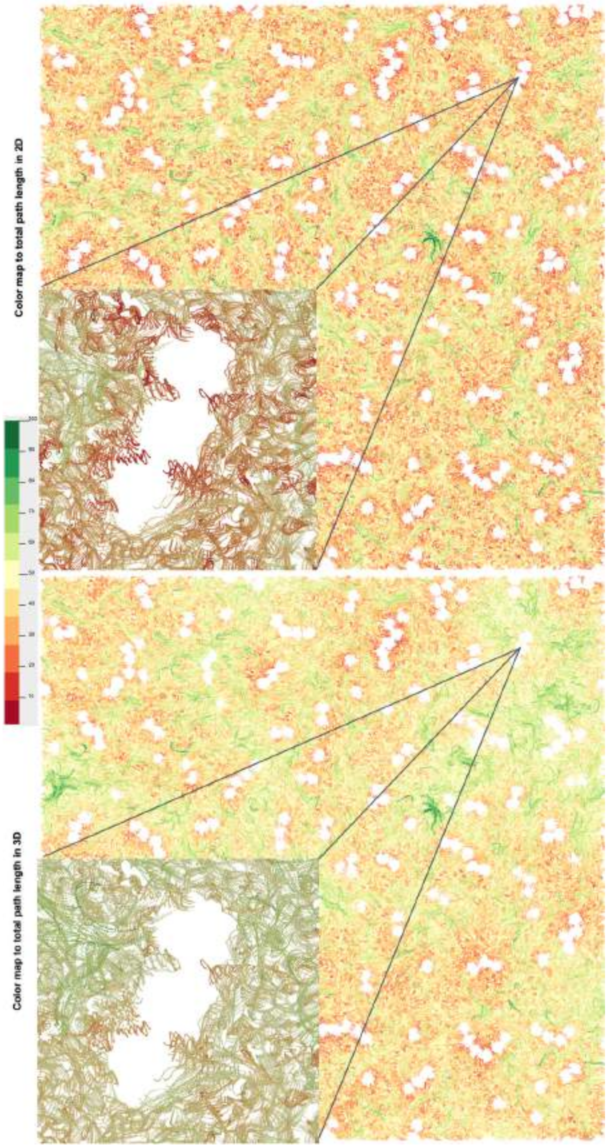

Color map to total path length in 2D
Color map to total path length in 3D
100
90
80
70
60
50
40
30
20
10

Figure 3.8: An overview comparison between total path length on 2D and 3D. Color is mapped to total length in 2D (top) and to total length in 3D (bottom). The comparison reveals a disadvantage of relying on a 2D representation. The total path length of atoms around proteins seems shorter in 2D view while it is not in 3D.

while rendering the entire dataset. The number of rendered frames per second varies based on the selected representation approach. The option of rendering the atom as spheres results in 7 frames per second (FPS) and the path rendering option results in 10 FPS whereas the combination approach i.e. rendering atoms and their paths synchronously results in 5 FPS.

Observations The following observations were made as a result of utilizing the novel dual representation approach. By mapping the length of paths to color in 2D and 3D spaces. Figure 3.8 illustrates that the 2D representation conveys different information about the length of paths around proteins. The length of the paths around proteins in the 2D view seems to be short even though a lipids trajectory takes longer journey around the proteins in the 3D space. Therefore, 3D view provides us with good understanding of the behavior of the data.

With my approach, it appears very clearly that the dynamic of lipids is more important in the area devoid of protein (see Figure 3.3). Conversely, the path length are relatively shorter in the vicinity of proteins showing that the protein aggregates may hamper lipids dynamics.

Domain Expert Feedback This work has been developed in close collaboration with Dr. Chavent Matthieu from the Department of Biochemistry in Oxford university since the start of this project in January 2015. MolPathFinder is my first work and it is designed to fulfil the initial requirements of my collaborator. The tool video demo is available online (https://github.com/NaifAlhar6i/Chapter3). The feedback shows positive reactions to my visualization software. The following is feedback from my domain expert collaboration:

”First of all, the software is very useful as, currently, no other program allows users to decipher the complex movements of lipids molecules from MD simulations. This is a very hot topic as it is now possible, using MD simulations, to model larger and larger systems (Ingólfsson et al. [169], Perilla et al. [34]) but the analysis and visualization of these systems is now becoming a bottleneck.

The visualization is really the main breakthrough of this program where it can help the user to understand in a glance the main movements of lipids molecules both at the nano-scale (i.e. zoom in: molecular level to see the movement of each lipid) and at the meso-scale (zoom out: global movement of lipids such as currents). So far only a recent work has begun to develop this type of rendering Chavent et al. [151] and there is a completely new field of research to develop around this thematic.

The GUI is well defined so that even a non-expert can easily interact with the data very instinctively. There are not too many windows and the labels are self-explanatory so that it is easy to quickly access and analyse the ones you want. There are several filtering functions that help the user to quickly focus on a specific area of interest.

Finally, the filtering in function of lipid path length, curvature, vorticity, etc.. is clearly new and a very valuable addition to this program. It was previously impossible to grasp this type of information in such an easy way."

3.5 Conclusion

Visualizing time-dependent trajectory data has received great attention over the past decades. A variety of visualization tools are used to investigate the MD data at different resolutions. This Chapter focuses on visualizing time-dependent trajectory MD data at the atomic level. It provides the user with 2D and 3D representations and a number of different visualization techniques. The user is able to render the entire set of trajectories or a sub-set of the data. A variety of filtering techniques is designed to reduce the focus of the rendered data based on the user chosen-trajectory attributes. The dual representation helps to compare trajectories in 2D and 3D. Focus + context and zooming are used to explore the local behavior of the molecule dynamics. The implementation, in this Chapter, relies on OpenGL to perform the computation task. However, other frameworks such as OpenCL and CUDA are designed for these purposes and they are a good candidates for future work.

While Chapter 3 is designed to study MD trajectories in general, in Chapter 4 the focus will be on a specific region of the MD trajectories. Namely, the area surrounding protein molecules at which the most molecule interaction take place. Chapter 4 introduces a new abstract interaction space to address computational and visualization challenges arising from lipid-protein dynamics.

Chapter 4

Novel Visual Abstraction for Protein-Lipid Interactions

Contents

> *"The greatest challenge to any thinker is stating the problem in a way that will allow a solution"-Bertrand Russell*[1]

This Chapter presents a novel solution for visualizing protein-lipid interaction in a membrane MD simulation. The proposed solution employs abstraction and simplifies the computa-

[1] Bertrand Russell (1872-1970) was a British philosopher, logician, mathematician, historian, writer, essayist, social critic, political activist, and Nobel laureate.

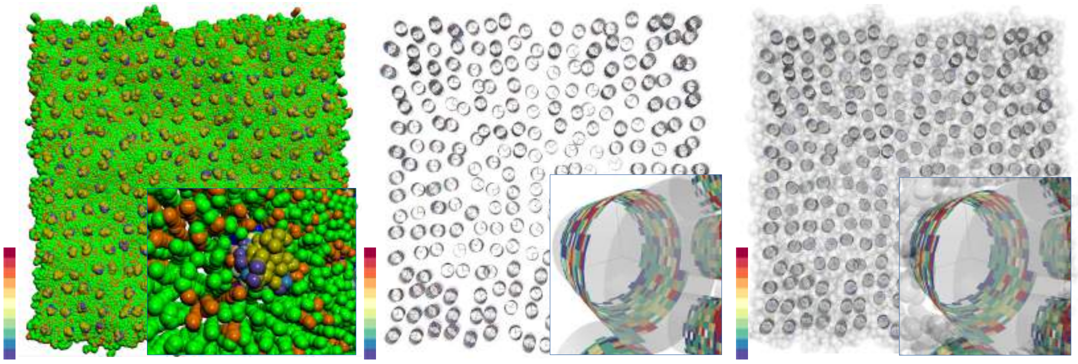


Figure 4.1: Visual Abstraction for Protein-Lipid Interaction (VAPLI). (Left) A naive visualization of protein-lipid interaction (PLI). (right) Focus-and-Context applied to the PLI. My PLI depiction is rendered with lipid particles, using transparent Van der Waals representation, to illustrate the membrane context. (middle) The PLI space is depicted using only my VAPLI approach, focusing on proteins greatly reducing both complexity and occlusion. Color is mapped to the frequency of PLI on a tiled cylinder.

tion and visualization of protein-lipid interaction. The GPU implementation of this Chapter is based on Chapter 7.

4.1 Introduction and Motivation

Visualization of Molecular Dynamics (MD) simulations has received a great deal of attention over the past decades. Numerous representations are available for the depiction of MD data from simple molecules to complex macromolecular systems [53, 45]. Protein molecules, for example, are often represented by surfaces [170, 3, 93] while small molecules can be depicted as licorice, ball-and-stick or Van der Waals spheres [28]. These representations generally take every particle constituting the molecules into account which gives a very accurate picture of the molecular systems. However, in the case of a very big and crowded environment, such as numerous proteins embedded in a large patch of lipid membrane, these same representations might also become inadequate due to computational complexity or by occluding useful information presented to the user. Here, I would like to propose a new type of abstraction in order to gain insight into very large model systems constituted by numerous protein and lipid molecules.

Membrane proteins perform a wide range of biological functions such as drug transporters, ion channels, or receptors transmitting molecules and information across the cell membrane. These proteins are often dynamically modulated by their environment, i.e., the surrounding membrane. This modulation can be mediated through global effects such as the membrane curvature [171] or rely on more subtle changes like close interactions in between lipids and proteins [172]. MD simulation is now a powerful tool to assess such interactions [150]. However, it can be difficult to fully understand the molecular mechanisms of such interactions especially for large systems. Furthermore, these interactions are often transitory and quickly change in a time-dependent manner. For example, the dynamics and complexity of membrane models cause many particles to be occluded or to overlap (see left image in Figure 4.1). This is creating a number of computation and visualization challenges which may not be overcome by classical molecular depictions.

My solution is to design a new type of molecular abstraction focusing on the Protein-Lipid Interactions (PLI). This representation enables the depiction of numerous membrane proteins while retaining useful information held by the PLI. I design and implement this Visual Abstraction for PLI (VAPLI) in close collaboration with domain experts to help computational biologists obtain insight into their membrane time-dependent systems. My contributions are:
1) A novel cylindrical definition and abstraction of the PLI for large MD simulations,
2) A fast GPU-based implementation to calculate PLI on the fly
In short, the frequency of PLIs is represented by a tiled cylinder. Color-mapped tiles indicate PLI frequency. The Chapter is organized as follows: in Section 4.2, I discuss works related to molecular visualization and abstraction. In Section 4.3, I describe the data from MD simulations I am using. Section 4.4 discusses the design and implementation of my VAPLI approach. The preliminary results are presented in Section 4.5 following by the domain expert feedback in Section 4.6.

4.2 Background

There are a number of recent and helpful survey papers on molecular visualization. O'Donoghue *et al.* [54] review visualization methods and tools that enable the community of life scientists to gain insight into their molecular structure data. In a recent state-of-the-art report, Kozlíková *et al.* [53] review and classify visualization techniques developed to render molecular structures.

In the field of molecular visualization using abstractions is mainly used to: 1) enhance software performance, or 2) reduce visual complexity, and enhance perception. Here, I classify the abstraction research into two groups based on the proposed approach: abstractions used to increase the computing performance and abstractions to improve user perception.

Abstractions To Accelerate Performance And Reduce Computational Complexity: In this subsection, I discuss previous work that incorporates abstraction and level-of-detail (LoD) techniques to represent large data-sets relying on spatial configuration approaches to accelerate performance. Previous literature often studies spatial configuration approaches that rely on 1) the arrangement of the atoms of a molecule, or 2) manipulating the resolution of the surface of a molecule with respect to the LoD. A common goal is to enable a user to visualize large molecular datasets interactively by reducing the representation of the data based on the desired level of detail.

TexMol [173] introduces a biochemically sensitive level-of-detail hierarchy for molecular representations. A single sphere is used to represent an atom with the atom's Van der Waal radius, whereas a residue is represented by a minimal single bounding sphere that encloses all of its component atoms. This approach reduces the visual clutter by communicating the appropriate volume occupancy and shape. In addition, their hierarchical image-based rendering enables mapping of derived physical properties onto molecular surfaces.

Sharma *et al.* [174] implement a hierarchical view frustum-culling algorithm based on an octree data structure. The algorithm efficiently removes atoms outside of the user's field-of-view.

Lee *et al.* [175] propose a view-dependent LoD technique for real-time surface rendering. It provides high-quality rendering and is able to display critical parts of molecular models. Lampe *et al.* [176] introduce a two-level approach to visualize large, dynamic protein complexes. In the first level, each residue is replaced with a single vertex based on the residue's rigid transformation. In the second level, the atoms which are contained in the residues are dynamically generated in the geometry shader based on the initial simulation vector and the transformation matrix.

Lindow *et al.* [177] introduce a GPU-based, 3D voxel grid to store the atomic data and utilize a fast ray-voxel traversal by which only spheres in the current voxel are tested for intersection. Similar to Lindow *et al.* [177], Falk *et al.* [119] store each molecule in its own supporting grid, which is then traversed with a level-of-detail via a ray. The grid is constructed based on the size of each atom whereas in [177] the grid is based on the number of atoms. Krone *et al.* [178] propose a novel fast molecule surface extraction approach which can be used to depict structural detail on a continuous scale. By manipulating the grid resolution and the density kernel function the scale can range from the atomic detail to reduced detail visual representations.

Le Muzic *et al.* [134] present a LoD approach that summarizes adjacent atoms into a single sphere depending on the distance to the camera. They utilize a texture to store the atom positions of a whole molecule. Their approach is similar to Lampe *et al.* [176] but uses the tessellation shader instead of the geometry shader.

Abstractions To Enhance User Perception: This subsection discusses work providing different visual abstractions of a molecule with respect to its structure to enhanced user perception. In these cases, each representation is generally associated with a specific molecular structure and different LoD techniques are applied to provide a smooth transition between structures.

Weber [179] proposes a texture-based approach using shader programming to generate pen-and-ink renderings of molecules in real-time. Van der Zwan *et al.* [180] propose a novel continuous abstraction space for illustrative molecular visualization. The method is GPU-based for visualizing continuous transitions between different stages of structural abstraction of a molecular system (i.e., space-filling, ball-and-stick, licorice, backbone, and ribbon). Parulek *et al.* [181] propose a novel LoD approach to visualize large protein complexes. The application of individual levels is based on the distance to the camera. They employ three different surface representations (solvent-excluded surface (SES), Gaussian kernels and Van der Waals spheres). Mohammed *et al.* [182] present a novel tool for visualizing astrocytes and neurons at different levels of abstraction. The tool benefits from a novel abstraction space to provide a set of visualizations ranging from a detailed 3D surface renderings of segmented structure (realistic images) to simplified abstractions (e.g. skeletons and graphs).

In this Chapter, I introduce a novel abstraction of the protein-lipid interaction space to address the visual complexity and perception of such a representation and provide a simplified depiction of the PLI. This representation supports detecting PLIs on-the-fly by using GPU-accelerated calculations.

4.3 Simulation Data Description

The dataset I have used represents a trajectory of a large patch of a membrane constituted by lipids and proteins extracted from molecular dynamics a simulation. The simulation is used to study the dynamic behaviour of both proteins and lipids in the context of a membrane aligned primarily in the *xy* plane [151]. The dimensions of the system are $116 \times 116 \times 10$ nanometers

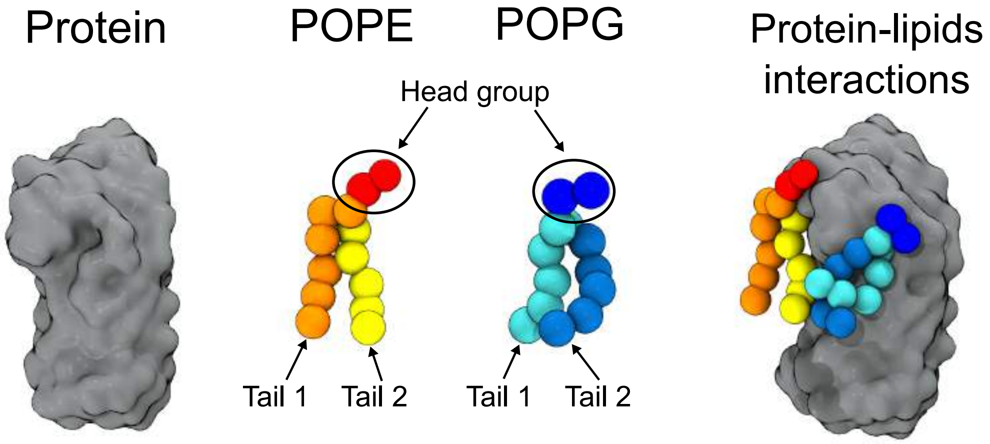


Figure 4.2: Typical representations of molecules: protein (surface) on left, POPG and POPE lipids (Van der Waals) in the middle, and two protein-lipids interactions (on right). Protein and lipids representations were generated with VMD [25].

(*x*, *y*, and *z* respectively) and represent individual trajectories of 336,260 particles over almost 2 microseconds (stored in c.a. 2000 frames). The system consists of three molecule types: one protein, and two lipid types (POPE and POPG) (see Figure 4.2). There are 256 protein molecules while the number of POPE and the POPG lipid molecules are 14,354 and 4,738 respectively. The models presented are based on the MARTINI coarse-grained forcefield [183] which does not represent all the atoms but simplifies 4 heavy atoms by one coarse-grained (CG) particle. Thus the hierarchy for each type of molecule is simplified:

Each protein is constituted by 171 residues and each residue is composed of 1 to 3 particles. The total number of particles per protein is 344 resulting in 88,064 particles in total (256 × 344).

The total number of lipids (POPG and POPE) is 19,092. Each lipid molecule has 13 particles which result in (19,092 × 13) 248,196 particles in total. For this representation, I divide each lipid into 3 groups: a head group of 2 particles, a first tail group of 6 particles, and a second tail group of 5 particles (see Figure 4.2).

In terms of the data size, the simulation contains more than 666 million space-time positions that occupy 8 Gigabytes of memory.

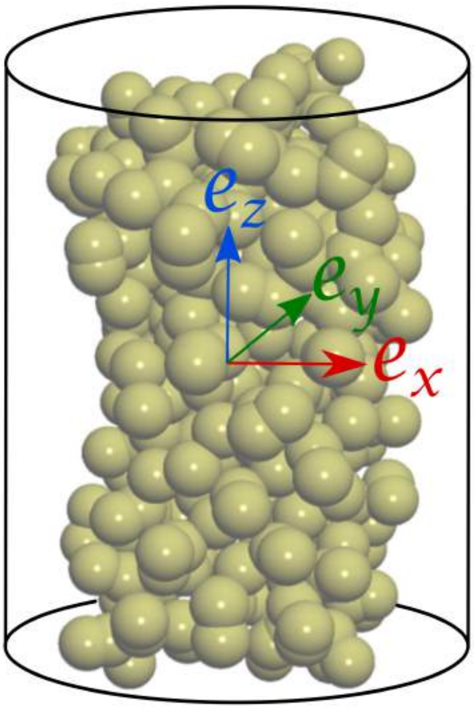


Figure 4.3: Principal components of a protein interaction space at $t = 0$. The SVD method is used to calculate the *eigenvectors* ($\mathbf{e}_x, \mathbf{e}_y$, and $\mathbf{e}_z$) to initialize the local cylinder and its extent from the original protein particles.

4.4 Visualization of Protein-Lipid Interaction (PLI)

In this section, I present why and how I have designed the PLI space as a cylinder surrounding the protein. I present different aspects of the implementation of the PLI abstraction including PLI testing, translation and rotation of the cylinder, and grid tiling.

Design: The interaction space is represented by a tiled cylinder. The main axis of the cylinder, $\mathbf{e}_z$, is aligned with the principal axis of the molecule, it surrounds, and is updated at each time step. The orientation captures the main component of lipid translation that influences PLI. The PLI frequency is reflected in the local tailing. The tiles are color-mapped based on how often a lipid crosses into the approximate molecular space. This is not typical as there are no previous abstractions that are dedicated to interaction space that I am aware of.

Defining the PLI Representation: I chose a cylinder surrounding each protein to represent the

PLI space. This simplified abstraction is a good approximation of membrane protein shape (see Figures 4.2 and 4.3) and features at least two advantages in comparison to a complex molecular surface: i) simplifying the visualization and ii) enhancing the computing performance.

To create the cylinder, I first calculate the Center of Mass (CoM) and the eigenvalues for each protein (see Equations 4.1, 4.2, 4.3). I then position the center of the cylinder onto the protein CoM. By default, the largest eigenvalue is used as the cylinder height while the second eigenvalue defines the cylinder radius (Figure 4.3). The cylinder height and radius can be also specified by the user to interactively manipulate the PLI space size. The cylinder mesh is triangulated on the GPU based on the protein CoM, and values of height and radius. The opacity of the mesh can also be adapted interactively to provide different levels of transparency.

I provide two representations, a static representation and a dynamic representation. The key difference between the two representations is that the static representation omits the x and y-axis rotation.

Updating the PLI Representation: The dynamics of protein particles play a significant role in defining the geometry of a protein at a given time step. Some protein properties (e.g. global position, CoM, principal axes, spatial extent, and rotation) might be influenced by the system dynamics. As a result, the PLI space can be updated for every time step. I use Singular-Value Decomposition (SVD) [184] to calculate the main properties of a protein interaction space i.e., the principal axes (*eigenvectors*), dimension sizes (*eigenvalues*) and rotation matrix. SVD requires the covariance matrix C of a protein space. C is a symmetric real $n \times n$ matrix. There is an orthogonal matrix V and diagonal Σ such that:

$$C = V \Sigma V^T \tag{4.1}$$

The diagonal entries of Σ, that is $\Sigma_{ii} = \sigma_i$, can be arranged to be non-negative in order of decreasing magnitude. Positive entries are called the singular values (*eigenvalues*) of C. The columns of V and V^T are called left and right singular vectors (*eigenvectors*) and form an orthonormal basis for $\boldsymbol{R^n}$ [185]. The eigenvectors represent the axes of a protein interaction space and the associated eigenvalues are used to define a protein interaction space extent. The first eigenvector is used to represent the z axis orthogonal to the primary protein bi-layer, and the second and third eigenvector are used to represent the x axis and y axis - the orientation of the dominant lipid motion. Figure 4.3 shows the eigenvectors of a protein interaction space. However, the order of the second and the third eigenvector is not necessarily constant over time and needs to be consistent. This challenge is addressed by performing two tests, the first test is performed at $t = 0$ to ensure that the x axis and the y axis are in the correct order. The second test is performed at every time step $t > 0$ to maintain consistency of the x and y axes.

PLI Cylinder Translation and Rotation: In the model of membrane studied here, each protein molecule is constituted by a finite number of particles (344 particles). These particles are linked together through constraints defined by the forcefield which limits the overall displacement of one particle towards the others belonging to the same protein. Thus, the overall protein movement can be simplified by the movement of its Center of Mass (CoM). For each time step, the CoM of a protein is calculated using the following:

$$CoM(t) = \frac{\sum\limits_{i=1}^{n} p_i(t)}{n} \tag{4.2}$$

Where $p(t)$ is the position of particle p at a given time step t and n is the number of particles. The trajectory of the CoM of a protein represents the evolution of the protein position over the simulation time. The rotation of a protein is defined by the internal motion of its particles. Two successive positions of all the protein's particles are needed to calculate the covariance matrix C of a protein.

$$C(t) = \sum_{i=1}^{n} p_i(t) \times p_i^T(t+1) \tag{4.3}$$

$p(t)$ is the position of particle p at a given time step t and p^T is the transpose vector of the position of particle p at $t = t+1$. Applying Equation 4.1 to C I get V and V^T. The rotation matrix R can be calculated from V and V^T:

$$R(t) = V(t)V^T(t) \tag{4.4}$$

The translation and the rotation are then applied to the cylinder mesh to update the cylinder position.

PLI Calculation: To calculate PLI, I provide two tests (see Figure 4.4) in order to simplify the overall calculations. First, I approximate the protein shape by the cylindrical region used to depict the PLI space defined previously. If the smallest distance between one lipid particle and the central axis of the protein is less than the cylinder radius I consider that the particle is interacting with the protein (first approximation). If this first test is accepted then the distance between every individual protein particle and the given lipid particle can be calculated for higher accuracy (second approximation). In this case, I consider an interaction between protein and lipid particles if the distance between the two is below a given threshold distance

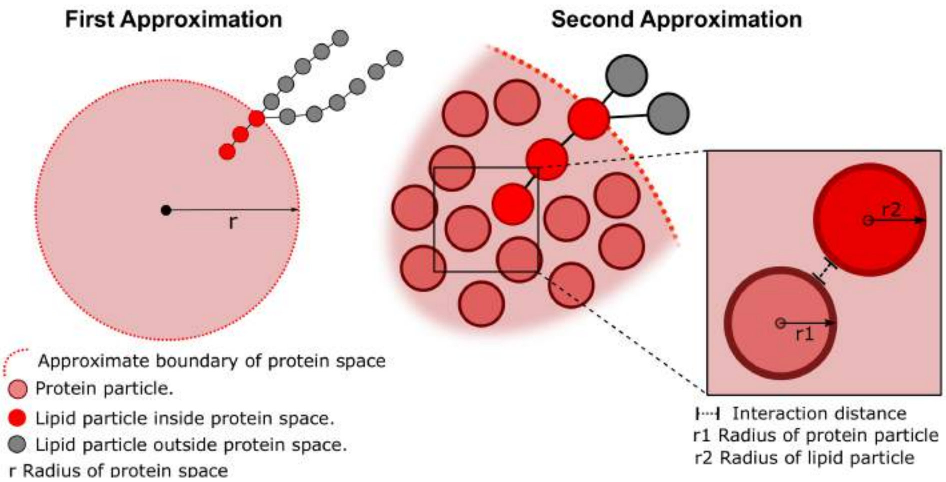

First Approximation
Second Approximation
r
r2
r1
Approximate boundary of protein space
Protein particle.
Lipid particle inside protein space.
Lipid particle outside protein space.
r Radius of protein space
Interaction distance
r1 Radius of protein particle
r2 Radius of lipid particle

Figure 4.4: The left image depicts my first approximation for the PLI test. The test requires only the distance between a lipid particle and the principal axis of a protein. The right image illustrates the second approximation for PLI: for every lipid particle that passes the first test, another distance test between the given lipid particle and the protein particles can be performed.

in the range of 0.6 −0.8 nm. The minimum and maximum values are defined in Marrink et al [186] and Goose et al [168] respectively. So I took the largest value to be sure that I have at least one particle interacting with the protein residues.

Tiling the Interaction: The interaction test result, for each protein, is stored in a uniform 2D grid structure. The grid size is 32×32 by default or can be specified by the user. The x-axis of the 2D grid represents the angle of PLI cylinder while the y-axis represents its height (see Figure 4.5). While the number of grid cells can vary, the grid has a fixed length along the x-axis of 2π and a fixed height h defined previously. Every cell of the 2D grid defines a protein area where the PLI is calculated and updated. In this Chapter, I use a regular 2D grid rather than a quad-tree structure. Nevertheless, I would like to implement such a structure in future work.

Projecting the Interaction onto the Abstract Protein Space: The interaction mapping process is performed on the GPU by using the geometry shader. The advantage of this approach is that it is no longer necessary to store, update, translate, or rotate the interaction positions for every time-step because they are based on the given local protein coordinate system. For the full detail representation, all the non-null cells, i.e cells that contain interaction detail, are projected as a color-mapped tile onto the surface of the respective protein space. Each tile represents the frequency with which a lipid has intersected its extent. The bottom-left corner of the projected patch is defined by the bottom, left edge of the cell. The size of the projected

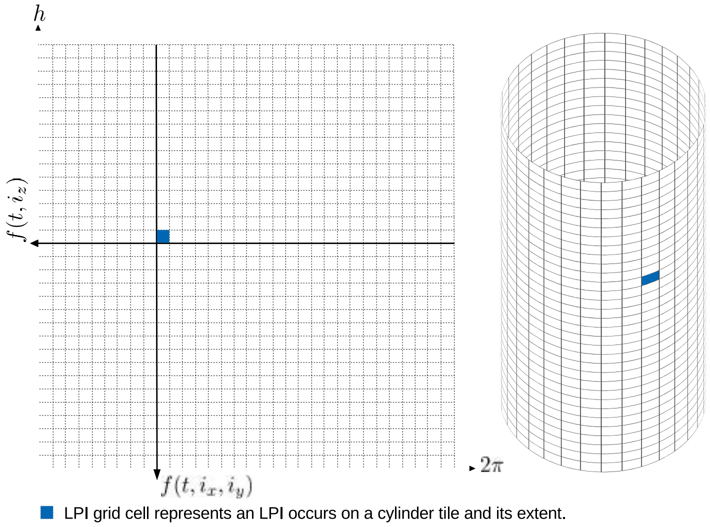

h
f(t, i_z)
f(t, i_x, i_y)
2π
🟦 LPI grid cell represents an LPI occurs on a cylinder tile and its extent.

Figure 4.5: PLI interaction frequency is stored in a uniform grid. The highlighted cell represents a PLI tile at the highest resolution. Tiles are used to depict PLI at different resolutions. The *x*-axis represents the PLI angular value at a given time step t. The *y*-axis represents the z component of PLI at t.

patch is based on the tiling. See Figure 4.5.

4.5 Preliminary Results

The complexity of PLI increases over time which requires special visualization rather than naive approaches. Figure 4.1 shows the visualization of complex simulation data and compares my approach to a naive approach. The left image depicts a crowded environment of lipids and proteins with a naive approach. The middle image visualizes the data based on my abstraction. Focus-and-context is used in the right image. One can easily see that in the naive visualization most of the PLI details are obscured due to lipid occlusion. My approach provides an overview of the PLIs for the whole system and the user can easily zoom in to obtain more details about a particular PLI. With my approach, some patterns emerge: PLI can form a wide-band of interactions on the abstracted surface. The highest frequency of PLI occurs at different locations but are mainly focused in the center of the cylinder while the upper and bottom edges are less populated reflecting the strong interactions between the tails of the lipids and the proteins. A supplementary video showing the results presented in this Chapter can be found at:

https://github.com/NaifAlhar6i/Chapter4

4.6 Domain Expert Feedback

I have been working closely with domain experts in molecular dynamics since 2015. All of the designs presented here are inspired by my discussions with them. Here is direct feedback from one of them. "*This new PLI abstraction allows the user to quickly understand where the lipids can interact with each protein. The main interactions are visible in the hydrophobic parts in the center of the proteins which is expected but was not seen so clearly with more classical representations. In general, it was necessary to perform a costly post-treatment on a part of the trajectory or on the whole trajectory to get the same information. Furthermore, using this PLI abstraction gave a very quick idea of where to focus the attention on interesting proteins which can strongly interact with the lipids.*"

4.7 Conclusion

Numerous tools have been designed to better understand the structures of molecules, from simple molecules to complex macro-molecular systems. Few tools are focusing on time-dependent data sets from Molecular Dynamics simulations, and almost none are focusing on visualizing the molecular interactions between lipids and protein. Chapter 4 introduces a novel abstract Protein-Lipid Interaction space to address computational and visualization challenges arising from lipid-protein dynamics. This representation highlights PLI better than classical naive representations and allows experts to quickly highlight interesting molecular features. Thus, the representation provides great potential to investigate PLI from molecular dynamics systems. In Chapter 5 I plan to design a quad-tree structure to handle the interaction data. I am, also, going to use Level-of-detail (LoD) techniques on lipid molecules to reduce overlap and occlusion.

Chapter 5

Level of Detail for Visualizing Time-Dependent, Protein-Lipid Interaction

Contents

> *"Tell me and I forget, teach me and I may remember, involve me and I learn."-Benjamin Franklin*[1]

This Chapter is built based on the future work described in Chapter 4. The Chapter describes the proposed abstract protein space in the context of membrane protein-lipid interaction, and it also describes six levels of detail for both lipid types and the PLI abstract space.

5.1 Introduction and Motivation

Visualization of Molecular Dynamics (MD) has received a great amount of attention over the past two decades. Numerous MD visualization tools are used for visualizing MD data from simple to complex molecules [[53] and [45]]. With recent advances in computer graphics, most of the MD visualization research focuses on optimal rendering quality and performance.

Proteins perform an array of functions such as drug and neurotransmitter transport. Lipids intimately influence the structure and function of membrane proteins by extensive protein-lipid interaction (PLI) [171]. Here the PLI plays a significant role in understanding the interdependence of membrane proteins with their surrounding lipids. Simulations facilitate structural understanding in the context of a lipid bi-layer. This is the application I focus on here.

In an MD visualization, a number of models are used to depict molecules at the molecular and atomistic scale. Protein molecules, for example, are often represented by surfaces. This representation provides an overview of the protein shape at the molecular scale. At the atomistic scale, small molecules including atoms' bounds are often represented by models such as ball-and-stick. These representations are commonly used to visualize the MD data at the desired scale. However, depending on the intended analysis task, these same representations might also become part of the challenge due to their complexity.

The visualization of PLIs in time-dependent systems involves a number of challenges. I aim to provide a LoD visualization of the PLI over the entire simulation time-span. Complex protein surfaces might not be an ideal solution in this case. The dynamics of the system may cause many particles to be occluded or to overlap. The PLI occurs at the atomistic level which presents computation and visualization challenges. Every single interaction holds information that might explain the lipid's influence on a protein's functions. Special techniques I can use to enable users to investigate the PLI at different levels of abstraction. At the very beginning of the simulation, a lipid might interact with one protein at most. Over time, the complexity of the system increases due to the evolution of the system particles. Figure 5.1 illustrates the

[1] Benjamin Franklin (1705-1790) was an American polymath and one of the Founding Fathers of the United States.

motivation behind this work where lipid particles interact with more than one protein. I design and implement a LoD PLI in close cooperation with domain experts to help computational biology scientists to obtain insight into PLI for time-dependent systems. My contributions are:

- A definition and abstraction of PLI with six levels of detail for the protein-lipid interaction.
- A smooth transition between the six LoDs utilizing lipid-scale and particle-scale effects. The lipid-scale effect provides a smooth transition for particle positions between levels. The particle-scale effect provides a smooth change of the radius of the particles between levels.
- A fast GPU-based implementation to represent PLI LoD in the abstract interaction space.

The Chapter is organized as follows: In Section 5.2, I describe the simulation data. Section 5.3 discusses the visual-design aspects following by the results in Section 5.4.

5.2 Simulation Data Description

Data-sets characterize biological dynamics of lipids and proteins within the perspective of a membrane model. The simulation domain's dimensions are 116 × 116 × 10 nanometers (x, y, and z respectively). Individual trajectories reflect the evolution of 336,260 particles over 1,980 nanoseconds (ns). The system comprises three molecule types: one protein, and two lipid types (POPE and POPG). The protein type simulates 256 protein molecules while the collective lipid POPE and the POPG types comprise of 14,354 and 4,738 molecules respectively. The collection of the structures is described below:

- Each protein has 171 residues. Each residue consists of 1 to 3 particles. A single protein consists of 344 particles which result in (256 × 344) 88,064 particles in total.
- The lipid collection consists of 19,092 lipids. Each lipid contains 3 groups: i) a head group (2 particles), ii) a tail group (6 particles), and iii) a second tail group (5 particles). Each lipid molecule has 13 particles which result in (19,092 × 13) 248,196 lipid particles in total.

The models discussed are on the basis of the MARTINI coarse-grained forcefield [183]] which does not signify all the atoms, however, simplifies four heavy atoms through one coarse-grained

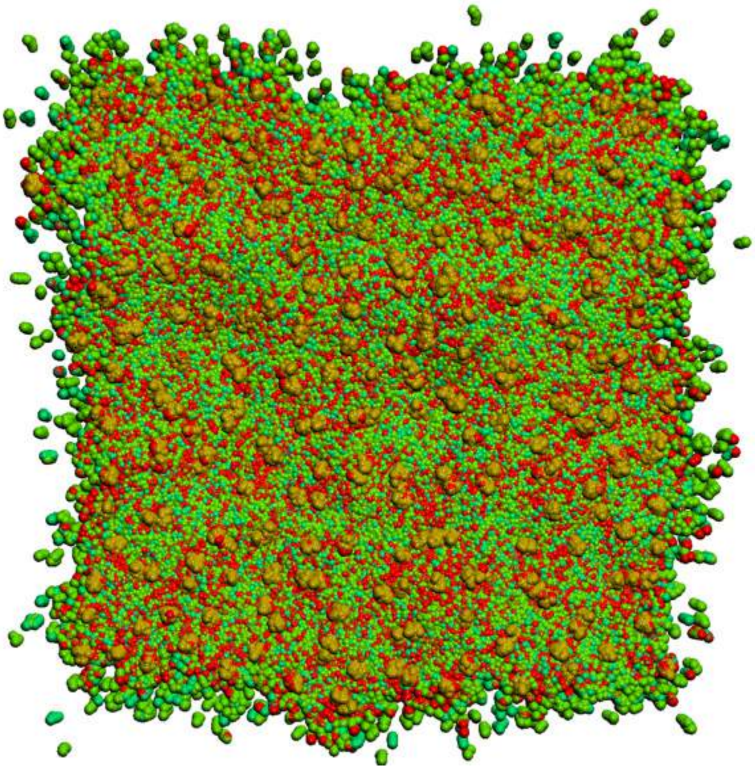


Figure 5.1: A snapshot of a naive Protein-Lipid interaction representation. Protein particles are yellow, and lipid particles are green initially and turn red after an interaction. Perceptual difficulties stem from complexity and occlusion.

particle. Figure 5.2 provides a depiction of the structure of a protein, lipid POPE type, and POPG type. Regarding the size of the simulation, the data-set contains more than 666 million space-time positions that occupy 8 Gigabytes of memory.

5.3 Visualization of Protein-Lipid Interaction

This section discusses the main aspects of the visualization of protein-lipid interaction. It describes the LoD representation of lipids and discusses the relation between lipid LoD representations and the PLI detail at different levels. It also discusses the protein interaction space and describes the LoD representations of the protein-lipid interaction space. The last two subsections discuss the construction of my IBQT (index-based quad-tree) and the projection of the PLI onto the protein LoD space.

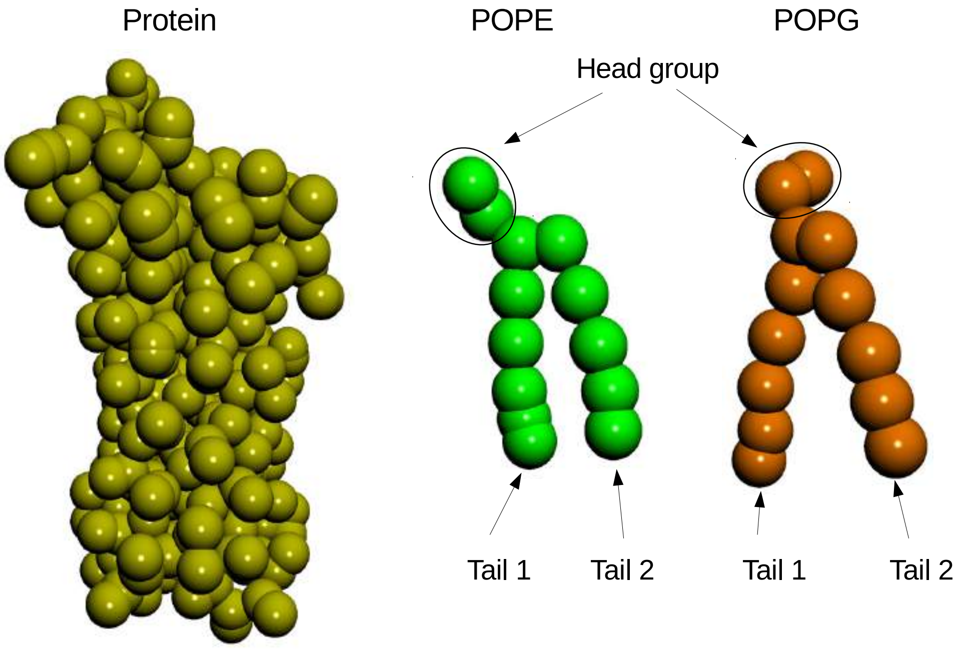


Figure 5.2: Representation of molecules: Protein, POPE and POPG types (left to right). The Protein and the Lipid type images are generated with my tool.

5.3.1 Lipid LoD Representation

The hierarchy for lipid is simplified based on the MARTINI coarse-grained forcefield [183]. Generally, lipids are represented either by an abstraction such as Van der Waals spheres, CPC, licorice [25] and hyperballs [187] or by atomistic representations where every individual particle is depicted by a single sphere (sticks might be used to represent bonds between atoms). I choose the atomistic representation over the abstract representations as the latter lends itself to a smooth LoD representation, especially with dynamic systems. I depict the lipid particles as spheres and cylinders representing bonds between particles. The rendering is accelerated by GPU-based ray-casting techniques for both spheres and cylinders.

Lipid Base-Structure: My lipid LoD representation starts with the lipid base-structure. As described in section 5.2 lipids have a well-defined structure: a head group and two tails. The three groups are connected via a linker particle in such a way that their final representation reflects the Lipid model in Figure 5.2. I employ the geometry of the base-structure to build five derivative structures such that each extends the previous one and represents the lipid at the given LoD.

Derived Structure: The first derived structure consists of only one relatively large particle, i.e., the linker particle which is the representative of the lipid at LoD 1. The second derived structure involves the linker and its closest neighbors which represent the lipid at LoD 2. New structures for LoD 3, 4 and 5 are derived by smoothly extending neighbors of existing particles until a lipid reaches the most detailed representation and smallest particle size at LoD 6. See Figure 5.3.

Smooth LoD Transition: By convention I use the term ”terminal particle” to describe relatively small particles that occupy leaf nodes of the given structure. I utilize a smooth dynamic transition between levels to avoid flickering caused by sudden transitions. The smooth transition is achieved by applying two continuous effects, a lipid-scale, and a particle-scale effect. The lipid-scale effect is responsible for providing a smooth, forward/backward translation for terminal particles between levels. The particle-scale effect is responsible for providing a smooth increasing/decreasing of each terminal particles’ radius during the transition. The speed of the transition is a user option, and it can be configured by increasing or decreasing the transition time. The longer the time, the smoother the transition. Figure 5.3 shows the transition between levels.

5.3.2 LoD Lipid Interaction

One of the properties of the PLI is that it occurs at the atomistic level. This property poses a LoD computational and visualization challenge. The property implies that for every lipid molecule each of its particles needs to be included in the PLI proximity test. For example, if the interaction test is performed at LoD 1 then, for every lipid, only one particle is involved in the interaction test while some of its other particles might pass the interaction test if they were involved. On the other hand, not all the particles of the given lipid necessarily pass the interaction test at the maximum LoD. For example, if the interaction is caused by a particle p that exists only at LoD n I will not be able to reveal its interaction detail at LoD $n-1$ as particle p is not represented at this level downward. These challenges might result in unexpected approximations at all the LoDs except LoD 6 as all the lipid particles are included in the structure. Hence the PLI test is performed on the GPU for LoD 6 as the lipid structure in this level involves all the lipid particles. The result is maintained in full detail. For every

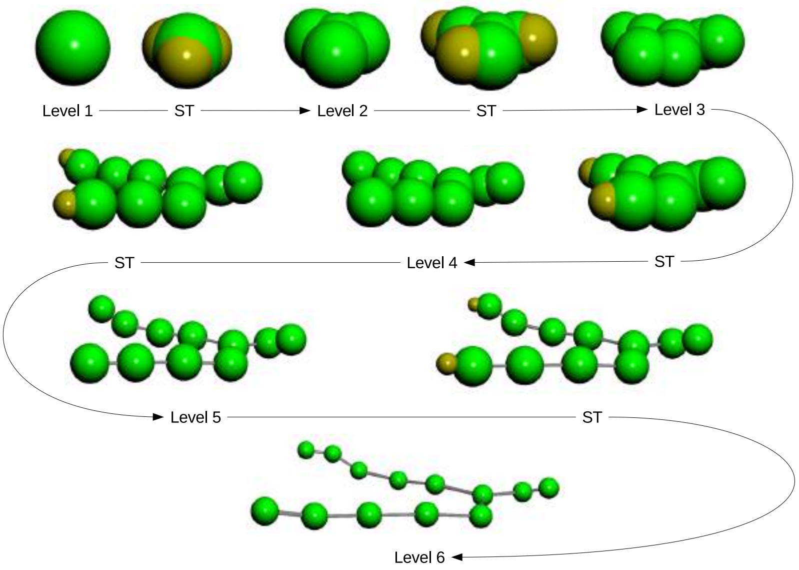


Figure 5.3: Six levels of detail for Lipids. A smooth transition (ST) is applied between the six levels. The size of the sphere shrinks between level as well. The yellow spheres represent the transition stage. The LoD is calculated based on the zoom value.

LoD, I inherit the interaction details from the upper level in such way the visualization reveals the interaction detail for the present LoD and higher approximations. It is clear from Figure 5.4 that adding lipid LoD helps reduce occlusion and clutter. However further innovation is required for protein LoD.

5.3.3 Protein Interaction Space

Cylindrical PLI Representation: A protein interaction space is simplified by an abstract cylinder to enhance the PLI computation performance and simplify the PLI visualization. I choose a cylinder object surrounding each protein to represent the PLI because it's surface is orthogonal to the dominant lipid motion that surrounding each protein. The orientation of the cylinder is orthogonal to the membrane layer orientation aligned with the x, y plane. The abstraction greatly simplifies the perception of PLI compared to trying to convey the same information on a very complex curved surface that typically represents a protein molecule. A cylinder design also enables us to add additional simple details on demand through a straightforward projection onto 2D space. The local projection captures the dominant motion of the lipid particles translating in the x, y orientation. A cylinder also lends itself to LoD representa-

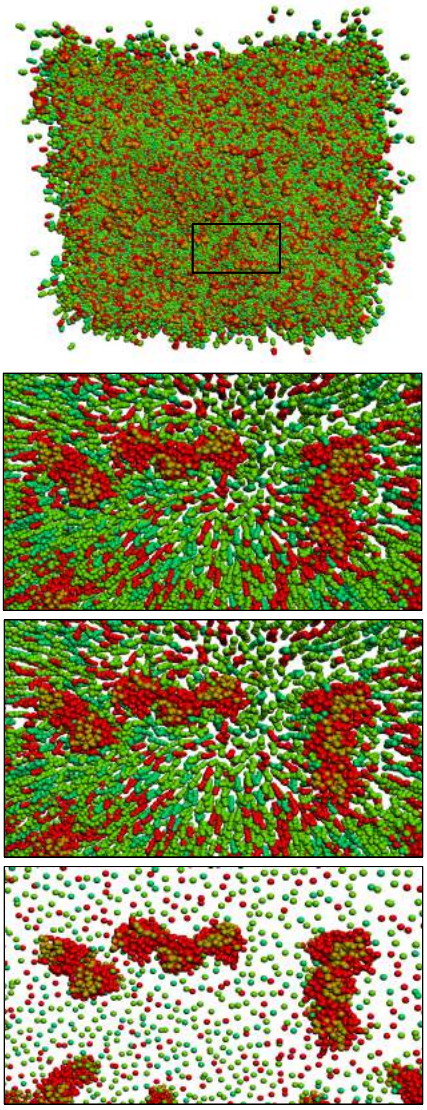


Figure 5.4: An overview of protein-lipid interaction. The top image shows the default representation of the lipid molecules (no level of detail). The middle and bottom images show the representation of the same lipid after enabling the level of detail (level 4 and 1 respectively). Protein particles are yellow, and lipid particles are green initially and turn red after an interaction. In the first image, more interaction details are occluded, due to overlapping particles.

tions. This dedicated abstraction of PLI offers many benefits over traditional complex molecular surface representations, some of which remain unexplored (as future work). I provide two representations, a static representation, and a rotating representation. The key difference between the two representations is that the static representation omits the protein translation and the x and y axis rotation. A cylinder is positioned at the center of mass (CoM) of a protein interaction space. The largest eigenvalue is used for the cylinder height and the second eigenvalue for the cylinder radius. The cylinder mesh is triangulated on the GPU utilizing the protein interaction space CoM, height, and radius. The height and radius can also be adjusted by the user to manipulate the interaction space size. Also, the opacity of the mesh is a user option to provide different levels of transparency. See Figure 5.5.

Testing for Interaction: As described in Section 5.2, a single protein molecule consists of 344 particles and a single lipid molecule consists of 13 particles. An interaction between proteins and lipids occurs if the distance between one protein particle to the nearest lipid particle is equal to or less than a given threshold distance of $0.6 - 0.8$ nm. I provide two estimates of PLI. The first estimate is based on a cylindrical region surrounding each protein whereas the second provides a more accurate interaction test (Figure 2 in the supplementary file shows an illustration of the two approximations). The distance between a lipid particle and the central axis of a protein is calculated first (first approximation). If the first test passes then the distance between every individual protein particle and the given lipid particle can be calculated for higher accuracy (second approximation).

Computing the Protein Interaction Space: The dynamics of protein particles play a significant roll in defining the geometry of a protein at a given time step. Some protein properties (e.g., global position, a CoM, principal axis, dimensions, and rotation) might be influenced by the simulation dynamics. As a result, a protein interaction space must be updated for every time step. I utilize the Singular-Value Decomposition (SVD) [184] to calculate the main properties of a protein interaction space i.e. principle axis (eigenvectors), dimension sizes (eigenvalues) and rotation ($V \times U^T$). If A is a symmetric real $n \times n$ matrix, there is an orthogonal matrix V and a diagonal, D, such that $A = VDV^T$. Here the columns of V are *eigenvectors* for A and form an orthonormal basis for R^n; the diagonal entries of D are the *eigenvalues* of A [185]. To emphasize the connection with the SVD, I refer to VDV^T as the eigenvalue decomposition, or EVD, for A. The eigenvectors represent the axes of a protein interaction space, and the associated eigenvalues represent a protein interaction space extent. See Figure 5.7. The first eigenvector is used to represent the z axis, and the second and third eigenvector are used to represent the x axis and y axis. However, the order of the second and the third eigenvector is not necessarily constant over time which needs to be consistent. This challenge is addressed by performing two tests; the first test is performed at $t = 0$ to make sure that the x axis and the y

axis are in the correct order. The second test is performed at every time step $t = n$ where $n > 0$ to maintain consistency the x, y and z axes.

5.3.4 LoD Protein-Lipid Interaction space

PLIs can occur at any position within a protein space. By utilizing an abstract cylinder to represent a protein space, a PLI can be directly depicted on the surface of the abstract protein space using a tile. Essentially, the surface of the abstract protein space is divided into 1024 tiles. Each tile is associated with an IBQT (index-based quad-tree) node (see Section 5.3.5). The IBQT nodes are responsible for maintaining the PLI detail while tiles are used to depict the nodes' information. I exploit the properties of IBQT to apply LoD techniques to a PLI space. The root of the IBQT represents the first LoD. The PLI frequency is accumulated from the nested levels before it is projected on the cylinder. The LoD level 1 helps the user obtain an overview of the PLI. The leaf nodes of level 6 represent the sixth LoD. Each node is mapped to a tile on the cylinder. LoD 6 provides the user with the PLI at the highest resolution where each tile depicts the smallest interaction area on the abstract surface. The user can investigate the PLI at different resolutions in between LoD 1 and LoD 6. The LoD is calculated based on the camera zoom level. The zoom interval between levels is parametrized enabling the user to control the switching time between levels. The user also is provided with a manual LoD slider. The manual LoD slider helps the user to focus on the PLI at particular LoD while zooming in and out or at a fixed camera position. Figure 5.5 illustrates six LoDs for a single PLI.

5.3.5 Constructing the Interactive Index-Based Quad-Tree (IBQT)

Taking into account 256 simulated proteins, I maintain a dynamic LoD interaction representation i.e each interaction position, time and frequency over 1981 time-steps. In terms of storing and retrieving the interaction data in a spatial data structure, there are a variety of solutions discussed by Samet [188]. However, the key challenge here is posed by the dynamics of the simulation. The interaction position of every individual particle needs to be identified, translated, and rotated, per time step with the given protein. Another challenge is that the PLI occurs in a three-dimensional space represented by an abstract cylinder. A naive solution might require conversion between a cylindrical coordinate system to a 3D Cartesian system. The conversion is necessary to map the PLI from/to the cylindrical space to a 3D Cartesian space which can be represented by a quad-tree. I address these challenges by proposing a fast quadtree-based map-

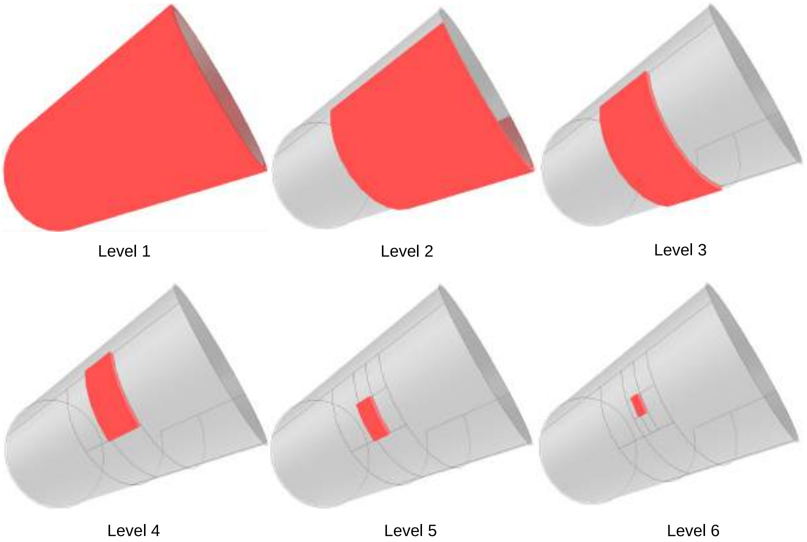


Figure 5.5: Six levels of detail to represent protein interaction. The color of each tile is mapped to the number of interactions with each tile's extent. The interaction detail is maintained by passing the number of particle interactions down to the next level. The LoD is dynamically updated based on the camera zoom value.

ping approach using the given protein's local coordinate space. My approach eliminates the need for storing the global interaction position of each individual particle or updating its translation or rotation. Also, it eliminates any explicit conversion between cylindrical and Cartesian coordinate systems. See Section 5.3.6.

I utilize a custom index-based quad-tree (IBQT) to capture and store the protein-lipid interactions. The IBQT is designed to accommodate a GPU-based projection onto each protein's local cylinder. Because the PLI is represented with an abstract cylindrical shape, the local polar system of each protein can be exploited in the IBQT design. The IBQT is inspired by the Samet's sector tree [188]. Samet [188] discusses a sector tree structure that utilizes a polar coordinate system to maintain an object's spatial data in two dimensions. In the sector tree, the angle of the intersection between the boundary of the two regions participates in its construction. The common facet between the sector tree and the IBQT is that in both angles play a key role in defining a tree structure. My IBQT differs from the sector tree in two aspects. First, my structure is a point-based quad-tree structure while the sector tree is region-based. Second, my IBQT encodes a three-dimensional position to represent a PLI in 3D space. Each IBQT node has a key index (called a base index). The base index of a node is used to identify the node with respect to a given LoD. The node index consists of two components; the first component encodes the interaction position along the z axis while the second component encodes the angle of the interaction. Every interaction node stores the interaction frequency, a list of particles involved in the interaction, and the interaction time-step. Figure 5.6 depicts my IBQT and illustrates the PLI encoding. The index components are derived from the interaction position by applying the following:

$$u1 = \frac{h}{2^{LoD-1}} \tag{5.1}$$

Where h indicates the height of the cylinder used to represent a protein's interaction space.

$$u2 = \frac{2\pi}{2^{LoD-1}} \tag{5.2}$$

With respect to a given LoD, Equation 5.1 calculates the size of a unit scale that span the z axis while 5.2 calculates the size of unit scale spanning 2π.

$$i_z = floor(\frac{z}{u1}) \tag{5.3}$$

Equation 5.3 calculates i_z which represents the PLI along the z axis with respect to a given LoD. The following calculates the angular value i_{xy} of the PLI with respect to a given LoD:

$$i_{xy} = floor(\frac{DoublePI(arctangent(y,x))}{u2}) \tag{5.4}$$

The function *DoublePI()* returns a positive *radian* $\in [0, 2\pi]$, *arctangent* returns an angle confined to the interval $[-\pi, \pi]$ and positive for $y > 0$. Equations 5.3 and 5.4 encode the PLI position into an IBQT node index. The index consists of two components. The first and second components are represented by i_z and i_{xy} respectively. Finally, I construct the IBQT by utilizing the derived node index instead of the global interaction coordinates. Figure 5.6 illustrates process of encoding the PLI position into IBQT node index. The following equation is used to find the index of the parent node index (PNIndex) of any given node:

$$d_f = \frac{2^{LoD_{max-1}}}{2^{LoD-1}} \tag{5.5}$$

LoD_{max} indicates the maximum level of detail. The d_f is used to calculate the parent node's index based on the first and second components of the given node index. It ensures that the node index $\in [0, 31]$.

$$PNIndex = (floor(\frac{i_z}{d_f}) \times d_f, floor(\frac{i_{xy}}{d_f}) \times d_f) \tag{5.6}$$

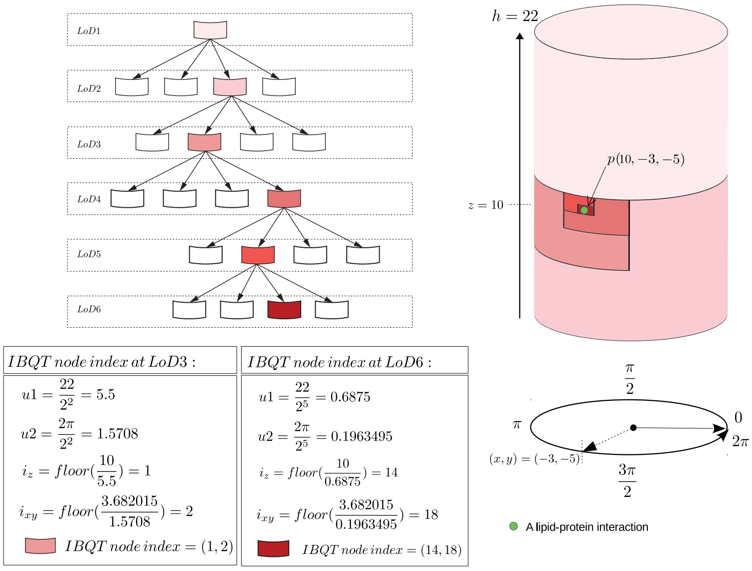

LoD1
LoD2
LoD3
LoD4
LoD5
LoD6
$h = 22$
$p(10, -3, -5)$
$z = 10$
$IBQT\ node\ index\ at\ LoD3:$
$u1 = \frac{22}{2^2} = 5.5$
$u2 = \frac{2\pi}{2^2} = 1.5708$
$i_z = floor(\frac{10}{5.5}) = 1$
$i_{xy} = floor(\frac{3.682015}{1.5708}) = 2$
$IBQT\ node\ index = (1, 2)$
$IBQT\ node\ index\ at\ LoD6:$
$u1 = \frac{22}{2^5} = 0.6875$
$u2 = \frac{2\pi}{2^5} = 0.1963495$
$i_z = floor(\frac{10}{0.6875}) = 14$
$i_{xy} = floor(\frac{3.682015}{0.1963495}) = 18$
$IBQT\ node\ index = (14, 18)$
$\frac{\pi}{2}$
π
0
2π
$(x, y) = (-3, -5)$
$\frac{3\pi}{2}$
● A lipid-protein interaction

Figure 5.6: This image illustrates the computation of the IBQT node's index. An IBQ node index is encoded on the GPU by applying Equations 5.1, 5.2, 5.3, 5.4 to an interaction position.

5.3.6 Projecting the Interaction on to the LoD Protein Space

The interaction mapping process is performed on the GPU utilizing a geometry shader. The advantage of this approach is that it is no longer necessary to store, update, translate or rotate the interaction positions for every time-step because they are based on the given protein's local coordinate system. First I retrieve the interaction data from the IBQT using three parameters: the time step, the LoD and a vector of a *TileData* object. The time step and the LoD parameters are used to specify the IBQT query. The *TileData* vector returns a list of *TileData* objects. Each *TileData* contains an interaction node index, a molecule type identifier and interaction frequency. Then, based on the chosen level, every non-empty node is projected and visualized by a tile on the surface of the corresponding abstract protein space. The projection involves three steps. The first step is to project the left edge of the interaction node on the surface. The second step to identify the right edge of the interaction before finally forming a tile by constructing a curved, triangular mesh between the two edges. To identify the left edge, I derive two 3D points from the interaction node index. For a given LoD, I calculate the z_{start} and the z_{end} of the left edge of the interaction node.

$$z_{start} = float(i_z) \times u1 \tag{5.7}$$

$$z_{end} = \frac{2^{Lod_{max}-1}}{2^{LoD}} \times u1 \tag{5.8}$$

The interaction angle is computed and used to construct the *interaction normal* $\vec{\boldsymbol{n}}_i$

$$i_\theta = \frac{i_{xy}}{2^{Lod_{max}-1} \times 2\pi} \; (5.10) \tag{5.10}$$

The $\vec{\boldsymbol{n}}_i$ components are computed as:

$$\vec{\boldsymbol{n}}_{i(x)} = cos(i_\theta) \times \vec{\boldsymbol{x}}_x + sin(i_\theta) \times \vec{\boldsymbol{y}}_x \tag{5.11}$$

$$\vec{\boldsymbol{n}}_{i(y)} = cos(i_\theta) \times \vec{\boldsymbol{x}}_y + sin(i_\theta) \times \vec{\boldsymbol{y}}_y \tag{5.12}$$

$$\vec{\boldsymbol{n}}_{i(z)} = cos(i_\theta) \times \vec{\boldsymbol{x}}_z + sin(i_\theta) \times \vec{\boldsymbol{y}}_z \tag{5.13}$$

Where $\vec{\boldsymbol{x}}$ and $\vec{\boldsymbol{y}}$ are the *eigenvectors* in the *x-direction* and *y-direction* of the protein interaction space respectively. The z_{start}, the z_{end} and $\vec{\boldsymbol{n}}_i$, then, can be used to find the left edge start and end points:

$$i_s = s_{CoM} + (z_{min} \times \vec{\boldsymbol{z}}) + (z_{start} \times \vec{\boldsymbol{z}}) + \boldsymbol{r} \times \vec{\boldsymbol{n}}_i \tag{5.14}$$

$$i_e = s_{CoM} + (z_{min} \times \vec{\boldsymbol{z}}) + (z_{end} \times \vec{\boldsymbol{z}}) + \boldsymbol{r} \times \vec{\boldsymbol{n}}_i \tag{5.15}$$

Where z_{min} is the min z value of the cylinder. The terms s_{CoM} and $\boldsymbol{r}$ are the center of mass and the radius of the abstract protein space respectively and $\vec{\boldsymbol{z}}$ is the *eigenvector* in the z direction of the protein interaction space.

Similarly, I can obtain the starting and ending points of right edge by adding the *tile segment count* to the *second component* and apply Equations 5.14 and 5.15. See Figure 5.7.

5.4 Experimental Results

The new visual design enables the user to investigate the system and obtain insight into the PLI. See the accompanying video for further visualization results at:

https://github.com/NaifAlhar6i/Chapter5

Figure 5.9 shows three different LoD representations for the PLI. The top image depicts lipid particles and a PLI space at high LoD (LoD 6). The visualization at this resolution reveals more PLI detail. i.e., each tile represents the number of PLI that take place within its extent. The middle image shows lipid particles and PLI at LoD 4. The PLI frequencies are spatially aggregated. The user can easily observe regions that receive the highest number of interactions. The bottom image shows lipid particle and PLI space at low LoD. This visualization is useful

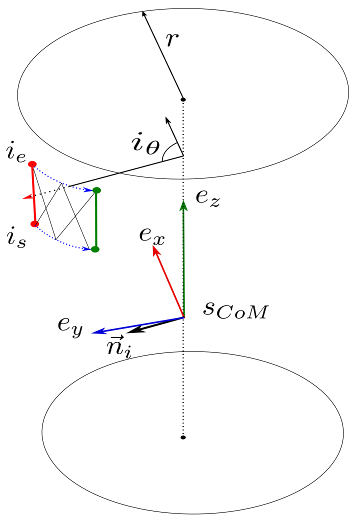


Figure 5.7: This image illustrates my GPU-based LoD projection. A protein interaction space and the index of a non-empty node are used to construct the associated tile on the surface of the cylinder. The red and green segments represent the left and right edges of the interaction node respectively. The blue dotted arrows represent two curves that connect the left and right edges to construct a tile.

for an overview. It can guide users to identify proteins that receive the highest frequency of interaction for further investigation.

Color Map Schemes and Adjusting the Color Map Legend: I provide the user with two-color maps. A continuous color map based on Telea [189] and a categorised color map using

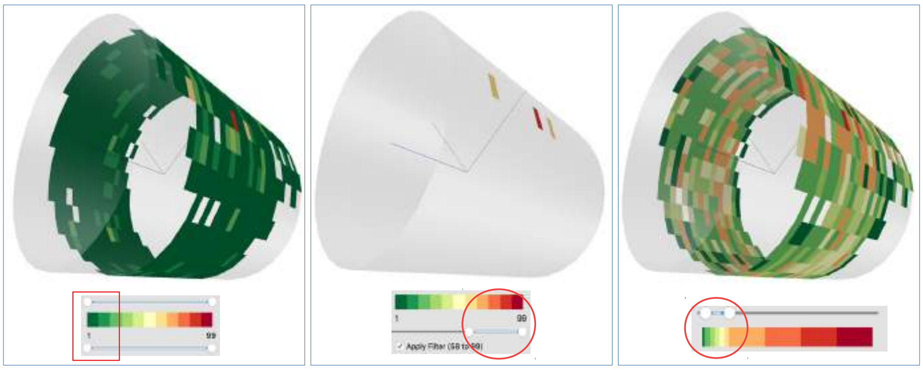


Figure 5.8: PLI filtering and color map steering. The left image shows no filtering and no modification to the color map algorithm. The middle image shows highest frequency PLI using the filtering slider. The right image shows the effect of steering the underlying map to enhance color distribution.

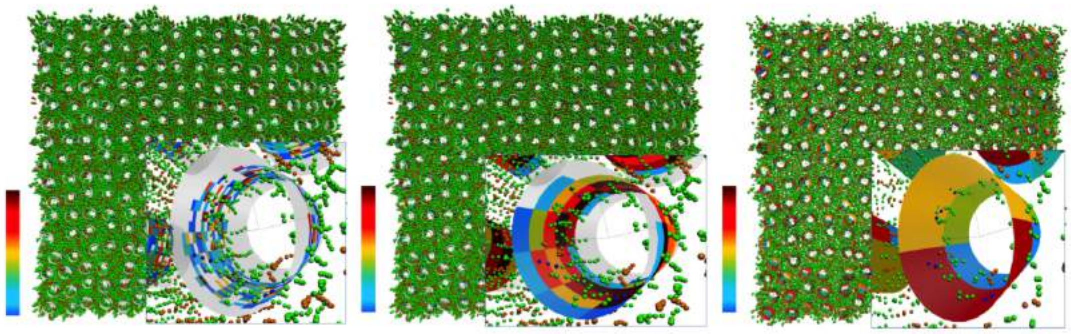


Figure 5.9: Three LoD PLI representations. PLI frequency is mapped to color. From top to bottom, PLI at high resolution, medium resolution, and low resolution. Lipid particles are green and brown. Blue spheres indicate lipid interaction particles.

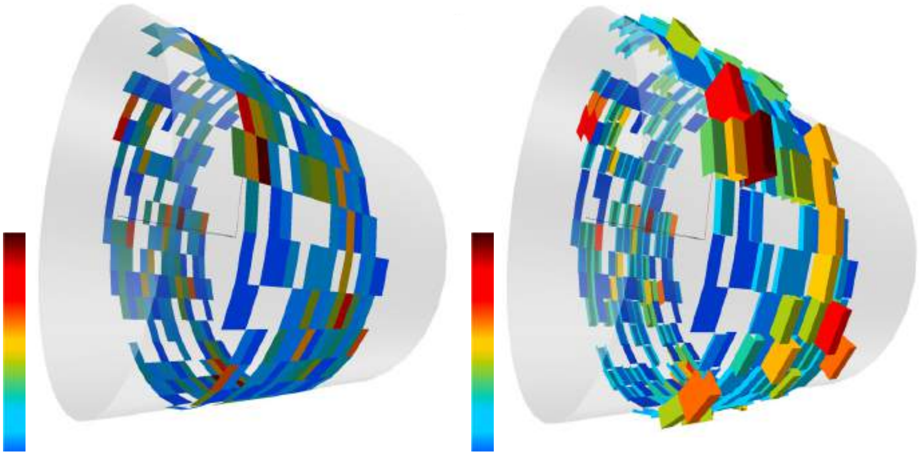


Figure 5.10: (Left) the number of protein-lipid interactions is mapped to color. (Right) the same data map to the height of the tiled interaction cylinder.

Color Brewer [190]. The PLI frequency is mapped to the selected color schema. However, the PLI histogram shows an uneven distribution which leads some colors to dominate. A naive approach to address this challenge is to clamp and rearrange the underlying data to obtain a better color distribution. However, this method might result in a mapping error for data that lay outside the lower and upper boundaries. I address this challenge by enabling the user to adjust the color map's underlying look-up table. The user can control the color map's look-up table through a range slider to obtain more color distribution. The right image in Figure 5.8 shows an example of using the color map slider.

PLI Frequency to Tile Height: A three-dimension histogram can enhance the visualization. The height of the histogram encodes a property to emphasize it. I enable the user to map the PLI frequency to the tile height. Figure 5.10 depicts the PLI by a simple histogram by mapping the PLI frequency to tile height.

PLI Space Cluttering Reduction: When two or more protein spaces intersect each other, they cause visual cluttering. The user may reduce cluttering by either omitting the back face of a protein cylinder or rendering the cylinder lids.

Lipid Type and Protein Selection and Filtering: Three types of molecules are involved in the PLI. The PLI molecule filter enables the user to focus on one or more molecule types. Applying the PLI molecule filter on one or more molecule types showing only the PLI of the selected types. The user also can filter the PLI based on the PLI frequency. The PLI frequency filter accepts a range of values between the PLI min and max of the current histogram data.

Focus-and-Context: I apply the focus-and-context technique to reduce visual complexity. It can result in more detailed visualization, especially for an overview. See the accompanying video for a demonstration.

5.5 Conclusion

In this Chapter, I proposed six levels of detail for both lipid types and the PLI abstract space. The visual design of the abstract space simplifies the PLI problem and helps users obtain insight into the PLI. The LoD lipid representations reduce clutter caused by lipid particles. The LoD PLI space representations enable users to investigate the PLI space at the desired LoD. I also introduced a number of useful user options to aid the PLI visualization. In Chapter 6, I aim to focus more on combining 2D and 3D visualizations. Properties of cylinder shapes can be exploited to map the PLI from a cylinder onto a 2D plane.

Chapter 6

Hybrid Visualization of Protein-Lipid Interaction and Protein-Protein Interaction

Contents

"(visual) images—have some pictorial properties, but they are of limited capacity and are actively composed."-S. M. Kosslyn, Pinker, Smith, & Shwartz

Chapter 6 is built on Chapter 4 and 5. It describes a novel PLI and PPI visualization framework. The framework utilizes four visual designs that enable the user to study a time-dependent membrane simulation. It employs abstraction and space projection to address a number of visualization challenges. It also utilizes a novel hybrid view to enable the user to study PLI and PPI and the behavior of the PPI and clusters.

6.1 Introduction and Motivation

In computational biology, scientists rely on simulation to study life cells and the behavior of molecules. Monte Carlo (MC) sampling and Molecular Dynamics Simulation (MD) are widely used to simulate large molecular systems (Frenkel and Smit [191]). With the recent advances in GPU computing and acceleration techniques, MC and MD software can produce very large time-dependent simulations. Simulation involves the evolution of molecular systems in the form of sequential position-snapshots that represent the trajectory of particles. The nature of dynamical systems is influenced by many molecular phenomena such as molecular interactions and molecular clustering.

Molecular interactions are studied in different fields such as protein folding, drug design, and origins of life study. Spatio-temporal protein clustering also plays an essential role in studying the spatio-temporal dynamics of membranes [143]. The molecular interactions occur at the atomic level resulting from attractive or repulsive forces between molecules and between non-bonded atoms. However, a molecular interaction can involve molecules of different types and sizes, e.g. protein-lipid interaction (PLI) or molecules from the same classes and sizes, e.g. protein-protein interaction (PPI). Protein spatio-temporal clustering occurs when two proteins (or more) are in close proximity to make an interaction.

Molecular interactions are often classified into two interaction types; PLI and PPI. The two types of interaction occur in the same physical space which increases the challenge of

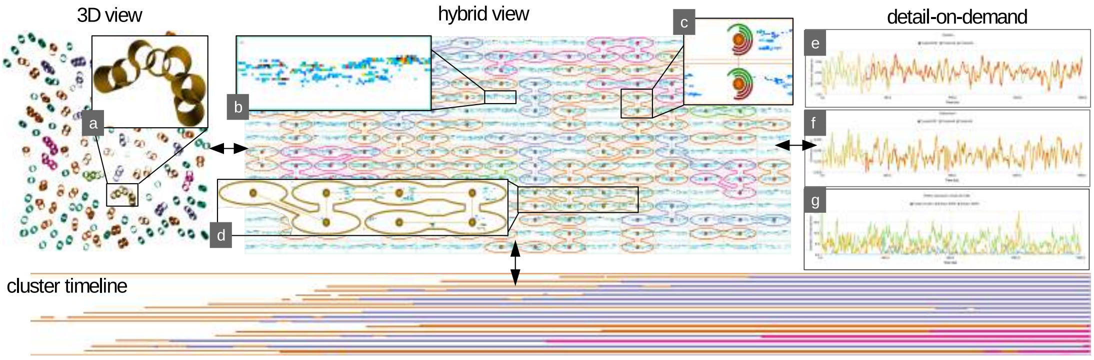


Figure 6.1: Four views of Protein-Lipid Interaction (PLI) and Protein-Protein Interaction (PPI). The 3D view (top left), a) zoomed in snapshot of a cluster in the 3D view. The 2D hybrid view (top middle). Abstraction is used to explore PLI and PPI in 2D space. b) PLI depicted with a tiled heat-map style visual design. c) A depiction of PPI, and rotation. The PPI is represented by a tiled visual design (same as PLI) and a glyph is used to convey the protein behavior. d) A large cluster representation (the same selected cluster in the 3D view). The hybrid 2D view enables the user to couple and decouple the PLI and PPI imagery. Details-on-demand (top right), e) and f) interactive time series graphs display the amount of rotation and displacement of a cluster and its proteins respectively. g) an interactive time series that displays the frequency of interactions between a given protein and other molecules (a protein and two lipid types). The cluster timeline (bottom), an interactive timeline by which the user can track the history of each cluster. The framework can be used to study the historical evolution of a cluster. Any cluster can be selected in the 3D view to be analyzed in the hybrid view. The user can observe details and investigate a particular protein from the cluster. More detail about the protein such as PLI, PPI, and amount of rotation can be conveyed in the form of glyphs and graphs.

understanding the behavior of the simulation. The interplay between PLI and PPI is expected to be a determinant factor in the assembly and dynamics of such membrane complexes [192]. Nevertheless, it is helpful to study the PLI and PPI in a unified context. Figure 6.2 shows the visualization of PLI and PPI in complex simulation data utilizing a naive approach. It can be seen, in addition to the view-dependency problem in the 3D visualizations, most of the interaction details are obscured due to particle overlap and occlusion. If we try to use color-mapping to visualize both PLI and PPI in the traditional 4D coordinate system, we encounter occlusion, visual complexity, and overlap. However, domain experts are still interested in analyzing and visualizing these two types of interactions both together and separately. Thus, I propose a novel, hybrid unfolded view that enables this. The PLI is depicted as a projected tiled space in a heat-map style visual design, while the PPI can be represented in the same coordinate space

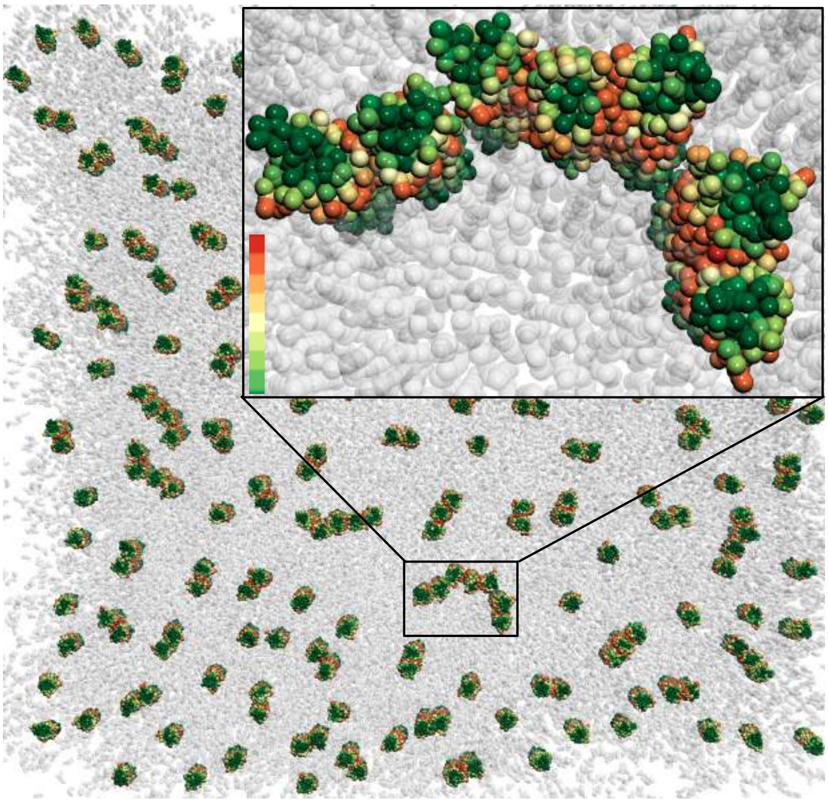


Figure 6.2: Visualizing PLI and PPI using a naive approach. Protein particles are color mapped to the number of interactions they received. Lipid particles are rendered as context to reduce occlusion, overlap and visual complexity.

using tiles or glyphs which are easily separable from the tiled PLI visual design. By exploiting visually separable visual designs in a unified coordinate space I can provide the domain expert with a unified view of PLI and PPI with reduced occlusion and overlap while at the same time enable focusing on either PLI or PPI. I propose a novel visual design utilizing a tiled map of both PLI and PPI based on the abstract interaction space of proteins described in Chapter 5. Similarly, protein clustering and PPI occur in the same physical space and are associated with each other. It makes sense that they are studied in a unified visual framework. I employ abstract glyphs to convey the dynamic behavior of PPI and the different dynamic properties of proteins and their clusters.

In this chapter I propose a novel molecular interaction and protein clustering visualization tool including a linked 3D, visual abstract, and novel, projected hybrid view to convey a unified representation of PLI, PPI, and protein clustering. The tool enables the user to explore PLI, PPI and protein clustering interactively in a unified system. Additionally, I enable the user to analyze the details of an individual protein molecule or cluster by applying selection, focus-and-context, zooming, and details-on-demand techniques. Computational biologists can investigate both PLI and PPI, and protein clustering using a hybrid 2D and 3D view in a unified tool. I make the following contributions in this Chapter:

- The first tool to both visually separate and combine time-dependent PLI and PPI imagery in a unified visual design. My novel design is based on projecting the cylindrical interaction-space of each protein into a tiled, time-dependent 2D space.

- A novel, glyph-based representation of PPI that also encodes rotational properties. I utilize abstraction to depict proteins and clusters in a hybrid 2D view side-by-side the 3D view. Dynamic glyphs are used to visualize time-dependent properties such as angle of cluster rotation, displacement, and radius.

The rest of the Chapter is structured as follows. After discussing related work in Section 2.3.5, I describe the simulation and my methodology on computing the properties of PLI, PPI and protein clusters in Section 6.2. The molecular interaction and protein clustering visualization approaches are described in Section 6.3. In Section 6.4 I provide two case studies and report domain-expert feedback.

6.2 Simulation Data and Properties Computational Methodology

In this section I describe the simulation data and briefly discuss the methodologies I use to compute meta-data of interaction properties.

6.2.1 Simulation Data Description

The dataset I study represents the behavior of a membrane constituted by lipids and proteins resulting from a molecular dynamics simulation. The simulation is used to study the dynamic behavior of both proteins and lipids in the context of a membrane aligned primarily in the xy plane [151]. The dimensions of the system are $116 \times 116 \times 10$ nanometers (x, y, and z respectively) and the simulation represents individual trajectories of 336,260 particles over almost 2 microseconds (stored in c.a. 2000 frames). The system consists of three molecule types: one protein, and two lipid types (POPE and POPG). There are 256 protein molecules while the number of POPE and the POPG lipid molecules are 14,354 and 4,738 respectively. The models presented are based on the MARTINI coarse-grained forcefield [183] which does not represent all the atoms but simplifies four heavy atoms into one coarse-grained (CG) particle. Thus, the hierarchy for each type of molecule is simplified:

Each protein is constituted by 171 residues, and each residue is composed of 1 to 3 particles. The total number of particles per protein is 344 resulting in 88,064 particles in total (256 proteins × 344 particles).

The total number of lipids (POPG and POPE) is 19,092. Each lipid molecule has 13 particles which result in (19,092 molecules × 13 particles) 248,196 particles in total. For this representation, I divide each lipid into three groups: a head group consisting of 2 particles, a tail group consisting of 6 particles, and a second tail group consisting of 5 particles. In terms of the data size, the simulation contains more than 666 million space-time positions that occupy 8 Gigabytes of memory.

6.2.2 Computation

Deriving Rotational Data In the model of the membrane studied here, each protein molecule is constituted by a finite number of particles (344 particles). These particles are linked together through constraints defined by the forcefield which limits the overall displacement of one particle with respect to the others belonging to the same protein. The dynamics of protein particles play a significant role in defining the geometry of a protein at a given time step. Some protein properties (e.g. global position, Center of Mass (CoM), principal axes, spatial extent, and rotation) are influenced by the system dynamics. As a result, these properties should be updated for every time step. I use Singular-Value Decomposition (SVD) [184] to calculate the main properties of a protein space i.e., the principal axes (*eigenvectors*), dimension sizes (*eigenvalues*) and rotation matrix as described in Chapter 4. In the case of clusters, proteins that form a cluster are addressed as a unified entity that has its own properties (e.g. global position, CoM, principal axes, and rotation).

Protein Radius The size and shape of a protein are often estimated, and the result varies depending on the mass of the protein. A sphere is often used to approximate the shape of a protein. The radius of the sphere can be calculated utilizing a number of parameters as described in Erickson [193]. In this work, instead of the spherical shape, I employ a cylindrical shape dedicated specifically to the protein interaction space as described and computed in

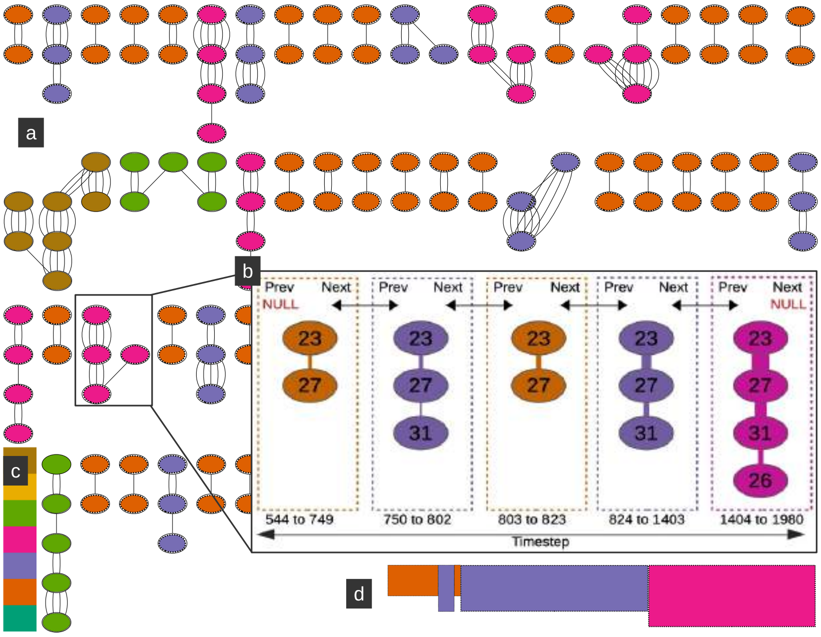


Figure 6.3: a) Image shows a static, disconnected graph consisting of 172 connected sub-graphs (based on PPI). Each sub-graph represents a cluster snapshot (based on the final timestep). Cluster color encodes the size of the cluster. Nodes represent a protein. Edges between protein pair represent PPI at the molecule level (i.e. the protein pair are attached). In the static graph, the number of edges between a protein pair indicates how many times the pair participates in forming a cluster. b) A resultant Time Varying Graph (TVG) shows the evolution of a cluster of four proteins. The thickness of the edge between protein pair encodes the length of the connection. Each dashed box represents a cluster with respect to the time span of its PPI. The arrows show the relationship between clusters within the TVG. c) A color map legend, the color mapps to the the cluster size between 1 to 7 (bottom to top). d) A cluster timeline of the relevant TVG. The height of the clusters in the timeline also encodes the cluster size to make the small cluster more visible.

Chapter 4. The calculation results in a CoM and three principal axes (eigenvectors). The major axis is aligned with the height of the cylinder and cuts through the membrane while the other two axes are aligned with the cylinder base. Therefore, the radius of the cylinder can be estimated by the largest distance between the major axis and particles that lie in the *xy* plane. For each time-step, I calculate the radius of every individual protein. The result is utilized in spatio-temporal protein clustering.

Spatio-temporal Clustering Spatio-temporal clustering is a result of PPI. Proteins within 5 nm of one another are considered as interacting Chavent *et al.* [143]. The usual threshold represents a static diameter of a protein. Based on my calculations, the size of a protein's radius (the radius of a cylinder) varies between 1.7 nm to 2.5 nm, which implies that utilizing a static threshold might result in a less accurate PPI. In this work, I utilize a dynamic protein radius to determine PPI. I redefine the threshold distance of PPI based on a protein pair within a distance less than or equal to the sum of their respective radii.With this definition, I ensure that protein pair is always close enough to each other to initiate an interaction.

Cluster Timeline Construction In order to construct a cluster timeline I utilize a custom time-varying graph (TVG). A node represents a protein and an edge represents a connection (a PPI at the molecular level) between a protein pair. The lifespan of the connection between a protein pair is stored in the corresponding edge. The construction of the timeline is a three step process: 1) constructing a static graph of the PPI (for the whole simulation Figure 6.3), 2) deriving a time varying graph (TVG) from the sub-graphs of the static graph (Figure 6.3 - a), 3) building the cluster timeline by using the TVG (Figure 6.3 - b). First, the static graph is constructed by inputting all existing protein pairs including time steps at which they interact. The result is a large, disconnected graph consisting of a number of sub-graphs. Each sub-graph represents a snapshot of a cluster's lifespan. I define a sub-graph by its connectivity i.e. a sub-graph is said to be connected if there is a path between every pair of nodes. A node pair might have more than one edge. Each edge represents the interaction in the input time span (from the first step). Second, for each sub-graph, I iterate through its edges and nodes to build a TVG. With respect to time, each sub-graph is linked to its previous and subsequent sub-graph in the time series. The conjunction of two edges that have a node in common results in a new sub-graph until no conjunction is found in the TVG. Finally, the TVGs are used to visualize the cluster timeline by converting the sub-graphs into timelines. See Figure 6.3. The clustering timeline provides historical information on protein clustering. It is useful to understand the behavior of proteins prior to and after forming a cluster.

6.3 Hybrid Interactive MD Visualization

My compact MD visualization enables the user to explore three MD interaction types in a unified context. In this section, I provide an overview of the system design followed by a description of the PLI, PPI, and clustering visualization.

6.3.1 Design Overview

The primary aim of my design is to enable the user to explore and analyze PLI, PPI and protein clustering interactively in the same context. The system is designed to show the history of the behavior within a molecular dynamics system. The first step in the workflow starts in the 3D view (Figure 6.1 - top left). The user observes the spatio-temporal clustering of proteins and selects a desired cluster in the 3D view. Then, the hybrid view (Figure 6.1 - top middle) can be used to obtain additional information about the system and the selected cluster. Finally, details concerning the molecular interactions and the properties of proteins and clusters can be visualized in the form of graphs (Figure 6.1 - top right). My visual design includes an interactive clustering timeline view (Figure 6.1 - bottom) to enable the user to investigate the clusters over time. To realize this, the design should fulfill the following requirements: 1) the user must be able to investigate PLI, PPI, and protein clustering simultaneously, and 2) the three visualizations (PLI, PPI, protein clustering) must be linked to each other. To do so, I propose three resizable views, 1) a 4D overview, 2) a hybrid overview, 3) a clustering timeline view, and 4) a details-on-demand view. See Figure 6.1. The hybrid, 2D time-dependent view and 4D view provide the user with an overview of the system with spatial information. Both views offer a zooming feature which can be used to explore a single molecule or a cluster in both hybrid 2D and 4D. The clustering timeline enables the user to investigate any cluster at any time step in the simulation. The details-on-demand view provides information concerning the behavior of the system in the form of graphs. The four views are linked to each other in such a way the user can interact with the system using any of them. For example, if the user observes an interesting pattern in the line-chart graph he/she can steer the simulation directly to the corresponding timestep by clicking on the pattern to update the other views. The justification of the design for each view is described in the corresponding subsections.

6.3.2 Hybrid Visualization of PLI and PPI

Protein surfaces are often used to depict molecular interactions in 3D. The density of the interaction, for example, can be color-coded on the surface of a protein to indicate its accessibility. However, this approach poses three challenges. Keeping track of the motion of more than 8 objects, even at slow speed, over 4D space-time is beyond general human perception [194]. Hence, he first challenge is posed by the complex motion of multiple proteins in 4D space-time. The second challenge is caused by the complex geometry of protein surfaces. The third

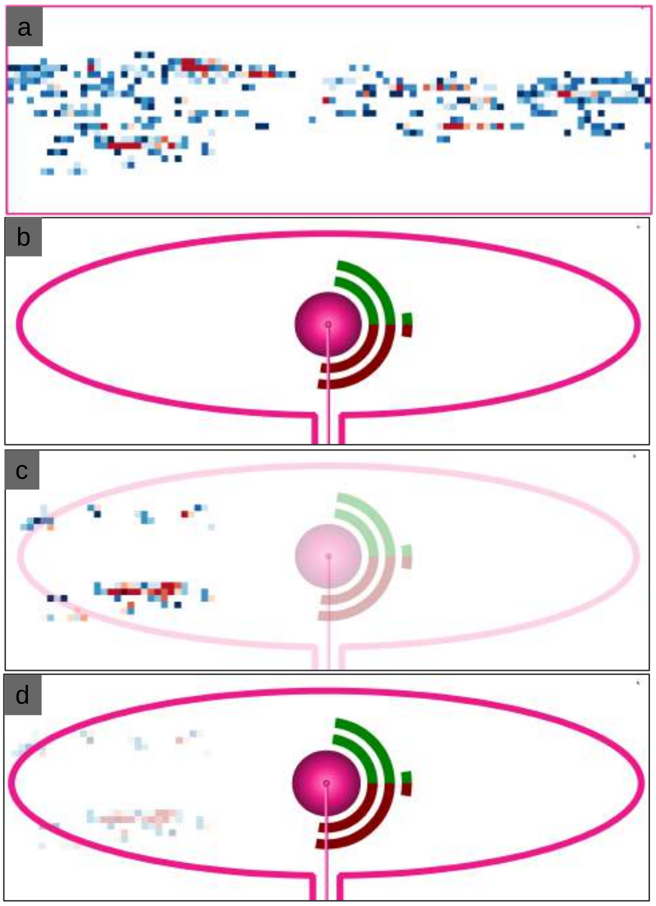


Figure 6.4: Four different visualizations of the same protein at time step 1980 utilizing the hybrid view. The protein is part of a cluster. a) Visualization of the projected, tiled space of both PLI and PPI. b) Visualization of the clustering and protein behavior. c) Coupling PPI tiled space and PPI behavior with a focus on the tiled space. d) Coupling PPI tiled space and PPI behavior with a focus on clustering and protein behavior.

challenge is associated with the nature of 3D visualizations in general (view-dependency and occlusion). I exploit abstraction and projection to address these challenges. In the hybrid view, each protein is projected onto a fixed position in 2D space (the initial 3D seeding point) and is depicted via a glyph. See Figure 6.1 - top middle. Each protein is the center of its own local coordinate system in this new grid-style layout (Figure 6.5 - right). This new protein coordinate and abstraction visual design addresses the challenges posed by the complex motion of protein and the general 4D visualization challenges. To address the challenge posed by the complexity of the geometry I utilize the protein abstract interaction space as described in Chapter 5. It simplifies the problem by representing PLI and PPI (at the particle level) using a dedicated

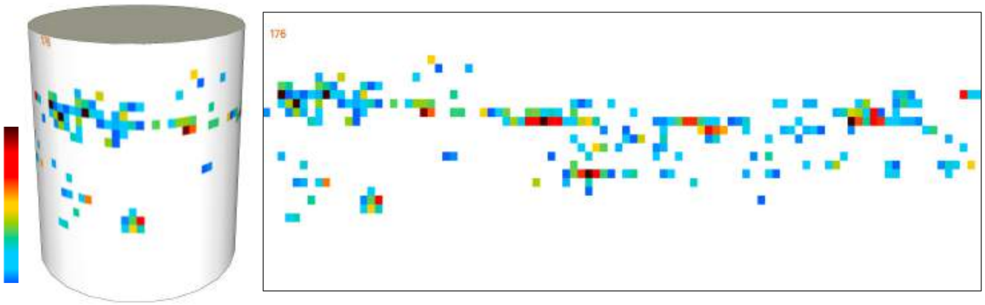


Figure 6.5: The PLI and PPI tiled visual design. (left) A protein interaction space in 3D space-time. The image based on Chapter 4. (right) The corresponding projected, tiled space in the hybrid view. Color is mapped to the frequency of interaction. The image is produced with this tool.

cylindrical shape. The cylinder, in 3D, is responsible for capturing and storing PLI in its tiles. In the hybrid view, the interaction space is projected onto a 2D plane to reduce occlusion. The projected space can be used to obtain an overview and identify areas of interest. It also provides a continuous map of the interaction space. See Figure 6.5.

Protein Abstraction In the 3D view, I use two types of abstraction, spheres and cylinders. They are used to visualize proteins and their interaction space respectively. A protein is depicted by a group of spheres. Each sphere represents a residue or particle. A protein interaction space is abstracted and visualized by a cylindrical shape (Figure 6.5). In the hybrid 2D view, proteins are abstracted via glyphs. The initial seeding point of each protein is used to indicate the location of the relevant abstract protein in the 2D view. The abstract protein in the 2D view is considered as a hybrid because PPI is represented in two different ways, one way using tiles, the other, using glyphs. Using two modes of visual design: a rasterized, tiled space, and a discrete glyph-based presentation makes them visually separable (exploiting focus and context rendering as well). However, laying the glyphs on top of the tiled space also enables the user to study their behavior in a unified, complementary context. The hybrid 2D view is used to visualize PLI and PPI both together and separately. The next two subsections discuss the visualization of the two types of interaction.

Interactive Visualization of PLI A PLI occurs when a lipid particle interacts with a protein particle. The frequency and the duration of the PLI are stored in the tiles of the protein interac-

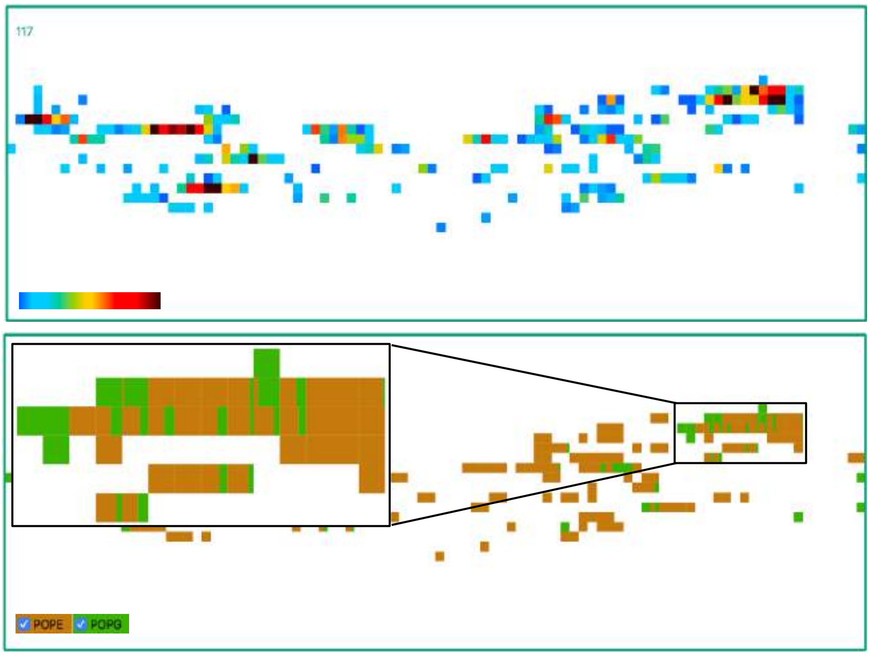


Figure 6.6: 2D images of the same protein focusing on PLI. (top) the tiles' color is mapped to the number of interactions. (bottom) the color of tiles is mapped to molecule types. See Section 6.3.2.

tion space (in the 3D space-time Figure 6.5 - left). In the hybrid 2D view, the PLI is visualized by a rectangle placed on the 2D abstraction of the corresponding proteins. See Figure 6.6. Each rectangle represents an unfolded view of the protein interaction space. The color of the tiles can be mapped to either the frequency of interaction with the relevant tile or mapped to the donor molecule type (POPE type and POPG type). See Figure 6.6. By mapping the tiles' color to the frequency of interaction (Figure 6.6 - top), the user can easily identify the accessible areas of the protein. The other representation, mapping the tiles' color to the donor type (Figure 6.6 - bottom), enables the user visually to understand which type has more interactions with a protein in the areas of interest. Behavior that underlies an area of interest, such as the frequency of interaction between an amino acid and the donor particles, can be revealed by clicking on the desired tile.

Interactive Visualization of PPI A PPI occurs when a protein particle interacts with a particle from another protein. PPI is a special case of molecular interaction. In the case of PLI, the two interacting molecules (I.e the lipid and protein molecule) are not studied to form clusters. The PPI, on the other hand, involves two aspects: the interaction between the particles of a protein pair, and the spatio-temporal clustering of protein pairs and their properties. Similar to PLI, the tiled visual design can be used to visualize behavior of the first aspect. The PPI tiles also can be mapped either to the frequency of PPI, or the type of the donor. The second aspect is associated with a number of properties (rotation, radius and distance to seed position of a protein). These protein attributes are visualized via glyphs. Figure 6.7 illustrates the visual design of the glyph properties. Arcs surrounding the glyph are used to visualize the amount of rotation. See Figure 6.7. I use three separate arc segments to present three aspects of rotation. 1) The amount of rotation before a protein joins a cluster (inner-most arc). 2) The amount of rotation after it joins a cluster (middle arc). 3) The amount of rotation of the parent cluster (outer arc). This design enables the user to compare the collective rotation of the cluster with the proteins that belong to it. The rotation here focuses on an axis at the center of each protein. The axis is orthogonal to the membrane layer. Thus, each axis pierces (conceptually) the membrane. The changes to the protein's radius size are reflected by a circle-glyph (Figure 6.7 - d). The cluster membership of each protein in a cluster is indicated using an outline that encompasses the proteins (Figure 6.7 - e). The portion of the distance between protein pair and their original seed point in 3D is visualized by an indicator placed on the connector-glyph (Figure 6.7 - f).

Focus and Context The initial design of the hybrid view enables the user to toggle the visualization of either tiled space or PPI behavior individually (Figure 6.4 - a and b). The focus and context option is introduced to enable the user to couple and decouple the visualization of tiled space and PPI behavior (Figure 6.4 - c and d). The user can visualize the two aspects at the same time, one in focus while the other in context. This feature is useful in studying the relation between the two aspects and understanding how PLI and PPI influence protein behavior.

6.3.3 Interactive Visualization of Protein Clusters

In the 4D view, a protein initially is depicted by cylindrical shape. The cylinder is color-mapped to the size of the cluster it belongs to. See Figure 6.1 - a. In the 3D view, the position of a protein changes over time. Therefore, a cluster can be easily identified when I find a group of cylinders that have the same color and are in close proximity. In the hybrid view, each protein is visualized via a glyph and is projected onto a fixed position in the 2D space (the initial

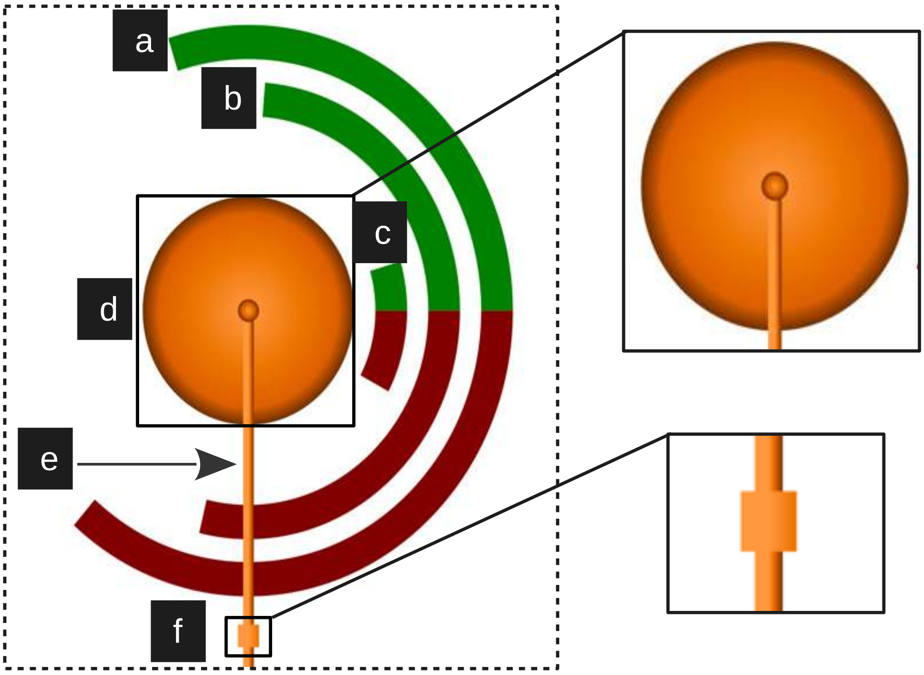


Figure 6.7: Visualization of protein properties with respect to PPI and rotation. Glyphs show three rotational aspects. The green and red arcs show the positive and negative *gross* rotation. a) The outer arc represents the rotation of the entire cluster. b) The middle arc represents the amount of rotation after joining the cluster. c) The closest arc to the protein ellipse represents the amount of rotation of a protein before forming a cluster. d) A dynamic glyph represents the change in the radius of a protein. e) A glyph connects protein pair in a cluster. f) A dynamic indicator shows how far a protein is from its original seed point in 3D relative to the pair. See Section 6.3.2.

seeding point in the simulation). To identify protein clusters in the hybrid 2D view I utilized three techniques. Firstly, a group of proteins that form a cluster are bounded in such a way that they represent one object (a cluster). Secondly, for each protein pair in a cluster, an edge is used to connect them. The edge features a dynamic indicator that shows how far a protein is from its original seed point in 3D relative to the pair. Thirdly, glyphs that represent a protein are color mapped based on the size of the cluster (Figure 6.1 - d).

Interactive Timeline The timeline is responsible for tracking the history of clusters. The x-axis represents the time and the y-axis represents each cluster. A simple cluster consists of two proteins, while a more complex cluster is formed by the combination of three or more proteins. The timeline shows the evolution of the clusters over time and the size of a cluster is mapped to color. See Figure 6.1 - bottom. The longer a cluster lifespan, the stronger the stability of the cluster. The timeline is linked to the other views in such a way the user can click on a cluster in the timeline, to move the simulation pointer to the timestep at which the cluster is formed. Also, the user can hover-over on the timeline to highlight the corresponding cluster in the hybrid view.

6.3.4 Detail-on-Demand

I provide the user with a set of interactive graphs in the form of line-charts. See Figure 6.1 - top right. The graphs are linked to the other views. The user can click on any interesting pattern in a graph to investigate them in the other views. The graphs also feature a time indicator pointing to the current simulation timestep. The graphs can be displayed by clicking on either a cluster or an individual protein. The cluster graph displays the rotation and translation of the cluster and the proteins that belong to it (Figure 6.1 -e and f). The protein graph displays the frequency of three types of interaction over time (Figure 6.1 -g). PPI, POPE type protein interaction, and POPG type protein interaction. The framework also enables the user to analyze the PLI and PPI in more detail by revealing the underlying interaction details. i.e. the user can click on areas of interest to reveal details concerning the particles or residues that are involved in the interaction. The visualization classifies the interaction based on the interacting particle pair and the frequency of interaction between them. See the accompanying video.

6.4 Case Studies and Domain Expert Feedback

In this section, I provide three case studies and the domain expert feedback. This work has been developed in close collaboration with Dr. Matthieu Chavent. Dr. Chavent has more than 15 years of experience in the field of visualizing and analyzing molecular dynamics systems. In additional to his Ph.D. in computational biology, Dr. Chavent held the following positions: a post-doc (2009-2010) at CNRS laboratory IBPC (Paris France), post-doc 2010-2011) at CEA (Arpajon, France), and research associate position (2011-2017) at the Structural Bioinformatics & Computational Biochemistry group at the University of Oxford (UK). Since 2017, he is a CNRS researcher at the Institute of Pharmacology and Structural Biology (IPBS), Toulouse, France. He uses multiscale modeling and NMR experiments to study lipids, membrane proteins, and their interactions. He is especially interested in deciphering how lipids can modify the physical properties of membranes and the functions of membrane proteins. All of the fea-

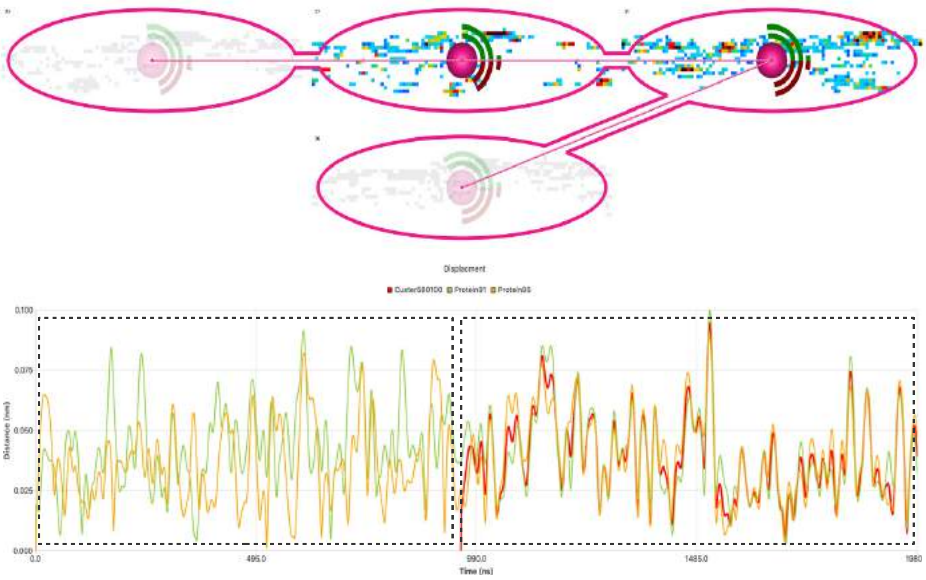


Figure 6.8: Two snapshots of the visualization result illustrate the two observations. (top image) Proteins at the extremity of a cluster have interaction less than the average frequency. PPI less than the average is rendered as context. (bottom image) A cluster unifies the translation behavior of its members. (left dashed box) The translation of proteins before forming the cluster. (right dashed box) The translation of proteins after forming the cluster. The red line display

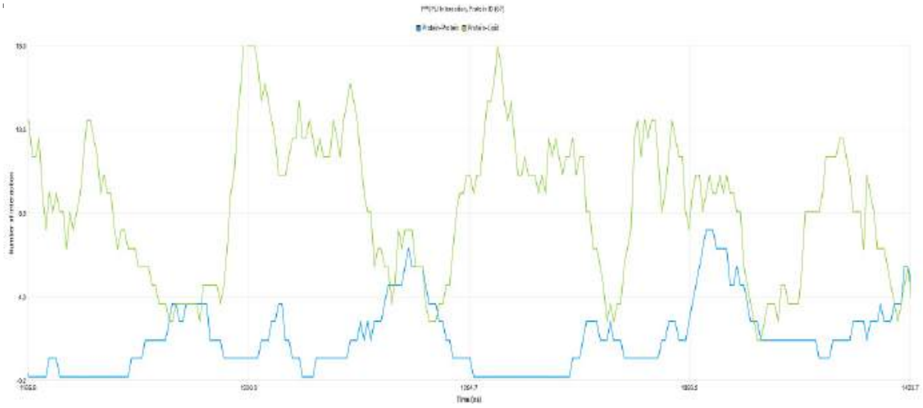


Figure 6.9: Image shows an inverse relationship between the PLI and PPI. The blue line represents the frequency of PPI overtime. The green line represents the frequency of PLI over time.

tures were guided by and developed together with him. Since 2015, I conducted 11 meetings (60-90 minutes each), demonstrating and receiving feedback about the software during its development stages. Each meeting was video recorded.

Case study hypothesis: If a protein is interacting with lipids then it is interacting less

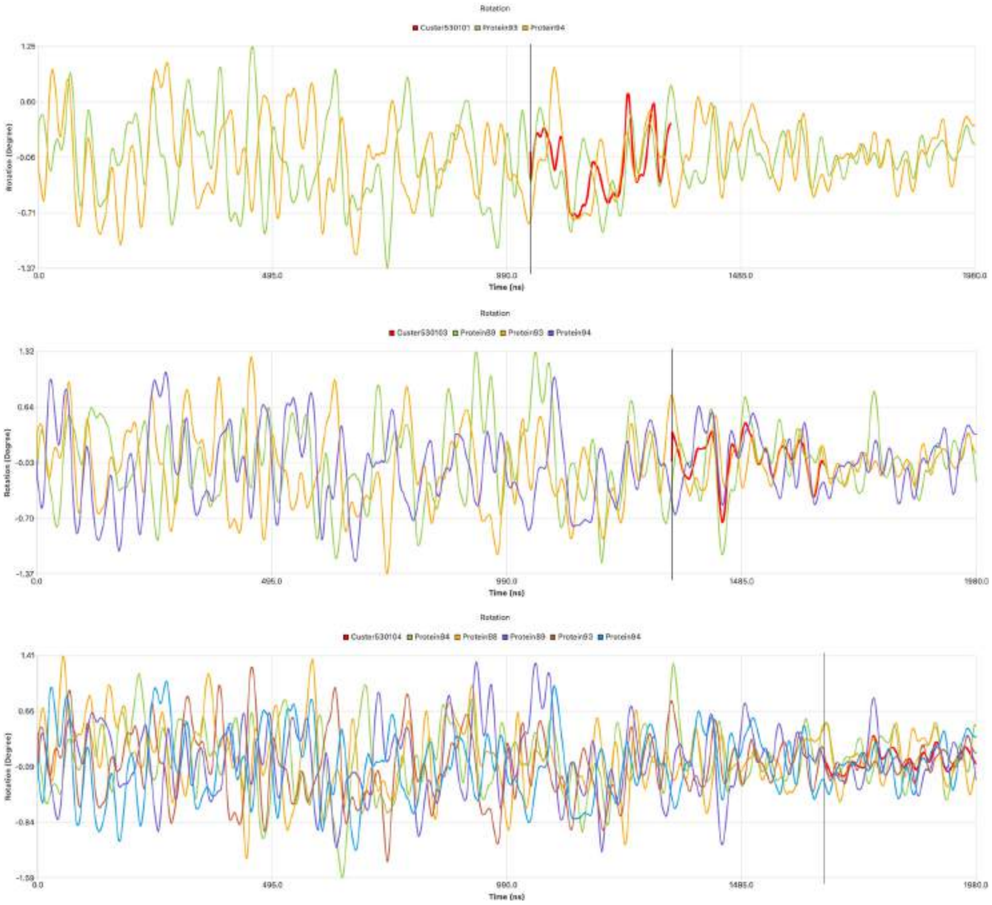


Figure 6.10: Three line-charts illustrate the change in the rotation of a cluster during its evolution. The amount of cluster rotation is displayed by the thick, red lines. (top) A line-chart displays the rotation of the cluster and its members (the first formation of the cluster). The cluster's accumulative rotation ranges between 0.70 to -0.75 radian. (middle) The second formation of the cluster (three proteins). The line-chart shows a small decrease in the counterclockwise rotation. (bottom) The cluster rotation after the cluster formed by four proteins. The line-chart shows a decrease by approximately 0.50% in both the clockwise and counterclockwise rotation.

with other proteins and vice versa. This hypothesis can be tested by the hybrid view and the details-on-demand view. I utilize the protein interaction graph to examine the hypothesis. I invoke the graph by selecting a protein that is involved in a cluster. See Figure 6.1 (g). The graph displays the PLI and PPI of the selected protein. From the graph, it can often be seen that the peak of PPI is associated with the lowest point of PLI. The visualization using the detailed interaction line-chart view supports the hypothesis. See Figure 6.9. Furthermore, by examining

the hybrid view I observe that the proteins at the extremity exhibit less than average interaction. See Figure 6.8 (top image). However, in exceptional cases, proteins at the extremity of a cluster might exhibit greater than average interaction. For example, consider two clusters that are well-established at the beginning of the simulation. Then afterward, a third cluster is formed. If the third cluster forms a new, larger cluster by linking the two clusters together, it is likely to have extreme proteins exhibiting greater than average interaction. In order to understand this further, I need to investigate the history of the cluster and the proteins in question with respect to PPI. The clustering timeline view plays a significant role in understanding the behavior of these two clusters and helps us to investigate this issue. See the accompanying video for further visualization results at:
https://github.com/NaifAlhar6i/Chapter6

Case study Hypothesis: The larger the size of a cluster, the less the rotation and translation. I examined clusters of varying sizes and use the details-on-demand feature to compare the translation and rotation of each. From the line-chart, I found that the amount of rotation of individual proteins decreases immediately after joining a cluster. See Figure 6.10. Also, I find that the size of a cluster plays a significant role in reducing the amount of rotation, which provides strong evidence supporting the rotation hypothesis. I do not witness a considerable change in the translation speed. Yet, I observe that a cluster unifies the translation behavior of its members in such a way the proteins behave similarly after entering a cluster. See Figure 6.8 (bottom image).
See the accompanying video for further visualization results at:
https://github.com/NaifAlhar6i/Chapter6

Domain Expert Feedback:
The following feedback was produced directly by the domain expert: “Visualizing easily both PLI and PPI is especially novel as, currently, no other molecular viewer are designed to perform this type of analysis concurrently. Going back and forth in between the different views help to better understand the formation of the protein clusters and if lipids surrounding the proteins may modulate such cluster formation. The 2D tiled space is especially useful to quickly compare in a large systems interactions in between proteins and in between protein and lipids and see how these interactions evolve in function of the time.”

6.5 Conclusion

In this Chapter, I proposed a novel PLI and PPI visualization framework. The framework utilizes four vital, visual designs that successfully enable the user to study a time-dependent membrane simulation. The framework employs a complementary abstraction and space projection to address a number of visualization challenges. Overall, the Chapter proposes a novel hybrid view to enable the user to study PLI and PPI, and the behavior of the PPI and clusters. A details-on-demand view is used to provide the user with desired information about proteins, clusters, and their behavior.

Chapter 7

Real-Time Rendering of Molecular Dynamics Simulation Data

Contents

”There are no such things as applied sciences, only applications of science.” -Louis Pasteur [1]

[1] Louis Pasteur (1822-1895) was a French biologist, microbiologist and chemist renowned for his discoveries of the principles of vaccination, microbial fermentation and pasteurization.

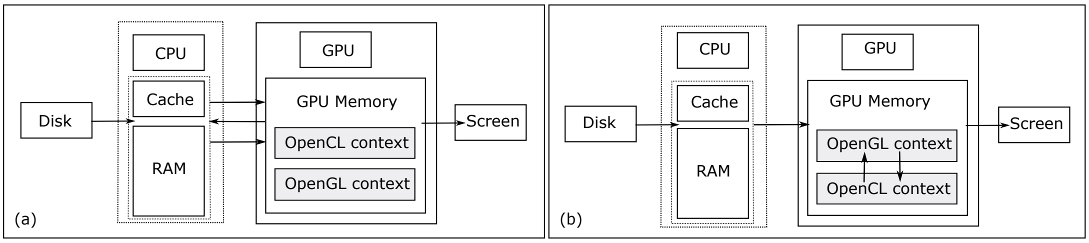


Figure 7.1: A traditional approach vs. an advanced approach. The arrows indicate the flow of the data throughout the visualization pipeline. a) the traditional approach requires copying the data four times per computation. The first copy is from disk to the RAM, and the remaining to exchange the data between the CPU and GPU. b) the advanced approach utilizes a mapping technique and OpenGL interoperability. The data is copied only twice: from disk to RAM and from RAM to the GPU.

This Chapter addresses some performance challenges that were posed by the calculation of the MD trajectories' properties described in Chapter 3. In this chapter, I introduced an efficient approach that harnesses GPU to boost the visualization. The proposed approach was used in Chapters 4-6.

7.1 Introduction and Motivation

In computational biology, Molecular Dynamics Simulation (MD) is used to simulate the dynamics of lipids and proteins. MD is capable of producing a high volume of MD data resulting from simulating the motion and interaction of billions or even trillions of particles. The MD data can be investigated utilizing various visualization techniques. The ever-increasing size of MD output poses a number of challenges and requires data visualization scientists employing new technologies and techniques to address these challenges. Thankfully, advances in commodity Computer Graphics hardware including Graphics Processing Unit (GPU) technology result in a significant acceleration of graphics applications including scientific visualization. Commodity graphics hardware is capable of processing millions of textured triangles per second [195]. However, even though the new hardware is designed to ensure high rendering quality and performance, the size and nature of the data, the required processing per frame, and the rendering approaches affect the final rendering frame rate and pose three main challenges, the I/O, the computation, and the rendering challenge respectively.

In terms of the size of data, which forms the basis of the first challenge, I consider the

data to be big data when it does not fit into the RAM. Data not held in RAM means it requires more frequent file I/O access which is very expensive compared to RAM access and may turn to a rendering bottleneck. The second challenge is posed by the computation requirements. Regardless of the complexity of the computation, the computation challenge is tightly coupled with the size of the data and the methods that are used to handle it. The third challenge is tightly related to the (CPU versus GPU) computation location, and the data exchange approach by which data is passed between the computational and the rendering stages. In terms of location, the computation might be performed on the CPU or on the GPU. In both cases, the data transfer method plays an essential role in determining the visualization performance.
A real-time visualization can be achieved by increasing the rendering frame rate. The higher this number, the better the user's perception of the fluidity of the scene. An optimal frame rate is around 30 fps, as the human visual system is only perceives up to about 25 images per second (called the persistence of vision) [196]. If such high frame rates are achieved, the rendering is considered real-time [28].

This article addresses the big data I/O challenge, computation challenge and fast rendering by introducing a memory mapping technique and an advanced GPU approach utilizing the OpenCL interoperability feature. The simulation is performed previously by GROMACS [32]. I choose OpenCL because it is free, cross-platform, well documented and fast. See figure 7.2. The article is targeted at developers who would like to render MD data and guides the user on how to achieve a real-time rendering. My contributions are:

- A memory mapping technique to enhance the I/O performance.

This article focuses on *OpenGL 4.3* and *OpenCL 1.1*, and it does not cover data visualization techniques such as filtering, sampling, Level of Detail (LoD) etc, as these techniques are beyond the scope.

The rest of the Chapter is organized as follows: in section 7.2, I provide an overview of related work. Section 7.3 describes the requirements of my real-time molecular dynamics rendering. In section 7.4 I describe the challenges and their solution, and provide a comparison of my solution with a traditional solution.

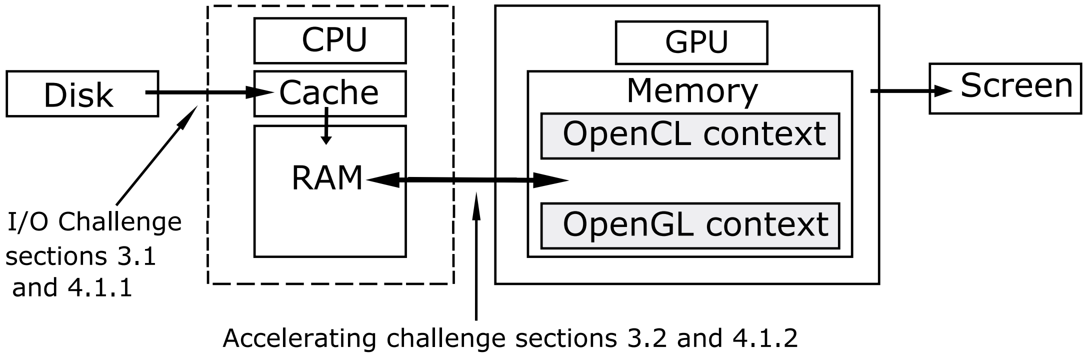

CPU
Disk
Cache
RAM
I/O Challenge sections 3.1 and 4.1.1
GPU
Memory
OpenCL context
OpenGL context
Screen
Accelerating challenge sections 3.2 and 4.1.2

Figure 7.2: An overview of the challenges throughout the visualization pipeline.

7.2 Background

GPU technology plays a significant role in enhancing the performance of MD data computation and visualization. Molecular modeling algorithms that exploit the GPU have been largely successfully compared to alogorithms that use CPU alone. See Stone *et al* [55] for an overview. Recent developments utilizing GPUs address performance limitations and herald the next generation of molecular visualization solutions according to Chavent *et al* [28].

In general, a number of researches have been published illustrating GPU technology and how to code GPU-based programs. Stone *et al.* [197] provide an overview of the key architectural features of recent microprocessor designs and describe the programming model and abstractions provided by OpenCL. Shreiner *et al* [198] and Wright *et al.* [199] can be considered OpenGL specification-based references. Shreiner *et al* and Wright *et al*, are accompanied by a collection of C++ code examples. Shreiner *et al.* [198] focus on the latest methods and techniques for OpenGL 4.3 application development. They describe every stage of the programmable rendering pipeline and cover shading techniques found in the OpenGL Shading Language (GLSL) including the compute shader which is introduced in OpenGL 4.3. The compute shader runs in a separate stage of the GPU and allows an application to make use of the power of the GPU for general purpose work that may or may not be related to graphics. Wright *et al.* [199] is designed for readers who are learning computer graphics through OpenGL and readers who may already be familiar with graphics but want to learn about OpenGL, however, readers are required to understand computer programming in C++. In addition to the essential OpenGL concepts, each book involves a number of various topics that cover some advanced concepts including high-performance rendering techniques and GPU analysis and OpenGL debugging tools. These researches cover the compute shader which can be used to fulfil computational requirements. There are a number of efficient frameworks that are essentially designed to harness GPU to perform computational tasks.

Weiskopf [200] consists of 5 Chapters and it is designed to be a starting point to understand the GPU-based visualization. The main parts of the book focus on efficient GPU-based visualization techniques for interactive exploration of 3D scalar and vector fields, and for enhancing visual perception of non-photorealistic rendering. A number of useful topics are discussed throughout the book including parallelization on clusters with several GPUs, adaptive rendering methods and multi-resolution methods.

Gaster *et al.* [201] guide readers, by example, on how to program heterogeneous environments with OpenCL and define the concepts that the readers need to understand before starting to program any heterogeneous system. They concentrate on illustrating the OpenCL framework in a disconnected context. However, Gaster *et al.* [201] briefly discuss the concept of sharing a context between OpenCL and OpenGL in some of their examples. Munshi *et al.* [202] explain how OpenCL 1.1 can be used to express a wide range of parallel algorithms. They cover the entire OpenCL 1.1 specification including advanced OpenCL features i.e. OpenCL interoperation which enables developers to access and manipulate OpenGL buffers from a shared context. Scarpino [203] involves 16 Chapters illustrating the OpenCL language by example. The first 10 Chapters explore the OpenCL language following by four Chapters that show how OpenCL can be used to perform large-scale tasks. The last two Chapters focus on the OpenCL interoperation and show how OpenCL can be used to accelerate OpenGL applications. I utilize the OpenCL interoperation approach in the same manner used in [202, 203], however, I also integrate the *glMapBuffer()* and the MMPs in the solution to enhance the final rendering performance.

Schatz *et al.* [204] present a novel method that enables users to explore one trillion particles. Their method is based on an advanced focus and context technique and a camera space visualization i.e. only particles that surround the camera are selected. They utilize a dual-GPU configuration that splits the workload between the GPUs based on the type of data. Schatz *et al.* decode and visualize different attributes of the MD data and allow users to explore the data-set by moving the camera throughout the scene. However, even though they apply their method to a dynamic data-set, the current version of the method handles the data-set as a volume.

Hrabcak and Masserann [205] illustrate the concept of asynchronous buffer transfers to enhance the performance of two way data traffic i.e uploading and downloading data to and from the graphics card. The proposed approach focuses on optimizing data traffic per frame by utilizing two techniques: i) a map buffer, and ii) a swapping buffer technique. This approach is useful for applications that render a static data-set, or applications that separate computation from rendering by performing computations on the CPU. I utilize the map buffer technique to stream data from the CPU to the GPU.

Movania and Feng [206] describe a method for implementing and visualizing real-time deformation. They utilize a modern GPU transform feedback mechanism. The transform feedback mechanism enables developers to feed an OpenGL buffer, the so-called transform feedback buffer (TFBB), via either the vertex or geometry shader which means the calculation must be done in the vertex (or geometry) shader. One of the most significant advantages of the transform feedback is that the TFBB content can be rendered directly from the buffer which eliminates unnecessary traffic between the CPU and GPU. However, this approach is not feasible for intensive computations or computing big data and the vertex and geometry shaders tend to be used for light computations as the vertex shader and the geometry shader are limited to a single vertex and primitive respectively.

In this Chapter, I focus on the performance aspect of real-time MD rendering. I cover a number of techniques that can enhance the performance throughout the visualization pipeline. These techniques are Memory-Mapped Files (MMFs), a Mapped buffer, and integrating OpenGL and OpenCL by allocating a shared context.

7.3 Real-Time Visualization Requirements

Scientific visualization research utilizes a well-defined pipeline to create the final representation of the input data-set. For simplicity, I use a basic definition for the visualization pipeline. I consider the visualization pipeline consisting of three main stages (Moreland [207]): i) the data I/O stage, ii) the computation stage, and iii) the rendering stage.

In this section, I briefly describe my data set, a programming environment for developing a solution, and three popular frameworks that can be utilized through the visualization pipeline to achieve the real-time rendering.

Data Description My data-set represents biological dynamics of lipids and proteins at high resolution. The system's dimensions are 116.01860 x 116.01860 x 10.13590 nano meters (x, y,

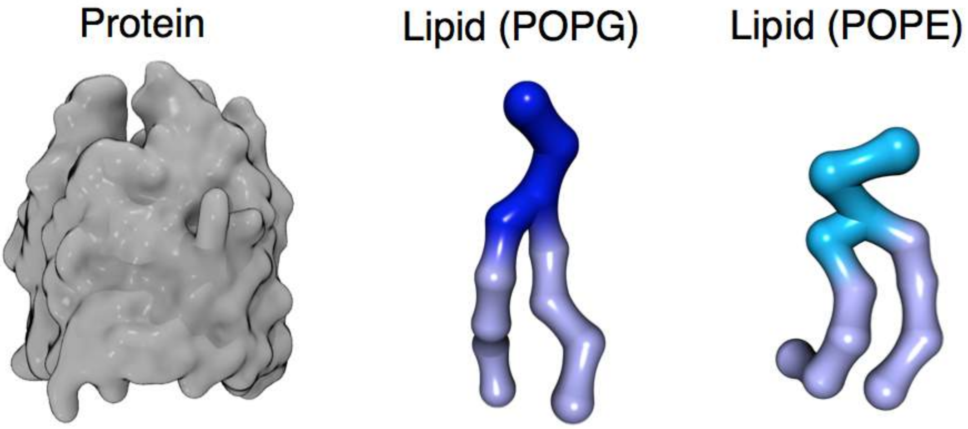


Figure 7.3: Structure of molecules: Protein, POPG and POPE types (left to right). The hallow of protein particles is used to construct the protein surface. An advance ball and stick is used to represent the POPG and POPE types. The Protein image is generated with VMD [25], and the Lipid type images are generated with UnityMol [27] based on the hyperballs representation [28].

and z respectively) and the individual trajectories reflect the evolution of 336,260 particles over 1,981 nanoseconds (ns). The system consists of three molecule types: one protein type, and two lipid types (POPE type and POPG type). The protein type involves 256 protein molecules while the lipid POPE and the POPG type consist of 14,354 and 4,738 molecules respectively. The hierarchy of the structures is described below:

- The protein structure consists of 256 proteins. Each protein has 171 residues. Each residue consists of 1 to 3 particles. A single protein consists of 344 particles which results in (256x344) 88,064 particles in total.

- The lipid structure consists of 19,092 lipids. Each lipid contains 3 groups: i) a head group (2 particles), ii) a tail group (5 particles), and iii) a second tail group (6 particles). Each lipid molecule has 13 particles which results in (19,092x13) 248,196 lipid particles in total.

Figure 7.3 provides a depiction of the structure of a protein, POPG type and POPE type. In terms of the size of the data, the data-set contains more than 666 million vertices that occupy 8 Gigabytes of memory.

My requirement is to visualize the interaction between lipids and proteins on-the-fly. A protein's body is represented by a sphere glyphs while shaded spheres are used to represent lipid particles. The interaction between lipids and proteins is represented by color-mapping the interacting particles (Figure 7.4).

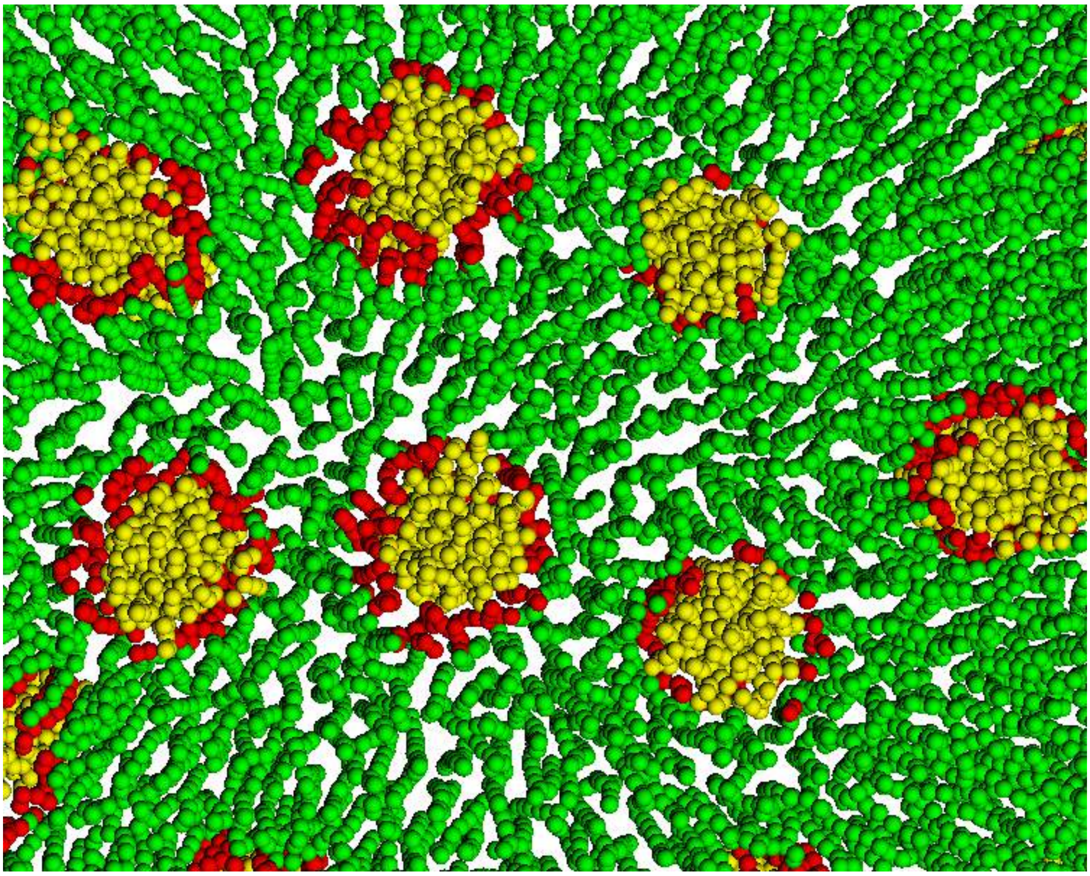


Figure 7.4: Protein-Lipid interaction. The protein particles are represented by gold spheres and the lipid particles are represented in green. The red spheres represent the lipid particles that interact with the protein particles within 0.6 angstrom. The image is generated with the accommodated example code.

Development Framework There are a number of a powerful programming languages that can be utilized to implement the visualization pipeline. In this Chapter, the implementation of these stages is done in C++ utilizing the Qt Framework (v. 5.7) [208]. Qt is chosen for a number of reasons:

- Qt is cross-platform.

- Qt provides developers with a free development GUI called Qt Creator [209].
- In addition to the QOpenGlWidget class, Qt provides developers with a set of QOpenGL* classes that interface with most of the OpenGL objects including the shader.
- Qt has up-to-date on-line documentation.
- Qt for application development is available under two licence agreements: commercial and open source licenses.

The code is implemented following Laramee's [210] concise coding conventions which considerably enhance the code readability and result in well organized source code. The full source code and the supplementary materials can be downloaded from the following **URL** `https://github.com/NaifAlhar6i/Chapter7`

7.3.1 Data Processing and Multi-Operating System Configuration

In this sub-section, I describe the molecular dynamics simulation files, and illustrate the necessary confirmation to run the code on different operating systems. I process two types of files (gro and xtc). The first one, GRO, is used to describe the molecules structure and the XTC file stores the actual dataset.

gro Files contain a molecular structure that describe a trajectory by simply concatenating files [211]. However, in my example the GRO file is just used to obtain the molecule structure. A sample is included below:

```
SELF-ASSEMBLY
336260
   1ALA CA 1 2.892 3.409 2.771
   2PRO CA 2 3.076 3.738 2.707
   2PRO CB 3 3.033 3.900 2.767
   3LYS CA 4 3.344 3.706 2.460
   3LYS CB 5 3.254 3.391 2.339
   3LYS CG 6 3.231 3.204 2.403
   4ASP CA 7 3.706 3.775 2.437
   4ASP CB 8 3.778 3.727 2.211
   ..
   ..
   ..
   37698POPG C4B75914 94.693 104.448 6.085
   37698POPG C5B75915 94.632 104.578 5.616
116.01860 116.01860 10.13590
```

Lines contain the following information (top to bottom):

- title string (free format string, optional time in ps after ’t=’)
- number of atoms (free format integer)
- one line for each atom (fixed format, see below)
- box vectors (free format, space separated reals), values: v1(x) v2(y) v3(z) v1(y) v1(z) v2(x) v2(z) v3(x) v3(y), the last 6 values may be omitted (they will be set to zero).

This format is fixed, ie. all columns are in a fixed position. Columns contain the following information (from left to right):

- residue number (5 positions, integer)
- residue name (5 characters)
- atom name (5 characters)
- atom number (5 positions, integer)
- position (in nm, x y z in 3 columns, each 8 positions with 3 decimal places)
- velocity (in nm/ps (or km/s), x y z in 3 columns, each 8 positions with 4 decimal places)

I provide a class *GROManager* contains a method *fetchStructure()* that derives the molecules structure from the GRO file.

xtc Files is a portable format for trajectories [212]. It uses the xdr routines for writing and reading data which was created for the Unix NFS system. The trajectories are written using a reduced precision algorithm which works in the following way: the coordinates (in Nanometres) are multiplied by a scale factor, typically 1000, so that coordinates are in Picometers.

These are rounded to integer values. Then several other customization are performed, for instance making use of the fact that atoms close in sequence are usually close in space too (e.g. a water molecule). To this end, the xdr library is extended with a special routine to write 3-D float coordinates.

All the data is stored using calls to xdr routines and can be read using the same xdr routines. I utilize the XDR liblary from GROMACS.

The variable described below are used to obtained data from the XTC file

int magic a magic number, for the current file version its value is 1995.

int natoms the number of atoms in the trajectory.

int time step the simulation step.

float real time in the simulation.

float box[3][3] the computational box which is stored as a set of three basis vectors, to allow for triclinic Periodic boundary condition (PBC).

For a rectangular box the box edges are stored on the diagonal of the matrix.

3dfcoord x[natoms] the coordinates themselves stored in reduced precision. Please note that when the number of atoms is smaller than 9 no reduced precision is used.

XTC Library The XTC library is an open source library under the lesser GNU public license [159]. It is design to help developers reading and writing xtc, edr and trr files. I utilized this library to read my xtc file. Reading data from XTC file requires three steps: 1) opening the xtc file, 2) looping through the xtc file, and 3) closing the xtc file. The xtc file can be opened via *xdrfile_open()*. This function has two parameters *FileName* and *AccessType*. Then *read_xtc()* can be used inside a logic loop to read frames until no frame found in the file. Finally, the xtc file must be closed via *xdrfile_close()*. I provide a class *XTCManager* that interfaces the xtc file reading functionality. The *XTCManager* class has only one main function *FetchFrames(const unsigned int start, const unsigned int end, short stride=0)*. The start and end parameters hold the first frame and the last frame to be read respectively, and the stride parameter holds the number of frames that I must skip. The stride parameter can be used to get a time-sampled data-set.

7.3.1.1 Context Properties Configuration for OpenGL Interoperability

In order to use OpenGL interoperability, the OpenCL context properties must be configured with respect to the operating system. The *cl_context_properties* is a data structure and it has three fields. The first and the second fields specify the the property name to be configured and its value respectively while the third field must be set to 0.

Configure OpenCL shared context on Apple Platform

On Mac OS, the cl_context_properties structure requires only one property [203, 202]: CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE. However, the value associated with this property must have the data type CGLShareGroupObj, and it can be acquired via the function *CGLGetShareGroup()*. This function requires a *CGLContextObj* structure, which can be obtained by calling *CGLGetCurrentContext()*. The following code shows how these functions work together:

```
CGLContextObj glContext = CGLGetCurrentContext();
CGLShareGroupObj shGroup = CGLGetShareGroup(glContext);
cl_context_properties prop[] = {
  CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
  (cl_context_properties)shGroup,
  0};
```

Configure OpenCL shared context on Linux Platform

The configuration of the cl context property on Linux requires three structures: CL_GL_CONTEXT_KHR, CL_GLX_DISPLAY_KHR, and CL_CONTEXT_PLATFORM. There are three functions that can be used to acquire the associated values with these properties *glXGetCurrentContext()* for the first property CL_GL_CONTEXT_KHR and *glXGetCurrentDisplay()* for the second property CL_GLX_DISPLAY_KHR and *clGetPlatformIDs()* to acquire the platform id.

```
clGetPlatformIDs( 1, &cpPlatform, NULL );
 cl_context_properties prop[] = {
   CL_GL_CONTEXT_KHR,
   (cl_context_properties)glXGetCurrentContext(),
   CL_GLX_DISPLAY_KHR,
   (cl_context_properties)glXGetCurrentDisplay(),
   CL_CONTEXT_PLATFORM,
   (cl_context_properties)cpPlatform,
   0};
```

Configure OpenCL shared context on Windows Platform

On Windows platform the CL context properties configuration requires setting three properties: CL_GL_CONTEXT_KHR, CL_WGL_HDC_KHR, and CL_CONTEXT_PLATFORM. Similar to Linux, on Windows there are three functions that can be used to acquire the associated values with these properties *wglGetCurrentContext()* for the first property CL_GL_CONTEXT_KHR and *wglGetCurrentDC()* for the second property CL_WGL_HDC_KHR and *clGetPlatformIDs()* to acquire the platform ID.

```
clGetPlatformIDs( 1, &cpPlatform, NULL );
```

The following code shows how to configure the CL context properties for Windows:

```
 cl_context_properties prop[] = {
  CL_GL_CONTEXT_KHR,
  (cl_context_properties)wglGetCurrentContext(),
  CL_WGL_HDC_KHR,
  (cl_context_properties)wglGetCurrentDC(),
  CL_CONTEXT_PLATFORM,
  (cl_context_properties)cpPlatform,
 0};
```

Creating a Shared Context Once the CL context properties have been set, *clCreateContext()* function can be used to create the shared context. This function can create a context with one or more devices. It utilizes the CL context properties that I have configured to create a context with the selected device. The following code shows how to create a context with one device:

```
 sharedGPUContext = clCreateContext(prop, 1, &ourDevices, NULL, NULL, &erNum);
```

Creating OpenCL memory object From OpenGL Buffer If the shared OpenCL context was successfully created then I can create OpenCL memory objects by invoking *clCreateFromGLBuffer()*. This function requires the OpenCL context to be created from an OpenGL one and it requires the OpenGL buffer to be exist before the function is called. The following

example shows how to achieve this goal via *clCreateFromGLBuffer()*:
cl_mem sharedCLGLBuffer = clCreateFromGLBuffer(sharedContext, CL_MEM_WRITE_ONLY, glBufferID, &errCode);
The function returns a pointer to the OpenCL memory object stored in *sharedCLGLBuffer*. The first parameter of the function is a shared OpenCL context. The second parameter is a flag to specify the access permission (CL_MEM_READ_ONLY, CL_MEM_WRITE_ONLY and CL_MEM_READ_WRITE). The third parameter specifies the identity of the OpenGL buffer that needs to be set shareable. The last parameter is used to return an appropriate error code.

7.3.2 File I/O and Memory Management

In general, a standard C or C++ library is utilized for the implementation of file I/O through the FILE class and iostream library. In C++, a developer can open a file when instantiating a stream. The stream is utilized to perform the I/O operations and finally the file is closed automatically on destruction of the stream [213]. However, the standard C++ iostream has some limitations which must be taken into account. See section 7.4. Even though Qt addresses this limitation with the QFileDevice class, for learning purposes I decide to implemet the file I/O solution using the Boost library [214]. The Boost library involves a collection of useful libraries [215, 216] that can be used as an alternative to the C++ Standard Template Library (STL). The Boost iostream Library address the STL iostream limitation via the Memory-Mapped Files classes: 1) *mapped_file_params* 2) *mapped_file_source*, 3) *mapped_file_sink* and 4) *mapped_file*. They provide access to memory-mapped files on Windows and the Portable Operating System Interface (POSIX) systems [214]. The *mapped_file_params* class is responsible for the parameters used to open a memory-mapped file. The *mapped_file_source*, *mapped_file_sink* and *mapped_file* classes provide read, write and read-write access to memory-mapped files respectively. I propose these classes to enhance the file I/O performance and reduce the memory management limitations.

7.3.3 GPU Accelerator

The GPU is essentially designed to accelerate the rendering process, however, it is widely used for general purpose computation as well. There are a number of Application Programming Interfaces (APIs), such as CUDA [217], OpenCL [218], OpenGL [219], and the promising graphics and computing API Vulkan [220], that can be utilized to program a commodity GPU.

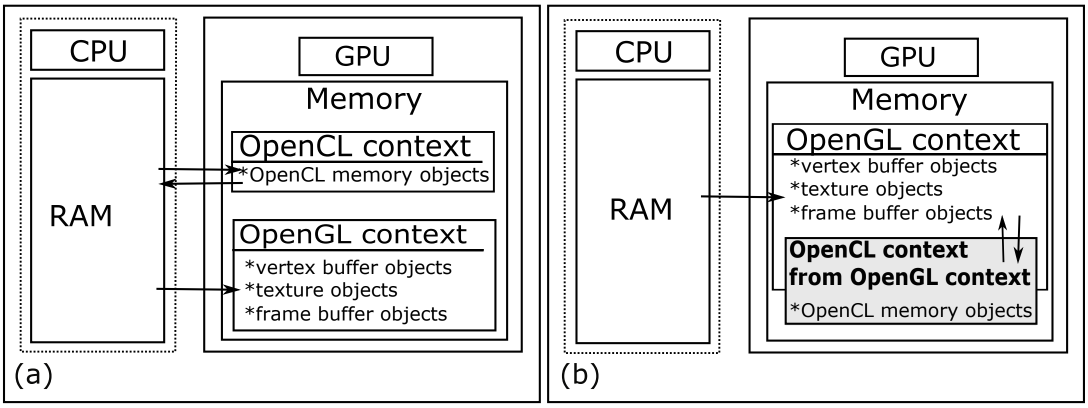


Figure 7.5: OpenCL and OpenGL integrating approaches. a) OpenCL context and OpenGL context are separated. This approach requires exchanging data between CPU and GPU. b) OpenCL creates its own context from an existing OpenGL context. This shared context allows OpenCL to access the shared OpenGL objects in GPU memory. The data is computed and visualized solely on the GPU.

They enable developers to harness GPU parallelism through straightforward C code that runs in thousands or millions of parallel invocations. Even though all these frameworks provide the developer with an interface to communicate with graphics cards, each one is designed for a particular goal. CUDA and OpenCL are designed to accelerate the computation process. However, they can access OpenGL buffers and manipulate buffer content before it is rasterized. OpenGL concentrates on the graphics card rendering pipeline. However, since OpenGL 4.3 developers are able to perform computation tasks utilizing the shader. Vulkan is known as the new generation of OpenGL. It provides a comprehensive computation and rendering framework. My solution relies on OpenGL and the OpenCL API as they are supported by all commodity graphics cards. The proposed solution benefits from the concept of sharing data. See Figure 7.5.

7.4 Rendering Big MD Data: A Proposed Solution

As mentioned, rendering big MD data involves three main challenges. These challenges are distributed throughout the visualization pipeline and are tightly related to each other. My solution utilizes an advanced memory techniques that reduces the data traffic during the visualization and utilizes a GPU feature that enables OpenCL to access OpenGL buffers. See Figure 7.1. In this section, I describe these challenges and illustrate how can they be addressed by my proposed solutions.

Data Size Challenge Even though the size of the data can be reduced before it is sent to graphics card by applying filtering and LoD for example, the essential issue, before hand, is finding an optimal approach to extract the data from big files. The standard C++ library provides developers with many file I/O operations such as open, read, seek, write etc. via the iostream class. However, consider a sequential read of a file on disk using the standard system calls open, read, and write. Each file access requires a system call and disk access which is considered one of the iostream limitations. Alternatively, I can use a virtual memory techniques to treat file I/O as a routine memory access. This approach, known as memory mapping a file, allows a part of the virtual address space to be logically associated with the file. Memory mapping a file is accomplished by mapping a disk block to a page (or pages) in virtual memory [221]. Another limitation of the iostream is that it requires buffering the data from files (the buffer is allocated in RAM then it is passed as a parameter to the read function) before it can be used, whereas memory mapped files provide us with a pointer to the required data in the virtual memory space (Figure 7.6). I decide to use a Memory Mapped file as it has two advantages: first it reduces the number of system calls, second it provides us with a pointer to a process's address space instead of copying all of the data into RAM.

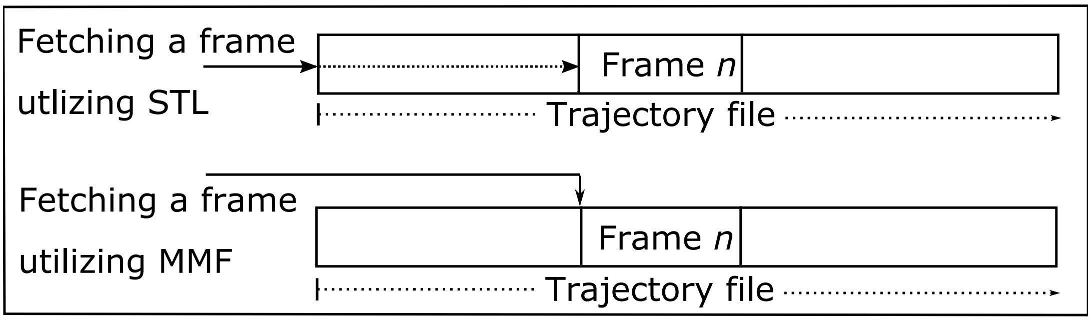


Figure 7.6: Data I/O stage. The process of fetching data from hard-disk utilizing standard C++ library, and fetching data utilizing MMFs. The MMF communicates directly with the OS and provides a random access. STL copies all of the data from file to RAM before it can be used. MMF maps all or part of a file to virtual address space, that can be accessed by CPU, and provides developer with a pointer to that space. Unnecessary data may be skipped rather than read into RAM.

Separate Memory Challenge OpenCL and OpenGL are widely used for heterogeneous computing and graphics development. Both *OpenCL* and *OpenGL* require a valid *context* in order to perform their functionalities. The OpenGL *context* is an object that stores information (e.g. buffer binding) about an OpenGL state. An application may have one or more OpenGL contexts [222]. A *context* can be created by invoking *clCreateContext()* and *glCreateContex()* for *OpenCL* and *OpenGL* respectively which results in two disconnected contexts. Typically the results produced by the OpenCL computation are used as input by OpenGL to render the data. The data computed using OpenCL is located on the GPU and cannot be directly accessed by the OpenGL runtime as they are created in a disconnected mode. As a result, the data has to be explicitly copied from the GPU to the CPU memory. OpenGL can then use the data as an OpenGL buffer for rendering on the GPU. This approach requires data to be transferred between the CPU and GPU each time the application switches from OpenCL compute to OpenGL rendering. Exchanging data between the CPU and GPU adds a significant execution overhead and affects the performance of the application [223]. To handle this issue OpenCL provides an optional extension to share memory objects between OpenCL and OpenGL [202]. The extension enables OpenCL to create a context from an existing OpenGL context. This approach generates a bridge between OpenCL and OpenGL where the OpenCL can access the OpenGL objects in GPU memory. Leveraging the OpenCL-GL shared context results in a 2.2X performance increase [223]. Figure 7.5 illustrates the concept of OpenCL and OpenGL shared context.

7.4.1 Mapped Memory and GPU framework interoperability: Illustration by Example

Decompressing the XTC file MD systems produce trajectory files that contain the atomistic behavior of molecular systems. Each system adopts its own formats for storage of trajectory data [224]. GROMACS [225] defines different formats such as TRR and XTC format. XTC is a lossy compression format [226] and it can be read by special libraries like *xdrfile* [159].

The XTC format is optimized for reducing the size of the XTC files. However, it is not practical to read the trajectories from the XTC file as it requires decompressing the trajectory each time I access the file. The first step in this Chapter is to decompress the trajectory file and save the result in a new file via *xdrfile* and the Boost library. The *xdrfile* library consists of three classes: xdrfile, xdrfile_xtc and xdrfile_trr. The xdrfile_xtc class encapsulates the decompressing processes and provides a straightforward function for reading the trajectory data frame by frame (See the supplementary materials). The output is written to a new file after calculating the decompressed frame size by utilizing the MMF functionality.

7.4.1.1 Reading and Writing Data Using MMF

Logically, I should start by illustrating how to utilize the Boost library to read files, however, reading and writing using Boost is transferable and the concept is applicable for general reading and writing. As mentioned MMFs maps a file, in reading and writing mode, to the virtual memory space. The Boost Library supports the writing mode via two classes: mapped_sink and mapped_file. The former is designed for writing mode only whereas the latter can be utilized to simultaneously read and write a file. If the file is entirely opened, these classes return a pointer of const char type that points to the first byte of the file in virtual memory. However, it is not practical to open the entire file to access a portion of it. Instead of reading the entire file, the required part can be mapped to a virtual memory space, however, the developer is responsible for writing an algorithm that calculates the right position of the required data in the virtual memory space based on the operating system's virtual memory allocation granularity. i.e. the returned pointer always points to a value that must be a multiple of the operating system's virtual memory allocation granularity. The developer might need to handle this issue by adding an offset to the pointer as in Figure 7.7. The implementation of this functionality requires the size of the decompressed frame and the value of the operating system alignment. In my case, each frame represents the position of 336,260 particles in 3D space. As all the frames hold the same amount of data, the frame size is calculated only once. The frame size can be obtained by multiplying the number of particles by two variables: the number of dimensions of the domain space, and the size of the float data type.

$Frame\ Size = p \text{ x } c \text{ x } FLOAT\_SIZE$

Where p and c are the particle number and the number of components to represent the particle position respectively. The third parameter $FLOAT\_SIZE$ is the size of the float type with respect to the specification of the operating system. Applying the above, the decompressed frame occupies 4,035,120 bytes of memory. The alignment value can be obtained directly from the Boost class by which the file is opened. My operating system (macOS Sierra) aligns data in a blocks of 4,096 bytes.

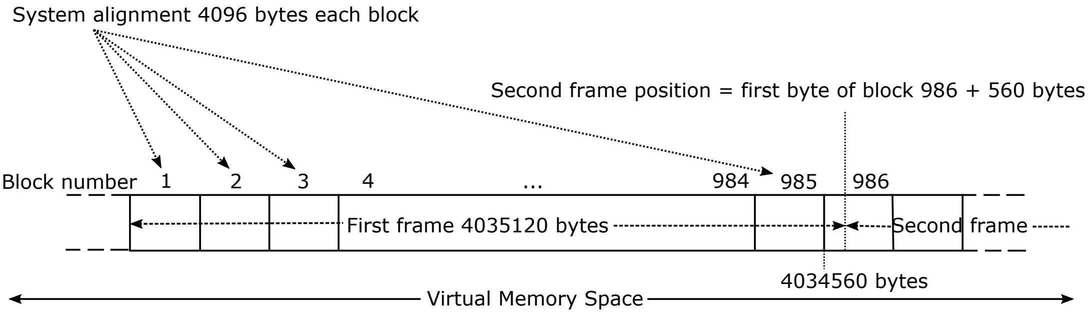

System alignment 4096 bytes each block
Second frame position = first byte of block 986 + 560 bytes
Block number
1
2
3
4
...
984
985
986
First frame 4035120 bytes
Second frame
4034560 bytes
Virtual Memory Space

Figure 7.7: Calculating the position of the second frame in virtual memory space.

Calculating The First Byte of a Frame (a time-step) Operating systems do not allow random access to file data. They provide developers with a pointer that points to the first byte of the required block or blocks. Each block must be a multiple of the operating system's virtual memory allocation granularity which means I cannot ensure that the first byte of a frame will be directly returned. In order to get the correct position of a frame in the file, for either reading or writing, I need to calculate the position. First, I find the index of a block that involves the first byte of a frame. Then I calculate the correct position of the frame. I use the above values to calculate the correct position to write the frame data to the new file. To illustrate, in my case, the first frame occupies 985 blocks and 560 bytes from block number 986. To write the second frame I need to access the block number 986 at byte 560. See Figure 7.7. However, the operating system can not return a pointer to a random position. Alternatively, I open the file at the block 986 (which involves 560 bytes of the first frame) then I write the next frame starting at the returned pointer plus 560 bytes.

Paging the File to a Virtual Memory Space Paging the file means mapping a particular part of the file to a virtual memory space. The paging requires two parameters: the starting position (must be a multiple of the alignment), and the size of the required data (in bytes). The bigger the page size the smaller the number of mapping operations. The performance can be optimized by measuring the system performance against various page sizes. My optimal page size is 14,857,600 bytes. Each page involves three frames, the following is the number of frames per page:

$$Number\ of\ Frames\ Per\ Page = PS/FS$$

Where PS indicates the page size, and FS indicates the frame size.

Writing Data to File First, a file for storing the decompressed data must be opened via either mapped_file or the mapped_file_sink class. These classes obtain the properties of the new file from the mapped_file_params class through its members: 1) path, 2) flags, 3) offset, 4) length and 5) new_file_size [214]. The path holds the file name and location on the hard-disk. The flags identify the access mode (read, write, and read/write). The offset identifies where the mapping begins. The length identifies the size of the mapped data (the page size). The new_file_size identifies the size of the entire file. The entire size of the file can be derived by multiplying a frame size by the total number of frames.

If the file does not exist, a new file is created automatically and a pointer to the first byte of the file in the virtual memory space is returned. If the file already exists a pointer to the first byte of the mapped part is returned. The compressed data can be copied to virtual memory via the *memcopy* function or by casting the pointer type and assigning the data to it directly.

Reading Data from File Mapping a file for reading only follows the same manner described above, however, I utilize the mapped_file_source class which is designed to map a file for reading only. The mapped_file_source returns a read only const char pointer. Reading the pointer value does not require any *memcopy* overhead, the returned pointer can be converted into any other type including user defined types.

7.4.1.2 Mapping OpenGL Buffers

An OpenGL buffer is a data container that belongs to the graphics card. A graphics card has its own memory which is used to store all the OpenGL buffer types and the other OpenGL objects. Traditionally, after creating the OpenGL buffer the data must be transferred or uploaded from RAM to the OpenGL buffer in the GPU memory. There are two ways to upload and download data to the GPU memory. See Figure 7.8. The first way is to use the *glBufferData* and *glBufferSubData* functions [205]. The other way to upload data to the GPU is to get a pointer to the internal drivers' memory with the functions *glMapBuffer* and *glUnmapBuffer*. This pointer can be used to fill the buffer directly which means I save one copy per memory transfer. *OpenGL 4.3* provides three functions to map and unmap buffers: 1) *glMapBuffer()*, 2) *glMapBufferRange()*, and 3) *glUnmapBuffer()*. The *glMapBuffer()* function maps the entire buffer data to a memory whereas the *glMapBufferRange()* maps only a subset of the buffer which means there is no need to re-upload the buffer completely. Finally, after filling the buffer the mapping can be released via the *glUnmapBuffer()*. The *glMapBuffer()* and *glMapBufferRange()* require a valid and binded OpenGL buffer. A valid buffer means that the buffer has a valid name/id that is obtained via *glGenBuffers()*. A Binded buffer means that the buffer is linked with the current *context* by calling *glBindBuffer()*. I note that the buffer size must be defined by allocating some memory via *glBufferData()* before using *glMapBuffer()* and *glMapBufferRange()*. When utilizing *glMapBuffer()* there may be significant performance advantages if a NULL pointer is passed to *glBufferData()* instead of using an initial data [198].

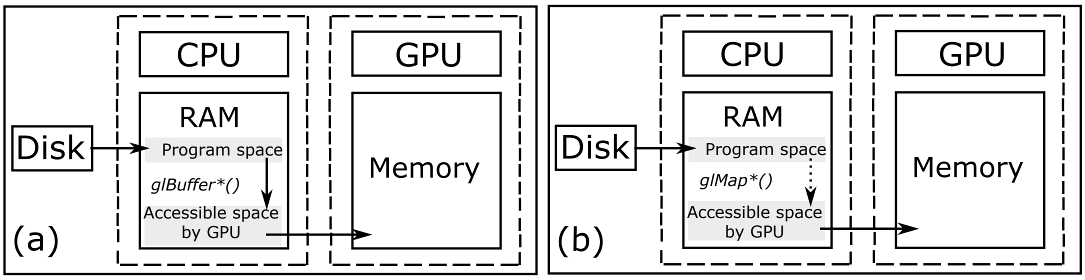


Figure 7.8: Data flow via *glBufferData()* and *glMap*()*. Copying is indicated by a solid arrow and mapping is represented by a dashed arrow. a) *glBufferData()* copies a data from a program space to a buffer or a subset of it. b) *glMap*()* returns a pointer to a memory space that is accessible by the GPU.

7.4.1.3 OpenGL shared context

On the GPU side, my solution relies on the concept of objects sharing. By default the OpenGL framework and OpenCL framework generate two unique *contexts*, the OpenGL *context* and the OpenCL *context*. The OpenGL context is responsible for maintaining the OpenGL objects state and the OpenCL context is responsible for managing the OpenCL memory objects. They are isolated from each other. i.e the OpenGL context is not accessible by the OpenCL framework and the OpenCL context is invisible to the OpenGL framework which means the computation process and the rendered process cannot be synchronized on the GPU. See Figure 7.5. An OpenCL extension addresses this limitation via the so-called OpenCL interoperability [202] . This extension enables the OpenCL framework to access the OpenGL *context*. At a high level, the OpenGL interoperability can be achieved by, first, creating an OpenGL *context* and initializing an OpenCL *context* from it. As not all devices support this feature. A device needs to be queried to determine whether it supports OpenGL interoperability. The *clGetDeviceInfo()* function is used to obtain variety of information about the device (a CPU, a GPU, a

Digital Signal Processor (DSP)). *clGetDeviceInfo()* can return the supported extensions name in a string for the CL_DEVICE_EXTENSIONS property. The cl_APPLE_gl_sharing extension name is used for Mac OS and the cl_khr_gl_sharing name is used for other systems [228]. A proper name, cl_APPLE_gl_sharing or cl_khr_gl_sharing, must appear in the returned string with respect to the host OS. If the OS supports the OpenGL interoperability, then I can utilize it.

To benefit from this feature I need to create two things: 1) create an OpenCL *context* from an OpenGL *context*, and 2) create an OpenCL object from an existing OpenGL object. Creating an OpenCL *context* from an OpenGL *context* requires two steps: First, the OpenGL *context* must be initialized and current before creating the OpenCL *context* [203]. Second, the OpenCL *context* must be configured in order to enable the OpenCL framework to access the OpenGL objects. The configuration is achieved by setting the cl_context_properties structure with the proper values with respect to the OS. On Mac OS, the cl_context_properties structure requires only one property: CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE. However, the value associated with this property must have the data type CGLShareGroupObj, and it can be acquired via the function *CGLGetShareGroup()*. This function requires a *CGLContextObj* structure, which can be obtained by calling *CGLGetCurrentContext()*. The following code shows how these functions work together:

```
CGLContextObj glContext = CGLGetCurrentContext();
CGLShareGroupObj shGroup = CGLGetShareGroup(glContext);
cl_context_properties prop[] = {
  CL_CONTEXT_PROPERTY_USE_CGL_SHAREGROUP_APPLE,
  (cl_context_properties)shGroup,
  0};
```

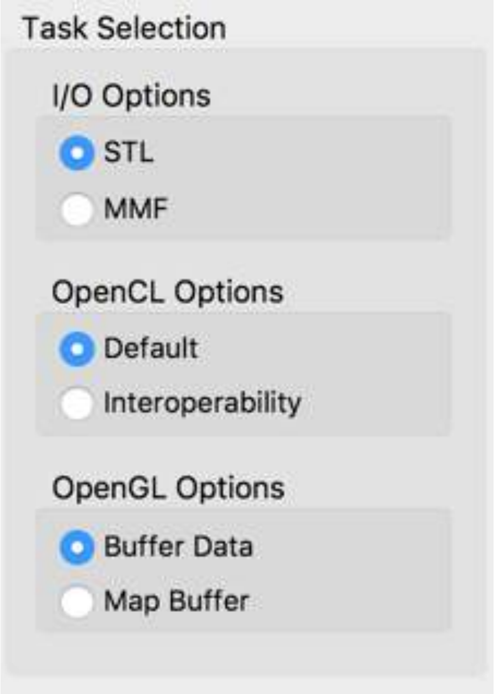
After the cl_context_properties structure is set, an OpenCL context that is capable of accessing OpenGL data can be created via *clCreateContext()* (for more detail and for Linux and Windows configuration see the supplementary material). Hence the OpenCL context is created from an OpenGL context. The OpenCL framework is able to access number of the OpenGL objects and modify them. OpenCL can access and modify vertex buffer objects (VBOs), texture data (texture objects) and pixel data (renderbuffer objects) by creating the corresponding OpenCL memory object via one of four functions: *clCreateFromGLBuffer()*, *clCreateFromGLTexture2D()*, *clCreateFromGLTexture3D()*, or *clCreateFromGLRenderbuffer()*. These functions require the

OpenGL object to be already initialized before they can create an OpenCL object from it. Hence an OpenCL memory object is created from an OpenGL object. Any change in its content affects the OpenGL object content.

Synchronization Sharing data between OpenGL and OpenCL doesn't mean they can access the shared data at the same time. i.e. if OpenCL is manipulating a memory object data created from an OpenGL vertex buffer, then OpenGL can't access the vertex buffer. OpenCL supports synchronization through two functions: 1) *clEnqueueAcquireGLObjects()*, and 2) *clEnqueueReleaseGLObjects()*. The main task for the two functions is to lock and unlock access to the OpenCL memory objects. The first function ensures that the OpenCL will have exclusive access to the data. The second function releases the data and enables OpenGL and other processes to access it. Before acquiring a lock I have to ensure that all OpenGL routines have completed their operation by invoking *glFinish()*. And after releasing the data from the lock I need to ensure that all OpenCL commands have completed their operation by calling *clFinish()*.

7.4.2 Performance Test

Performance measurement is an important step that helps in optimizing code and decide between available methods. I provide three performance tests and show how do they result in the frame rate : i) file I/O approaches (STL vs. MMFs), ii) OpenGL interoperability (OpenCL-GL separated context vs. shared context), and iii) uploading data to OpenGL buffers (Buffer data vs. Map Buffer). The user can choose between these approaches in order to investigate how do they work under different configurations. See figure 7.1. Due to the nature of the data which is dynamic data-set, and in order to provide a smooth transition of the dynamics of the data more than two time-steps per second must be rendered. I render $\sim$ 4 time-steps per second. The process of rendering each time-step involves the following: reading particles position from the file, computing the interaction between the particles, updating the OpenGL buffer based on the computation result, and finally rendering the result. The frame rate is measured per second, and it has been averaged by total time-step. All performances are measured in millisecond (ms) except the final rendering rate which is measured in frames per second (FPS). The performance tests are performed on MacBook Pro Retina, 13-inch, Early 2015 with a 2.7 GHz Intel Core i5 processor and 8 GB of memory (1867 MHz DDR3). The graphics card is Intel Iris Graphics 6100 (1.5 GB of memory). My MD data sample involves a trajectory of 336260 particles over



Task Selection

I/O Options

● STL
○ MMF

OpenCL Options

● Default
○ Interoperability

OpenGL Options

● Buffer Data
○ Map Buffer

Figure 7.9: Task selection. Each task has two options. The user can switch between these to see how do they affect the total rendering rate in FPS.

1981 time-steps. Each position is represented by a 3D floating point vector. The data occupies 8 Gigabyte and it is stored in a binary file. For each time-step the program performs three main tasks: 1) reading the data of the current time-step (so called frame) from the file. 2) for each protein I perform a Protein-Lipids interaction test. 3) visualizing the interaction result and the global molecular dynamics. The performance overhead is associated with the I/O and the computation tasks. The average performance of the I/O task is 17 ms per time step. The computation consumes ∼35 ms of the total rendering time per time step. The rendering frame rate is steady at ∼20 FPS by utilizing the OpenCL-GL default configuration while it almost doubled by utilizing the OpenCL-GL interoperability option.

STL vs. MMFs The program requires fetching the data from the file frame by frame. The performance of fetching the frames varies depending on the position of the frame in the file. STL shows a stable performance (17 ms) for extracting a single frame from the file at any position. MMF requires 5 ms to 20 ms to fetch one frame from the file.

I/O		CL-GL context		GL		Total rendering rate FPS
STL	MMF	Default	interoperability	Buffer Data	Map Buffer	
✓		✓		✓		21 fps
✓		✓			✓	24 fps
✓			✓	✓		42 fps
✓			✓		✓	42 fps
	✓	✓		✓		19 fps
	✓	✓			✓	21 fps
	✓		✓	✓		41 fps
	✓		✓		✓	41 fps

Table 7.1: An overview of the performance test result based on different options. Utilizing shared context results in no overhead of uploading data to the GPU for rendering.

OpenCL-GL shared context vs. separated context The OpenCL-GL context test includes time required to transfer data between the CPU and GPU. The separated context approach requires 40 ms to perform the computations and to transfer the data between the CPU and GPU per time step. The OpenCL-GL shared context results in slight enhancement by completing this task in ∼30 ms. This enhancement comes from the fact that, in the shared context approach, the data is sent from the CPU to the GPU once where as the separated approach requires the data to be uploaded from the CPU to the GPU twice: the first time to perform the computation and the second time to render the data.

Buffer data vs. Map data The final performance test is associated with the rendering. The new particles need to be rendered with respect to the interaction result. Based on the selected computation approach (Open CL-GL separated context or shared context), this task will require uploading the data from the CPU to the GPU. The Map buffer option requires 1 ms per time step whereas the buffer data requires 3 ms.

Chapter 8

Conclusion and Future Work

Contents

"Someday, with my sail piercing the clouds; I will mount the wind, break the waves, and traverse the vast, rolling sea."-Li Bai[1]

8.1 Thesis Conclusion

8.1.1 Thesis Summary

This thesis focused on introducing interactive visualization solutions. These solutions would, theoretically, provide biologists with the ability to obtain a better, and more informed, understanding of their Molecular Dynamics Simulation data. Each chapter in the thesis is dedicated to address specific challenges and then follows with a subsequent introduction to a new solution (Chapters 3, 4, 5, and 6). I began the thesis, in a coherent manner, by first providing the

[1] Li Bai (701-762), also known as Li Bo, Li Po and Li Taibai, was a Chinese poet acclaimed from his own day to the present as a genius and a romantic figure who took traditional poetic forms to new heights.

reader with a broad and deep review of molecular dynamic visualization literature (Chapter 2). In Chapter 3 I aimed to effectively visualize time-dependent, trajectory MD data at the atomic level and provide the user with 2D and 3D representations. This chapter is also concerned with constructively illustrating a number of different visualization techniques. This comprehensible layout, then, successfully enables the user to render the entire set of trajectories or a subset of the data. To maintain and commit to this ease, a variety of filtering techniques were designed to reduce the focus of the rendered data based on the user's chosen attributes. The dual representation, in this effort, helps to compare trajectories in 2D and 3D. Focus + context and zooming are used to explore the local behavior of the molecule dynamics.

In Chapter 4, I introduced a novel and abstract PLI space to address computational and visualization challenges arising from lipid-protein dynamics. The proposed representation highlights PLI better than the classical naive representations and allows experts to quickly highlight interesting molecular features. The visual design of the abstract space simplifies the PLI problem and helps users obtain insight into the PLI. Chapter 5 proposed six levels of detail for both lipid types and the PLI abstract space. The visual design of the abstract space simplifies the PLI challenge and helps users obtain insight into the PLI. The LoD lipid representations also are effective in reducing clutter caused by lipid particles. The LoD PLI space representations, thus, enable users to investigate the PLI space at the desired LoD. It constructively provides great potential to investigate PLI from molecular dynamics systems.

In Chapter 6, I proposed a novel PLI and PPI visualization framework, which utilizes four visual designs. These are very significant in enabling the user to study a time-dependent membrane simulation. The chapter, furthermore, employs abstraction and space projection to address a number of visualization challenges. This novel, hybrid view is not only put forth to produce an original outlook, but is proposed, primarily, to enable the user to study PLI and PPI and the behavior of the PPI and clusters. A details-on-demand view is used to provide the user with desired information about proteins, clusters, and their behavior.

In Chapter 7, I carried on some performance challenges, which led me to utilize advance techniques to address such limitations. The Chapter describes the implementation methodologies that are used to boost the performance. It illustrates the concept of memory mapping by proposing the MMF method from Boost Library and the *glMap*()* functions from OpenGL API. The illustration covers the usage of OpenCL as a means to perform computation tasks in GPU, and I provide an example of utilizing the OpenCL-GL interoperability to enhance the data traffic performance between the CPU and GPU. Chapter 7 also shows how the OpenCL-GL interoperability, and the *glMap*()* functions have a great impact in the final rendering frame rate. This representation highlights PLI better than classical naive representations and allows experts to quickly highlight interesting molecular features. Thus, the representation provides great potential to investigate PLI from molecular dynamics systems. Overall, utilizing abstractions is useful to reveal information that might be difficult to be observed by scientific visualization alone. These chapters, are then vital in securing this outcome.

8.1.2 Conclusion and Future Work

My goal was to set out to design and implement a new approach to exploring MD. The intent behind my efforts was to simplify bioscientists research and find new discoveries. Throughout this thesis, I proposed a number of solutions that help them to study a complex, time-dependant membrane simulation. These solutions are put forth with in-depth insight by utilizing interactive visualization. I produced a set of visualization tools, which enable the user to have full interaction and control over the MD data so that they are able to study the behavior of the molecular dynamics data. From using these tools to their advantage, they can discover different properties such as the trajectory's length and curvature by utilizing different visualization techniques. They can also study the protein-lipid interaction and protein-protein interaction from a 2D and 3D perspective. Additionally, they are able to investigate the protein's clusters and their properties such as their rotation and translation. This tool has successfully met the needs of the domain expert—providing him with the desired visualization that gave him a better understanding of the simulation behavior and it revealed to him new discoveries.

Visualizing molecular structures and models is one of the first analysis steps every computational biologist takes to assess their results. A broad spectrum of tools are available to visualize objects; ranging from protein structure to cavities, both as static items or dynamical data sets. Recent advances in GPU computing improve the efficiency and the quality of the rendering. Nevertheless, molecular visualization remains challenging due to the increasing size of simulation data [53]. First, dealing with models that can expand on different scales, both in terms of structure [52, 54, 28, 59] and time, [58, 53, 54] is not yet solved. This type of visualization needs to be coupled with other constructive methods to grasp the full complexity of molecular systems. Thus, there is a need for real time 3D annotation [28] and filtering [57].

These visualization advances may be combined with HCI and VR [60, 54] to help the user *immerse* in the system. Automating rendering and analysis [56] and storing the result for further analysis [35] will be equally important.

There are many challenges that have been addressed willingly by other researchers. Cavity tools effectively provide users with a sound prediction of protein binding sites and cavity detection [57]. Voronoi diagrams are used to compute and accurately visualize the paths of all regions of a molecule utilizing lighting effects to highlight the cavities and channels [7]. A 3D histogram based method is used to build an instantaneous and constructive graph of the entire time-dependent molecular dynamics simulation [9, 23]. The proteins and bio-molecules are conveniently clustered in the graph so that it can be queried for simulation states and kinetic pathways. A Sigma-r plot is used to plot the average standard deviation of inter-atomic distance (s) as a function of inter-atomic distance (r) [21]. Duality between a Voronoi diagram (VD) and a Delaunay Triangulation (DT) is used to identify tunnels, a high quality result within acceptable computational time [11]. Pathways of solvent molecules in and around protein cavities is identified and visualized [12] through the use of a specialized clustering algorithm.

GPU ray casting techniques are used in the effort to reduce visual complexity of protein surface to the implicit mathematical description of the Solvent Excluded Surface (SES)[3]. A fast density segmentation is used to extract a protein cavity through partial segmentation rather than segmenting the whole volume [13]. Tunnels and channels in large collections of protein structures are analyzed and information of individual transport pathways and their time evolution is visualized [14].

Co-variance matrices (dimension reduction, ray tracing and geometric deformation) are used to represent residues as spheres [15]. Path line-analogous representation for visualization of molecular motion is used and output files that are printed via 3D printers [18]. Diffuse Interreflections and Ambient Occlusion is employed in visualizing atomic molecular structures [19]. Molecular void spaces are visualized, and all lining amino acids are displayed throughout the whole simulation and even their properties are explored [22]. In future work, I will use a longer membrane simulation consisting of different types of proteins and I will focus on the visual analysis aspects of the MD data. There is still more information that can potentially be visualized, to achieve significant insight, such as lipid root-mean-square deviation (RMSD) or distance to protein.

An interactive classification of the protein molecules, based on a variety of similarity metrics, can help better understanding the MD. A new visualization is desired to investigate the impact of the correlated motions of lipid molecules on the proteins. Enabling the user to add annotation to the scene, which will ease the investigation. Also adding new features to help the user to easily and effectively compare different features of two or more proteins. Visual analytic techniques can be applied to the 2D PLI results to understand the relation between residues and particles involved in the interaction. Discrete abstract rings can be employed to visualize protein deformation. The depth of the interaction can then be encoded on the tiles.

Bibliography

[1] S. Doutreligne, T. Cragnolini, S. Pasquali, P. Derreumaux, and M. Baaden, “UnityMol: Interactive scientific visualization for integrative biology,” in *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, 2014, pp. 109–110, http://www.baaden.ibpc.fr/umol/ (last accessed: 14.02.17).

[2] S. Grottel, M. Krone, C. Müller, G. Reina, and T. Ertl, “Megamol - a prototyping framework for particle-based visualization,” *IEEE transactions on visualization and computer graphics*, vol. 21, no. 2, pp. 201–214, 2015, http://www.megamol.org (last accessed: 14.02.17).

[3] M. Krone, K. Bidmon, and T. Ertl, “Interactive visualization of molecular surface dynamics,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1391–1398, 2009.

[4] A. S. Rose and P. W. Hildebrand, “NGL Viewer: a web application for molecular visualization,” *Nucleic Acids Research*, vol. 43, no. W1, p. W576, 2015, http://proteinformatics.charite.de/ngl (last accessed: 14.02.17).

[5] A. S. Rose, A. R. Bradley, Y. Valasatava, J. M. Duarte, A. Prlić, and P. W. Rose, “Web-based molecular graphics for large complexes,” in *Proceedings of the 21st International Conference on Web3D Technology*, ser. Web3D ’16, 2016, pp. 185–186.

[6] M. Falk, M. Krone, and T. Ertl, “Atomistic Visualization of Mesoscopic Whole-Cell Simulations Using Ray-Casted Instancing,” *Computer Graphics Forum*, vol. 32, no. 8, pp. 195–206, 2013.

[7] N. Lindow, D. Baum, and H.-C. Hege, "Voronoi-based extraction and visualization of molecular paths," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 2025–2034, 2011.

[8] N. Lindow, D. Baum, A. Bondar, and H.-C. Hege, "Dynamic channels in biomolecular systems: Path analysis and visualization," *Biological Data Visualization (BioVis), 2012 IEEE Symposium on*, pp. 99–106, 2012.

[9] D. Rozmanov, S. Baoukina, and D. P. Tieleman, "Density based visualization for molecular simulation," *Faraday discussions*, vol. 169, pp. 225–243, 2014.

[10] D. Bhattarai and B. B. Karki, "Visualization of atomistic simulation data for spatio-temporal information," 2006.

[11] P. Medek, P. Benes, and J. Sochor, "Computation of tunnels in protein molecules using delaunay triangulation," *Journal of WSCG*, vol. 15, no. 1-3, pp. 107–114, 2007.

[12] K. Bidmon, S. Grottel, F. Bös, J. Pleiss, and T. Ertl, "Visual abstractions of solvent pathlines near protein cavities," *Computer Graphics Forum*, vol. 27, no. 3, pp. 935–942, 2008.

[13] M. Krone, M. Falk, S. Rehm, J. Pleiss, and T. Ertl, "Interactive exploration of protein cavities," *Computer Graphics Forum*, vol. 30, no. 3, pp. 673–682, 2011.

[14] E. Chovancova, A. Pavelka, P. Benes, O. Strnad, J. Brezovsky, B. Kozlíková, A. Góra, V. Sustr, M. Klvana, P. Medek, L. Biedermannová, J. Sochor, and J. Damborský, "CAVER 3.0: A tool for the analysis of transport pathways in dynamic protein structures," *PLoS Computational Biology*, vol. 8, no. 10, 2012.

[15] M. Fioravante, A. Shook, I. Thorpe, and P. Rheingans, "Visualizing motional correlations in molecular dynamics using geometric deformations," *Computer Graphics Forum*, vol. 32, no. 3pt3, pp. 311–320, Jun. 2013.

[16] D. Sehnal, R. S. Vareková, K. Berka, L. Pravda, V. Navrátilová, P. Banás, C.-M. Ionescu, M. Otyepka, and J. Koca, "Mole 2.0: Advanced approach for analysis of biomacromolecular channels." *J. Cheminformatics*, vol. 5, p. 39, 2013.

[17] M. Krone, D. Kauker, G. Reina, and T. Ertl, “Visual analysis of dynamic protein cavities and binding sites,” *Pacific Visualization Symposium (PacificVis)*, pp. 301–305, 2014.

[18] S. M. Dabdoub, R. W. Rumpf, A. D. Shindhelm, and W. C. Ray, “Moflow: visualizing conformational changes in molecules as molecular flow improves understanding,” *BMC Proceedings*, vol. 9, no. 6, pp. 1–12, 2015.

[19] R. Skanberg, P.-P. Vázquez, V. Guallar, and T. Ropinski, “Real-time molecular visualization supporting diffuse interreflections and ambient occlusion,” *IEEE Trans. Vis. Comput. Graph*, vol. 22, no. 1, pp. 718–727, 2016.

[20] N. Blöchliger, A. Vitalis, and A. Caflisch, “High-resolution visualisation of the states and pathways sampled in molecular dynamics simulations,” *Scientific reports*, vol. 4, 2014.

[21] H. Zhou, S. Li, and L. Makowski, “Visualizing global properties of a molecular dynamics trajectory,” *Proteins: Structure, Function, and Bioinformatics*, vol. 84, no. 1, pp. 82–91, 2016.

[22] J. Byška, M. Le Muzic, M. E. Gröller, I. Viola, and B. Kozlikova, “Animoaminominer: Exploration of protein tunnels and their properties in molecular dynamics,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 22, no. 1, pp. 747–756, 2016.

[23] H. Bhatia, A. G. Gyulassy, V. Pascucci, M. Bremer, M. T. Ong, V. Lordi, E. W. Draeger, J. E. Pask, and P.-T. Bremer, “Interactive exploration of atomic trajectories through relative-angle distribution and associated uncertainties,” pp. 120–127, 2016.

[24] I. A. Sadarjoen and F. H. Post, “Detection, quantification, and tracking of vortices using streamline geometry,” *Computers & Graphics*, vol. 24, no. 3, pp. 333–341, 2000.

[25] W. Humphrey, A. Dalke, and K. Schulten, “VMD: visual molecular dynamics,” *Journal of Molecular Graphics*, vol. 14, no. 1, pp. 33–38, 1996.

[26] N. Alharbi, M. Chavent, M. Krone, and R. S. Laramee, “VAPLI: Novel visual abstraction for protein-lipid interactions,” *IEEE SciVis Short Papers*, pp. 133–137, 2018.

[27] Z. Lv, A. Tek, F. Da Silva, C. Empereur-Mot, M. Chavent, and M. Baaden, “Game on, science-how video game technology may help biologists tackle visualization challenges,” *PloS one*, vol. 8, no. 3, p. e57990, 2013.

[28] M. Chavent, B. Lévy, M. Krone, K. Bidmon, J.-P. Nominé, T. Ertl, and M. Baaden, "Gpu-powered tools boost molecular visualization," *Briefings in Bioinformatics*, p. bbq089, 2011.

[29] P. Samui, *Handbook of Research on Advanced Computational Techniques for Simulation-based Engineering*. IGI Global, 2015.

[30] D. A. Case, T. E. Cheatham, T. Darden, H. Gohlke, R. Luo, K. M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R. J. Woods, "The amber biomolecular simulation programs," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1668–1688, 2005.

[31] B. R. Brooks, C. L. Brooks, A. D. MacKerell, L. Nilsson, R. J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch *et al.*, "Charmm: the biomolecular simulation program," *Journal of computational chemistry*, vol. 30, no. 10, pp. 1545–1614, 2009.

[32] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. Berendsen, "Gromacs: fast, flexible, and free," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005.

[33] M. Karplus, G. A. Petsko *et al.*, "Molecular dynamics simulations in biology," *Nature*, vol. 347, no. 6294, pp. 631–639, 1990.

[34] J. R. Perilla, B. C. Goh, C. K. Cassidy, B. Liu, R. C. Bernardi, T. Rudack, H. Yu, Z. Wu, and K. Schulten, "Molecular dynamics simulations of large macromolecular complexes," *Current opinion in structural biology*, vol. 31, pp. 64–74, 2015.

[35] M. Chavent, A. L. Duncan, and M. S. Sansom, "Molecular dynamics simulations of membrane proteins and their interactions: from nanoscale to mesoscale," *Current Opinion in Structural Biology*, vol. 40, pp. 8–16, 2016.

[36] C. D. Hansen and C. R. Johnson, *Visualization handbook*. Elsevier, 2011.

[37] D. Keim, G. Andrienko, J.-D. Fekete, C. Görg, J. Kohlhammer, and G. Melançon, "Visual analytics: Definition, process, and challenges," in *Information visualization*. Springer, 2008, pp. 154–175.

[38] L. McNabb and R. S. Laramee, "Survey of surveys (sos)-mapping the landscape of survey papers in information visualization," in *Computer Graphics Forum*, vol. 36, no. 3. Wiley Online Library, 2017, pp. 589–617.

[39] D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler, “Challenges in visual data analysis,” in *Tenth International Conference on Information Visualisation (IV’06)*. IEEE, 2006, pp. 9–16.

[40] T.-M. Rhyne, M. Tory, T. Munzner, M. Ward, C. Johnson, and D. H. Laidlaw, “Information and scientific visualization: Separate but equal or happy together at last,” in *Proceedings of the 14th IEEE Visualization 2003 (VIS’03)*. IEEE Computer Society, 2003, p. 115.

[41] E. F. Pettersen, T. D. Goddard, C. C. Huang, G. S. Couch, D. M. Greenblatt, E. C. Meng, and T. E. Ferrin, “Ucsf chimera–a visualization system for exploratory research and analysis,” *Journal of computational chemistry*, vol. 25, no. 13, pp. 1605–1612, 2004.

[42] W. L. DeLano, “Pymol: An open-source molecular graphics tool,” *CCP4 Newsletter On Protein Crystallography*, vol. 40, pp. 82–92, 2002.

[43] D. Brodbeck, R. Mazza, and D. Lalanne, “Interactive visualization-a survey,” in *Human machine interaction*. Springer, 2009, pp. 27–46.

[44] B. Shneiderman, “The eyes have it: A task by data type taxonomy for information visualizations,” in *The craft of information visualization*. Elsevier, 2003, pp. 364–371.

[45] N. Alharbi, M. Alharbi, X. Martinez, M. Krone, A. Rose, M. Baaden, R. S. Laramee, and M. Chavent, “Molecular visualization of computational biology data: A survey of surveys,” *Eurovis short papers*, pp. 133–137, 2017.

[46] N. Alharbi, R. S. Laramee, and M. Chavent, “Molpathfinder: Interactive multi-dimensional path filtering of molecular dynamics simulation data,” pp. 9–16, 2016.

[47] N. Alharbi, M. Chavent, and R. Laramee, “Real-time rendering of molecular dynamics simulation data: A tutorial,” *EG UK Computer Graphics*, pp. 9–16, 2017.

[48] N. Alharbi, M. Chavent, M. Krone, and R. S. Laramee, “Lod lpi: Level of detail for visualizing time-dependent, protein-lipid interaction,” *International Conference on Information Visualization and Applications (IVAPP)*, pp. 164–174, 2019.

[49] C. Levinthal, *Molecular Model-Building by Computer*. WH Freeman and Company, 1966.

[50] R. Langridge, T. E. Ferrin, I. D. Kuntz, and M. L. Connolly, "Real-time color graphics in studies of molecular interactions," *Science*, vol. 211, no. 4483, pp. 661–666, 1981.

[51] S. I. O'Donoghue, A.-C. Gavin, N. Gehlenborg, D. S. Goodsell, J.-K. Hériché, C. B. Nielsen, C. North, A. J. Olson, J. B. Procter, D. W. Shattuck *et al.*, "Visualizing biological data - now and in the future," *Nature methods*, vol. 7, pp. S2–S4, 2010.

[52] T. D. Goddard and T. E. Ferrin, "Visualization software for molecular assemblies," *Current Opinion in Structural Biology*, vol. 17, no. 5, pp. 587–595, 2007.

[53] B. Kozlíková, M. Krone, M. Falk, N. Lindow, M. Baaden, D. Baum, I. Viola, J. Parulek, and H.-C. Hege, "Visualization of biomolecular structures: State of the art revisited," *Computer Graphics Forum*, vol. 36, no. 8, pp. 178–204, 2017.

[54] S. I. O'Donoghue, D. S. Goodsell, A. S. Frangakis, F. Jossinet, R. A. Laskowski, M. Nilges, H. R. Saibil, A. Schafferhans, R. C. Wade, E. Westhof *et al.*, "Visualization of macromolecular structures," *Nature Methods*, vol. 7, pp. S42–S55, 2010.

[55] J. E. Stone, D. J. Hardy, I. S. Ufimtsev, and K. Schulten, "Gpu-accelerated molecular modeling coming of age," *Journal of Molecular Graphics and Modelling*, vol. 29, no. 2, pp. 116–125, 2010.

[56] 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*, vol. 31, no. 1, pp. 38–49, 2013.

[57] 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*, vol. 35, no. 3, pp. 527–551, 2016. [Online]. Available: http://dx.doi.org/10.1111/cgf.12928

[58] M. Secrier and R. Schneider, "Visualizing time-related data in biology, a review," *Briefings in bioinformatics*, p. bbt021, 2013.

[59] W. Im, J. Liang, A. Olson, H.-X. Zhou, S. Vajda, and I. A. Vakser, "Challenges in structural approaches to cell modeling," *Journal of molecular biology*, 2016.

[60] J. D. Hirst, D. R. Glowacki, and M. Baaden, “Molecular simulations and visualization: introduction and overview,” *Faraday discussions*, vol. 169, pp. 9–22, 2014.

[61] “Scopus,” (https://www.scopus.com (last accessed: 09.02.17).

[62] S. Bird, “Nltk: The natural language toolkit,” in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[63] A. Mueller, “word_cloud: A little word cloud generator in python,” https://github.com/amueller/word_cloud (last accessed: 31.01.17).

[64] “VIZBI - Visualizing Biological Data,” https://vizbi.org (last accessed: 31.01.17).

[65] “BioVis,” http://biovis.net (last accessed: 31.01.17).

[66] “PubMed,” https://www.pubmed.gov (last accessed: 31.01.17).

[67] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten, “Scalable molecular dynamics with namd,” *Journal of computational chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005.

[68] M. Petřek, M. Otyepka, P. Banáš, P. Košinová, J. Koča, and J. Damborskỳ, “Caver: a new tool to explore routes from protein clefts, pockets and cavities,” *BMC bioinformatics*, vol. 7, no. 1, p. 316, 2006.

[69] M. Tarini, P. Cignoni, and C. Montani, “Ambient occlusion and edge cueing to enhance real time molecular visualization,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1237–1244, 2006, cited By 189. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-33845621535&doi=10.1109%2fTVCG.2006.115&partnerID=40&md5=f68da324759cc8eba3d516c1e690a2de

[70] M. Krone, J. E. Stone, T. Ertl, and K. Schulten, “Fast Visualization of Gaussian Density Surfaces for Molecular Dynamics and Particle System Trajectories,” in *EuroVis - Short Papers*, vol. 1, 2012, pp. 67–71.

[71] D. S. Goodsell, “Visual methods from atoms to cells,” *Structure*, vol. 13, no. 3, pp. 347–354, 2005.

[72] A. Gillet, M. Sanner, D. Stoffler, and A. Olson, "Tangible interfaces for structural molecular biology," *Structure*, vol. 13, no. 3, pp. 483–491, 2005.

[73] C. W. Hopkins, S. Le Grand, R. C. Walker, and A. E. Roitberg, "Long-time-step molecular dynamics through hydrogen mass repartitioning," *Journal of chemical theory and computation*, vol. 11, no. 4, pp. 1864–1874, 2015.

[74] D. R. Roe and T. E. Cheatham III, "Ptraj and cpptraj: Software for processing and analysis of molecular dynamics trajectory data," *Journal of Chemical Theory and Computation*, vol. 9, no. 7, pp. 3084–3095, 2013.

[75] J. E. Bara, C. H. Turner, Z. Zhang, and C. I. Hawkins, "Tangible visualization of molecular dynamics simulations using 3-d printing," *Education for Chemical Engineers*, vol. 13, pp. 9–16, 2015.

[76] P. Isenberg, F. Heimerl, S. Koch, T. Isenberg, P. Xu, C. Stolper, M. Sedlmair, J. Chen, T. Möller, and J. Stasko, "Visualization publication dataset," *Dataset: http://vispubdata.org/.(2015). http://vispubdata.org/Published Jun*, 2015.

[77] R. S. Laramee, "How to read a visualization research paper: Extracting the essentials," *IEEE computer graphics and applications*, vol. 31, no. 3, pp. 78–82, 2011.

[78] M. Petřek, P. Košinová, J. Koča, and M. Otyepka, "Mole: a voronoi diagram-based explorer of molecular channels, pores, and tunnels," *Structure*, vol. 15, no. 11, pp. 1357–1363, 2007.

[79] 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*, vol. 73, no. 1, pp. 72–86, 2008.

[80] M. Manák and I. Kolingerová, "Fast discovery of voronoi vertices in the construction of voronoi diagram of 3d balls," *Voronoi Diagrams in Science and Engineering (ISVD), 2010 International Symposium on*, pp. 95–104, 2010.

[81] M. Raunest and C. Kandt, "dxtuber: Detecting protein cavities, tunnels and clefts based on protein and solvent dynamics," *Journal of Molecular Graphics and Modelling*, vol. 29, no. 7, pp. 895–905, 2011.

[82] M. G. Razul, J. Hendry, and P. Kusalik, “Mechanisms of heterogeneous crystal growth in atomic systems: Insights from computer simulations,” *The Journal of chemical physics*, vol. 123, no. 20, p. 204722, 2005.

[83] J. Vatamanu and P. G. Kusalik, “Molecular insights into the heterogeneous crystal growth of si methane hydrate,” *The Journal of Physical Chemistry B*, vol. 110, no. 32, pp. 15 896–15 904, 2006.

[84] I. M. Svishchev and P. G. Kusalik, “Crystallization of liquid water in a molecular dynamics simulation,” *Physical review letters*, vol. 73, no. 7, p. 975, 1994.

[85] J. Liang, C. Woodward, and H. Edelsbrunner, “Anatomy of protein pockets and cavities: measurement of binding site geometry and implications for ligand design,” *Protein science*, vol. 7, no. 9, pp. 1884–1897, 1998.

[86] D. Bakowies and W. F. van Gunsteren, “Water in protein cavities: A procedure to identify internal water and exchange pathways and application to fatty acid-binding protein,” *Proteins: Structure, Function, and Bioinformatics*, vol. 47, no. 4, pp. 534–545, 2002.

[87] J. Greer and B. L. Bush, “Macromolecular shape and surface maps by solvent exclusion,” *Proceedings of the National Academy of Sciences*, vol. 75, no. 1, pp. 303–307, 1978.

[88] M. F. Sanner, A. J. Olson, and J.-C. Spehner, “Reduced surface: an efficient way to compute molecular surfaces,” *Biopolymers*, vol. 38, no. 3, pp. 305–320, 1996.

[89] M. Weisel, E. Proschak, G. Schneider *et al.*, “Pocketpicker: analysis of ligand binding-sites with shape descriptors,” *Chem Cent J*, vol. 1, no. 7, pp. 1–17, 2007.

[90] B. Huang and M. Schroeder, “Ligsite csc: predicting ligand binding sites using the connolly surface and degree of conservation,” *BMC structural biology*, vol. 6, no. 1, p. 1, 2006.

[91] M. S. Till and G. M. Ullmann, “Mcvol-a program for calculating protein volumes and identifying cavities by a monte carlo algorithm,” *Journal of molecular modeling*, vol. 16, no. 3, pp. 419–429, 2010.

[92] M. Krone, C. Dachsbacher, and T. Ertl, “Parallel computation and interactive visualization of time-varying solvent excluded surfaces,” *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, pp. 402–405, 2010.

[93] N. Lindow, D. Baum, S. Prohaska, and H.-C. Hege, “Accelerated visualization of dynamic molecular surfaces,” *Computer Graphics Forum*, vol. 29, no. 3, pp. 943–952, 2010.

[94] G. Cipriano and M. Gleicher, “Molecular surface abstraction,” *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 6, pp. 1608–1615, 2007.

[95] J. Giard and B. Macq, “Molecular surface mesh generation by filtering electron density map,” *Journal of Biomedical Imaging*, vol. 2010, p. 10, 2010.

[96] S. Dias, K. Bora, and A. Gomes, “Cuda-based triangulations of convolution molecular surfaces,” *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 531–540, 2010.

[97] T. Can, C.-I. Chen, and Y.-F. Wang, “Efficient molecular surface generation using level-set methods,” *Journal of Molecular Graphics and Modelling*, vol. 25, no. 4, pp. 442–454, 2006.

[98] M. Phillips, I. Georgiev, A. K. Dehof, S. Nickels, L. Marsalek, H.-P. Lenhof, A. Hildebrandt, and P. Slusallek, “Measuring properties of molecular surfaces using ray casting,” *Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pp. 1–7, 2010.

[99] P. Medek, P. Beneš, and J. Sochor, “Multicriteria tunnel computation,” *Proceedings of the Tenth IASTED International Conference on Computer Graphics and Imaging, ACTA Press*, pp. 160–164, 2008.

[100] O. S. Smart, J. G. Neduvelil, X. Wang, B. Wallace, and M. S. Sansom, “Hole: a program for the analysis of the pore dimensions of ion channel structural models,” *Journal of molecular graphics*, vol. 14, no. 6, pp. 354–360, 1996.

[101] E. Yaffe, D. Fishelovitch, H. J. Wolfson, D. Halperin, and R. Nussinov, “Molaxis: a server for identification of channels in macromolecules,” *Nucleic acids research*, vol. 36, no. 2, pp. 210–215, 2008.

[102] A. Bryden, G. N. P. Jr., and M. Gleicher, "Automated illustration of molecular flexibility," *IEEE Trans. Vis. Comput. Graph*, vol. 18, no. 1, pp. 132–145, 2012.

[103] S. Barlowe, Y. Liu, J. Y. 0001, D. R. Livesay, D. J. Jacobs, J. Mottonen, and D. Verma, "Wavemap: Interactively discovering features from protein flexibility matrices using wavelet-based visual analytics," *Comput. Graph. Forum*, vol. 30, no. 3, 2011.

[104] D. G. Levitt and L. J. Banaszak, "Pocket: a computer graphies method for identifying and displaying protein cavities and their surrounding amino acids," *Journal of molecular graphics*, vol. 10, no. 4, pp. 229–234, 1992.

[105] 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*, vol. 15, no. 6, pp. 359–363, 1997.

[106] B. K. Ho and F. Gruswitz, "Hollow: generating accurate representations of channel and interior surfaces in molecular structures," *BMC structural biology*, vol. 8, no. 1, p. 49, 2008.

[107] N. R. Voss and M. Gerstein, "3v: cavity, channel and cleft volume calculator and extractor," *Nucleic acids research*, vol. 38, no. 2, pp. 555–562, 2010.

[108] R. G. Coleman and K. A. Sharp, "Finding and characterizing tunnels in macromolecules with application to ion channels and pores," *Biophysical journal*, vol. 96, no. 2, pp. 632–645, 2009.

[109] G. P. Brady Jr and P. F. Stouten, "Fast prediction and visualization of protein binding pockets with pass," *Journal of computer-aided molecular design*, vol. 14, no. 4, pp. 383–401, 2000.

[110] R. A. Laskowski, "Surfnet: a program for visualizing molecular surfaces, cavities, and intermolecular interactions," *Journal of molecular graphics*, vol. 13, no. 5, pp. 323–330, 1995.

[111] M. Pellegrini-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 Comput Biol*, vol. 5, no. 7, p. e1000440, 2009.

[112] L. Schrodinger, "The pymol molecular graphics system, version 1.3 r1. 2010," *There is no corresponding record for this reference*, 2010.

[113] 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*, vol. 26, no. 1, pp. 46–52, 2010.

[114] P. Schmidtke, A. Bidon-Chanal, F. J. Luque, and X. Barril, "Mdpocket: open-source cavity detection and characterization on molecular dynamics trajectories," *Bioinformatics*, vol. 27, no. 23, pp. 3276–3285, 2011.

[115] J. Parulek, C. Turkay, N. Reuter, and I. Viola, "Implicit surfaces for interactive graph based cavity analysis of molecular simulations," *Biological Data Visualization (BioVis), 2012 IEEE Symposium on*, pp. 115–122, 2012.

[116] M. Krone, G. Reina, C. Schulz, T. Kulschewski, J. Pleiss, and T. Ertl, "Interactive extraction and tracking of biomolecular surface features," *Computer Graphics Forum*, vol. 32, no. 3pt3, pp. 331–340, 2013.

[117] J. Schmidt-Ehrenberg, D. Baum, and H.-C. Hege, "Visualizing dynamic molecular conformations," in *Proceedings of IEEE Visualization 2002*, R. J. Moorhead, M. Gross, and K. I. Joy, Eds., IEEE Computer Society. IEEE Computer Society Press, Oct. 2002, pp. 235–242.

[118] D. Bhattarai and B. B. Karki, "Atomistic visualization: on-the-fly data extraction and rendering," *ACM Southeast Regional Conference*, pp. 437–442, 2007.

[119] M. Falk, M. Krone, and T. Ertl, "Atomistic visualization of mesoscopic whole-cell simulations using ray-casted instancing," *Computer Graphics Forum*, vol. 32, no. 8, pp. 195–206, 2013.

[120] A. Joshi and P. Rheingans, "Illustration-inspired techniques for visualizing time-varying data," *IEEE Visualization*, p. 86, 2005.

[121] J. T. Kajiya, "The rendering equation," *ACM Siggraph Computer Graphics*, vol. 20, no. 4, pp. 143–150, 1986.

[122] R. Hoetzlein, “Fast fixed-radius nearest neighbors: Interactive million–particle fluids,” *GPU Technology Conference*, 2014.

[123] N. Blöchliger, A. Vitalis, and A. Caflisch, “A scalable algorithm to order and annotate continuous observations reveals the metastable states visited by dynamical systems,” *Computer Physics Communications*, vol. 184, no. 11, pp. 2446–2453, 2013.

[124] L. Makowski, D. J. Rodi, S. Mandava, D. D. Minh, D. B. Gore, and R. F. Fischetti, “Molecular crowding inhibits intramolecular breathing motions in proteins,” *Journal of molecular biology*, vol. 375, no. 2, pp. 529–546, 2008.

[125] A. Zen, V. Carnevale, A. M. Lesk, and C. Micheletti, “Correspondences between low-energy modes in enzymes: Dynamics-based alignment of enzymatic functional families,” *Protein Science*, vol. 17, no. 5, pp. 918–929, 2008.

[126] M. Münz, R. Lyngsø, J. Hein, and P. C. Biggin, “Dynamics based alignment of proteins: an alternative approach to quantify dynamic similarity,” *BMC bioinformatics*, vol. 11, no. 1, p. 188, 2010.

[127] N. Lindow, D. Baum, A.-N. Bondar, and H.-C. Hege, “Exploring cavity dynamics in biomolecular systems,” *BMC bioinformatics*, vol. 14, no. Suppl 19, p. S5, 2013.

[128] A. Kanitsar, D. Fleischmann, R. Wegenkittl, P. Felkel, and M. E. Gröller, *CPR-curved planar reformation*. IEEE, 2002.

[129] S. Burov, S. A. Tabei, T. Huynh, M. P. Murrell, L. H. Philipson, S. A. Rice, M. L. Gardel, N. F. Scherer, and A. R. Dinner, “Distribution of directional change as a signature of complex dynamics,” *Proceedings of the National Academy of Sciences*, vol. 110, no. 49, pp. 19 689–19 694, 2013.

[130] J. Savage and G. A. Voth, “Persistent subdiffusive proton transport in perfluorosulfonic acid membranes,” *The journal of physical chemistry letters*, vol. 5, no. 17, pp. 3037–3042, 2014.

[131] N. Gehlenborg, S. I. O’donoghue, N. S. Baliga, A. Goesmann, M. A. Hibbs, H. Kitano, O. Kohlbacher, H. Neuweger, R. Schneider, D. Tenenbaum *et al.*, “Visualization of omics data for systems biology,” *Nature methods*, vol. 7, no. 3s, p. S56, 2010.

[132] H. Miao, T. Klein, D. Kořil, P. Mindek, K. Schatz, M. E. Gröller, B. Kozlíková, T. Isenberg, and I. Viola, "Multiscale molecular visualization," *Journal of molecular biology*, vol. 431, no. 6, pp. 1049–1070, 2018.

[133] M. Falk, M. Klann, M. Reuss, and T. Ertl, "Visualization of signal transduction processes in the crowded environment of the cell," pp. 169–176, 2009.

[134] M. Le Muzic, J. Parulek, A.-K. Stavrum, and I. Viola, "Illustrative visualization of molecular reactions using omniscient intelligence and passive agents," in *Computer Graphics Forum*, vol. 33, no. 3, 2014, pp. 141–150.

[135] P. de Heras Ciechomski, M. Klann, R. Mange, and H. Koeppl, "From biochemical reaction networks to 3d dynamics in the cell: The zigcell3d modeling, simulation and visualisation framework," pp. 41–48, 2013.

[136] N. A. Khazanov and H. A. Carlson, "Exploring the composition of protein-ligand binding sites on a large scale," *PLoS computational biology*, vol. 9, no. 11, p. e1003321, 2013.

[137] P. Hermosilla, J. Estrada, V. Guallar, T. Ropinski, A. Vinacua, and P.-P. Vázquez, "Physics-based visual characterization of molecular interaction forces," *IEEE transactions on visualization and computer graphics*, vol. 23, no. 1, pp. 731–740, 2017.

[138] P. Vázquez, P. Hermosilla, V. Guallar, J. Estrada, and A. Vinacua, "Visual analysis of protein-ligand interactions," *Computer Graphics Forum*, vol. 37, no. 3, pp. 391–402, 2018.

[139] R. D. Finn, M. Marshall, and A. Bateman, "i Pfam: visualization of protein–protein interactions in pdb at domain and amino acid resolutions," *Bioinformatics*, vol. 21, no. 3, pp. 410–412, 2004.

[140] D. R. Koes and C. J. Camacho, "Pocketquery: protein–protein interaction inhibitor starting points from protein–protein interaction structure," *Nucleic acids research*, vol. 40, no. W1, pp. W387–W392, 2012.

[141] K. Furmanová, J. Byška, E. M. Gröller, I. Viola, J. J. Paleček, and B. Kozlíková, "Cozoid: contact zone identifier for visual analysis of protein-protein interactions," *BMC bioinformatics*, vol. 19, no. 1, p. 125, 2018.

[142] M. Le Muzic, M. Waldner, J. Parulek, and I. Viola, "Illustrative timelapse: A technique for illustrative visualization of particle-based simulations," in *2015 IEEE Pacific Visualization Symposium (PacificVis)*. IEEE, 2015, pp. 247–254.

[143] M. Chavent, A. L. Duncan, P. Rassam, O. Birkholz, J. Hélie, T. Reddy, D. Beliaev, B. Hambly, J. Piehler, C. Kleanthous *et al.*, "How nanoscale protein interactions determine the mesoscale dynamic organisation of bacterial outer membrane proteins," *Nature communications*, vol. 9, 2018.

[144] D. M. Owen, C. Rentero, J. Rossy, A. Magenau, D. Williamson, M. Rodriguez, and K. Gaus, "Palm imaging and cluster analysis of protein heterogeneity at the cell surface," *Journal of biophotonics*, vol. 3, no. 7, pp. 446–454, 2010.

[145] D. M. Owen, D. Williamson, A. Magenau, J. Rossy, and K. Gaus, "Optical techniques for imaging membrane domains in live cells (live-cell palm of protein clustering)," in *Methods in enzymology*. Elsevier, 2012, vol. 504, pp. 221–235.

[146] B. B. Langdon, M. Kastantin, R. Walder, and D. K. Schwartz, "Interfacial protein–protein associations," *Biomacromolecules*, vol. 15, no. 1, pp. 66–74, 2013.

[147] J. Prescher, V. Baumgärtel, S. Ivanchenko, A. A. Torrano, C. Bräuchle, B. Müller, and D. C. Lamb, "Super-resolution imaging of escrt-proteins at hiv-1 assembly sites," *PLoS pathogens*, vol. 11, no. 2, p. e1004677, 2015.

[148] J. G. Croissant, D. Zhang, S. Alsaiari, J. Lu, L. Deng, F. Tamanoi, A. M. AlMalik, J. I. Zink, and N. M. Khashab, "Protein-gold clusters-capped mesoporous silica nanoparticles for high drug loading, autonomous gemcitabine/doxorubicin co-delivery, and in-vivo tumor imaging," *Journal of Controlled Release*, vol. 229, pp. 183–191, 2016.

[149] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, p. 90, 2007.

[150] G. Hedger and M. S. Sansom, "Lipid interaction sites on channels, transporters and receptors: recent insights from molecular dynamics simulations," *Biochimica et Biophysica Acta (BBA)-Biomembranes*, 2016.

[151] M. Chavent, T. Reddy, J. Goose, A. C. E. Dahl, J. E. Stone, B. Jobard, and M. S. Sansom, "Methodologies for the analysis of instantaneous lipid diffusion in md simulations of large membrane systems," *Faraday discussions*, vol. 169, pp. 455–475, 2014.

[152] P. Rassam, N. A. Copeland, O. Birkholz, C. Tóth, M. Chavent, A. L. Duncan, S. J. Cross, N. G. Housden, R. Kaminska, U. Seger *et al.*, "Supramolecular assemblies underpin turnover of outer membrane proteins in bacteria," *Nature*, vol. 523, no. 7560, pp. 333–336, 2015.

[153] B. Kozlíková, M. Krone, N. Lindow, M. Falk, M. Baaden, D. Baum, I. Viola, J. Parulek, and H. Hege, "Visualization of molecular structure: The state of the art," in *EuroVis STARs*, 2015, pp. 61–81.

[154] E. Krieger. (2003) Yasara. [Online]. Available: www.yasara.org/

[155] E. Krieger and G. Vriend, "Yasara view - molecular graphics for all devices - from smartphones to workstations," *Bioinformatics*, vol. 30, no. 20, pp. 2981–2982, 2014.

[156] D. R. Lipşa, R. S. Laramee, S. J. Cox, J. C. Roberts, R. Walker, M. A. Borkin, and H. Pfister, "Visualization for the physical sciences," vol. 31, no. 8, pp. 2317–2347, 2012.

[157] T. Ertl, M. Krone, S. Kesselheim, K. Scharnowski, G. Reina, and C. Holm, "Visual analysis for space–time aggregation of biomolecular simulations," *Faraday discussions*, vol. 169, pp. 167–178, 2014.

[158] M. Javanainen and H. Martinez-Seara, "Efficient preparation and analysis of membrane and membrane protein systems," *Biochimica et Biophysica Acta (BBA)-Biomembranes*, 2016.

[159] GROMACS. (2009) Xtc library. [Online]. Available: http://www.gromacs.org/Developer_Zone/Programming_Guide/XTC_Library

[160] ColorBrewer. (2009) Colorbrewer 2.0. [Online]. Available: http://colorbrewer2.org/

[161] T. Apajalahti, P. Niemelä, P. N. Govindan, M. S. Miettinen, E. Salonen, S.-J. Marrink, and I. Vattulainen, "Concerted diffusion of lipids in raft-like membranes," *Faraday discussions*, vol. 144, pp. 411–430, 2010.

[162] F. Quemeneur, J. K. Sigurdsson, M. Renner, P. J. Atzberger, P. Bassereau, and D. Lacoste, “Shape matters in protein mobility within membranes,” *Proceedings of the National Academy of Sciences*, vol. 111, no. 14, pp. 5083–5087, 2014.

[163] A. West, B. E. Brummel, A. R. Braun, E. Rhoades, and J. N. Sachs, “Membrane remodeling and mechanics: Experiments and simulations of α-synuclein,” *Biochimica et Biophysica Acta (BBA)-Biomembranes*, vol. 1858, no. 7, pp. 1594–1609, 2016.

[164] V. Gapsys, B. L. de Groot, and R. Briones, “Computational analysis of local membrane properties,” *Journal of computer-aided molecular design*, vol. 27, no. 10, pp. 845–858, 2013.

[165] R. Goldman, “Curvature formulas for implicit curves and surfaces,” *Computer Aided Geometric Design*, vol. 22, no. 7, pp. 632–658, 2005.

[166] E. Falck, T. Róg, M. Karttunen, and I. Vattulainen, “Lateral diffusion in lipid membranes through collective flows,” *Journal of the American Chemical Society*, vol. 130, no. 1, pp. 44–45, 2008.

[167] X. Lin, Z. Li, and A. A. Gorfe, “Reversible effects of peptide concentration and lipid composition on h-ras lipid anchor clustering,” *Biophysical journal*, vol. 109, no. 12, pp. 2467–2470, 2015.

[168] J. E. Goose and M. S. Sansom, “Reduced lateral mobility of lipids and proteins in crowded membranes,” *PLoS Comput Biol*, vol. 9, no. 4, p. e1003033, 2013.

[169] H. I. Ingólfsson, C. Arnarez, X. Periole, and S. J. Marrink, “Computational ’microscopy’ of cellular membranes,” *J Cell Sci*, vol. 129, no. 2, pp. 257–268, 2016.

[170] M. Chavent, B. Lévy, and B. Maigret, “MetaMol: high-quality visualization of molecular skin surface.” *Journal of molecular graphics & modelling*, vol. 27, no. 2, pp. 209–216, Sep. 2008.

[171] B. Antonny, “Mechanisms of Membrane Curvature Sensing,” *Annual review of biochemistry*, vol. 80, no. 1, pp. 101–123, Jul. 2011.

[172] H. Sprong, P. van der Sluijs, and G. van Meer, “How proteins move lipids and lipids move proteins,” *Nature reviews. Molecular cell biology*, vol. 2, no. 7, pp. 504–513, 2001.

[173] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane, “Texmol: Interactive visual exploration of large flexible multi-component molecular complexes,” in *Proceedings of the Conference on Visualization’04*. IEEE Computer Society, 2004, pp. 243–250.

[174] A. Sharma, R. K. Kalia, A. Nakano, and P. Vashishta, “Scalable and portable visualization of large atomistic datasets,” *Computer Physics Communications*, vol. 163, no. 1, pp. 53–64, 2004.

[175] J. Lee, S. Park, and J.-I. Kim, “View-dependent rendering of large-scale molecular models using level of detail,” in *Hybrid Information Technology, 2006. ICHIT’06. International Conference on*, vol. 1. IEEE, 2006, pp. 691–698.

[176] O. D. Lampe, I. Viola, N. Reuter, and H. Hauser, “Two-level approach to efficient visualization of protein dynamics,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1616–1623, 2007.

[177] N. Lindow, D. Baum, and H.-C. Hege, “Interactive rendering of materials and biological structures on atomic and nanoscopic scale,” in *Computer Graphics Forum*, vol. 31, no. 3pt4, 2012, pp. 1325–1334.

[178] M. Krone, J. E. Stone, T. Ertl, and K. Schulten, “Fast visualization of gaussian density surfaces for molecular dynamics and particle system trajectories,” *EuroVis Short Papers*, vol. 2012, pp. 67–71, 2012.

[179] J. R. Weber, “Proteinshader: illustrative rendering of macromolecules,” *BMC Structural Biology*, vol. 9, no. 1, p. 19, 2009.

[180] M. Van Der Zwan, W. Lueks, H. Bekker, and T. Isenberg, “Illustrative molecular visualization with continuous abstraction,” in *Computer Graphics Forum*, vol. 30, no. 3, 2011, pp. 683–690.

[181] J. Parulek, D. Jönsson, T. Ropinski, S. Bruckner, A. Ynnerman, and I. Viola, “Continuous levels-of-detail and visual abstraction for seamless molecular visualization,” in *Computer Graphics Forum*, vol. 33, no. 6. Wiley Online Library, 2014, pp. 276–287.

[182] H. Mohammed, A. K. Al-Awami, J. Beyer, C. Cali, P. Magistretti, H. Pfister, and M. Hadwiger, “Abstractocyte: A visual tool for exploring nanoscale astroglial cells,”

IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 1, pp. 853–861, 2018.

[183] S. J. Marrink and D. P. Tieleman, “Perspective on the martini model,” *Chemical Society Reviews*, vol. 42, no. 16, pp. 6801–6822, 2013.

[184] V. Klema and A. Laub, “The singular value decomposition: Its computation and some applications,” *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, 1980.

[185] D. Kalman, “A singularly valuable decomposition: the svd of a matrix,” *The College Mathematics Journal*, vol. 27, no. 1, pp. 2–23, 1996.

[186] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. De Vries, “The martini force field: coarse grained model for biomolecular simulations,” *The journal of physical chemistry B*, vol. 111, no. 27, pp. 7812–7824, 2007.

[187] M. Chavent, A. Vanel, A. Tek, B. Levy, S. Robert, B. Raffin, and M. Baaden, “Gpu-accelerated atom and dynamic bond visualization using hyperballs: A unified algorithm for balls, sticks, and hyperboloids,” *Journal of Computational Chemistry*, vol. 32, no. 13, pp. 2924–2935, 2011.

[188] H. Samet, *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann, 2006.

[189] A. C. Telea, *Data visualization: principles and practice*. CRC Press, 2014.

[190] M. Harrower and C. A. Brewer, “Colorbrewer. org: an online tool for selecting colour schemes for maps,” *The Cartographic Journal*, vol. 40, no. 1, pp. 27–37, 2003.

[191] D. Frenkel and B. Smit, *Understanding molecular simulation: from algorithms to applications*. Elsevier, 2001, vol. 1.

[192] M. L. Molina, A. M. Giudici, J. A. Poveda, G. Fernández-Ballester, E. Montoya, M. L. Renart, A. M. Fernández, J. A. Encinar, G. Riquelme, A. Morales *et al.*, “Competing lipid-protein and protein-protein interactions determine clustering and gating patterns in the potassium channel from streptomyces lividans (kcsa),” *Journal of Biological Chemistry*, vol. 290, no. 42, pp. 25 745–25 755, 2015.

[193] H. P. Erickson, "Size and shape of protein molecules at the nanometer level determined by sedimentation, gel filtration, and electron microscopy," *Biological procedures online*, vol. 11, no. 1, p. 32, 2009.

[194] G. A. Alvarez and S. L. Franconeri, "How many objects can you track?: Evidence for a resource-limited attentive tracking mechanism," *Journal of vision*, vol. 7, no. 13, pp. 14–14, 2007.

[195] C. J. Thompson, S. Hahn, and M. Oskin, "Using modern graphics architectures for general-purpose computing: a framework and analysis," in *Microarchitecture, 2002.(MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on*. IEEE, 2002, pp. 306–317.

[196] E. B. Goldstein, *Cognitive Psychology: Connecting Mind, Research and Everyday Experience*. Nelson Education, 2014.

[197] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in science & engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[198] D. Shreiner, G. Sellers, J. Kessenich, and B. Licea-Kane, *OpenGL programming guide: The Official guide to learning OpenGL, version 4.3*. Addison-Wesley, 2013.

[199] R. S. Wright Jr, N. Haemel, G. M. Sellers, and B. Lipchak, *OpenGL SuperBible: comprehensive tutorial and reference*. Pearson Education, 2010.

[200] D. Weiskopf, *GPU-Based Interactive Visualization Techniques (Mathematics and Visualization)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[201] B. Gaster, L. Howes, D. R. Kaeli, P. Mistry, and D. Schaa, *Heterogeneous Computing with OpenCL: Revised OpenCL 1.* Newnes, 2012.

[202] A. Munshi, B. Gaster, T. G. Mattson, and D. Ginsburg, *OpenCL programming guide*. Pearson Education, 2011.

[203] M. Scarpino, *OpenCL in Action: How to Accelerate Graphics and Computations*. Manning Publications, Nov. 2011. [Online]. Available: http://amazon.com/o/ASIN/1617290173/

[204] K. Schatz, C. Müller, M. Krone, J. Schneider, G. Reina, and T. Ertl, "Interactive visual exploration of a trillion particles," in *Large Data Analysis and Visualization (LDAV), 2016 IEEE 6th Symposium on*. IEEE, 2016, pp. 56–64.

[205] L. Hrabcak and A. Masserann, "Asynchronous buffer transfers," pp. 391–414, 2012.

[206] M. M. Movania and L. Feng, "Real-time physically-based deformation using transform feedback," in *OpenGL Insights*. AK Peters/CRC Press, 2012, pp. 231–246.

[207] K. Moreland, "A survey of visualization pipelines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 3, pp. 367–378, 2013.

[208] Qt. (2017) Licensing. [Online]. Available: https://www.qt.io/licensing/

[209] ——. (2017) Qt creator ide. [Online]. Available: https://www.qt.io/ide/

[210] R. S. Laramee, "Bob's concise coding conventions (c3)," *Advances in Computer Science and Engineering (ACSE)*, vol. 4, no. 1, pp. 23–26, 2010.

[211] GROMACS. (2009) gro file format. [Online]. Available: http://manual.gromacs.org/online/gro.html

[212] ——. (2009) xtc file format. [Online]. Available: http://manual.gromacs.org/online/xtc.html

[213] A. Langer and K. Kreft, *Standard C++ IOStreams and locales: advanced programmer's guide and reference*. Addison-Wesley Professional, 2000.

[214] Boost. (2017) Boost library. [Online]. Available: http://www.boost.org/

[215] R. Demming and D. J. Duffy, *Introduction to the Boost C++ Libraries; Volume I-Foundations*. Datasim Education BV, 2010.

[216] ——, *Introduction to the Boost C++ Libraries - Volume II - Advanced Libraries*. Datasim Education BV, 2012.

[217] N. CUDA. (2017) Nvidia cuda. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[218] K. Group. (2017) Opencl. [Online]. Available: https://www.khronos.org/opencl/

[219] ——. (2017) Opengl. [Online]. Available: https://www.khronos.org/opengl/

[220] ——. (2017) Vulkan. [Online]. Available: https://www.khronos.org/vulkan/

[221] A. Silberschatz, P. B. Galvin, G. Gagne, and A. Silberschatz, *Operating System Concepts*. Addison-wesley Reading, 1998, vol. 4.

[222] M. Segal, K. Akely, and J. Leech, "The opengl r graphics system: A specification. the kronos group," 2014.

[223] Y. Ukidave, X. Gong, and D. Kaeli, "Performance evaluation and optimization mechanisms for inter-operable graphics and computation on gpus," in *Proceedings of Workshop on General Purpose Processing Using GPUs*. ACM, 2014, p. 37.

[224] M. Lundborg, R. Apostolov, D. Spångberg, A. Gärdenäs, D. Spoel, and E. Lindahl, "An efficient and extensible format, library, and api for binary trajectory data from molecular simulations," *Journal of computational chemistry*, vol. 35, no. 3, pp. 260–269, 2014.

[225] B. Hess, C. Kutzner, D. Van Der Spoel, and E. Lindahl, "Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation," *Journal of chemical theory and computation*, vol. 4, no. 3, pp. 435–447, 2008.

[226] W. Thiel, D. Green, E. Clementi, and G. Corongiu, "In methods and techniques in computational chemistry: Metecc-95," *Eds. E. Clementi, G. Corongiu, STEF, Cagliari*, p. 141, 1995.

[227] Boost. (2017) Boost library. [Online]. Available: http://www.boost.org/doc/libs/1_50_0/libs/iostreams/doc/classes/mapped_file.html

[228] K. Group. (2010) Opengl. [Online]. Available: https://www.khronos.org/registry/OpenCL/sdk/1.1/docs/man/xhtml/EXTENSION.html