Visualisation and Tooling for Interactive Molecular Dynamics in Virtual Reality

By

ALEXANDER DAVID JAMIESON-BINNIE

Department of Chemistry
UNIVERSITY OF BRISTOL

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of DOCTOR OF PHILOSOPHY in the Faculty of Science.

JUNE 2022

Word count: forty-three thousand, eight hundred and one

Abstract

Interactive molecular dynamics (IMD) is a powerful technique which allows user intuition to guide and perturb a simulation. I do not believe that we have taken full advantage of this medium as by merely duplicating existing methods we do not address the new issues nor take advantage of the possibilities.

It is vital to ensure that the system can be rendered efficiently to avoid motion sickness. We must use the latest techniques to allow the visualisation of large systems at a range of magnification levels. I discuss how we achieve this using raycasting imposters, as well as proposing some new primitives that can find use in molecular visualisation.

Sudden visual discontinuities can be immersion-breaking and distracting when using virtual reality. I have developed two techniques to avoid these issues when visualising systems. For periodic systems, I propose a technique to crop atoms within the simulation box to ensure a smooth transition across periodic boundaries. I also correctly depict bonds across periodic boundaries.

Another visual discontinuity occurs when viewing proteins using ribbon diagrams, with discrete flips in the direction the ribbon twists prevalent at the time resolution used in IMD. I address this with a novel double-normal interpolation approach, which when combined with smoothly interpolating the colour and scale yields diagrams that are no longer visually jarring.

In addition to visualisation techniques, we can also exploit the additional freedom afforded in virtual reality to apply interactive forces. By leveraging the rotational degrees of freedom of a handheld virtual reality controller, I propose a new class of interactive forces that can rotate and translate a molecule without internal perturbations. By including damping forces as well, a user can now have precise control over a molecule's trajectory. This technique also extends to more complex coarse-grained systems with asymmetric particles.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .... DATE: ....

Acknowledgements

I would not have reached this point if not for the many talented, wonderful and amazing people that have been at Bristol these past few years. Through all the abuse that our supervisors put us through, it is the camaraderie and companionship that has held us together, and for that I am eternally grateful.

There are so many people that have contributed to my journey, and I'd like to thank a few here.

To Becca Walters, who has basically been my academic sibling through this all — I shall miss many evenings spent watching the dregs of reality TV and how you laughed at all my bad jokes. There was never a dull moment when you were here.

To Callum Bungey, who's chaotic and unbridled energy ensured that for the time we lived together I was not the messiest housemate. Somehow despite everything that life has thrown at you, you remain upbeat and acerbic. Never change.

To Mike O'Connor, who's work ethic, easy-going attitude and love of the craft was and still is a massive influence on me.

To Zack Williams, for being a great housemate and his encyclopedic knowledge of all things Trek and Star Wars.

To Simon Bennie, who always put his neck out for the rest of us, your absence through the last couple of years is sorely noted.

To Joe Crossley-Lewis, I've never met someone quite as affable, charming or fluent in as many European languages. I shall miss having my afternoons punctuated with a 'Salut Salut!'.

To Jonathan Barnoud, our resident rubber duck who was always open to hearing our worries and problems. Thanks for your many helpful comments on this thesis.

To Helen Deeks, who has somehow survived here for this long. Clearly this is because of how politically savvy you are. Thanks for being such a consistent presence, and helping maintain afternoon tea during lockdown.

To Robin Shannon, who though you live a good hour and a half away always made the effort to come to our social occasions.

To Lars Bratholm, who's comfy former office chair I am currently writing this from. Your forthrightness, many bottles of peppermint schnapps and unexpected collection of fancy dress costumes never failed to put a smile on our faces.

To Stephanie Hare, who brought such joy and energy into the CCC. Bristol is a worse place in your absence.

To Harry Stroud, just genuinely one of the nicest and good-hearted people I've met. I hope the shitshow of the last few years does not put a damper on your infectious love of science.

To Oliver Feighan, whose dogged belief that heating a banana would cause it to melt shows good scientists are prepared to consider any outcome.

To Rhos Roebuck Williams, one of the sunniest and least-aggressive people I know. The world needs more optimistic and candid people like you.

To Alex Jones, I will never know how you maintained that constant sense of being both frazzled and chill (through the balance may have shifted since parenthood).

To Luke Lewis-Borrell, for being our token experimentalist friend and always up for discussing your love for Sci-Fi.

To Lex O'Brien, who suffered through more than most and got through to the other side.

To Mark Wonnacott, for being a valued colleague, and for being the source of one of the most surreal stream-of-consciousness twitter's I have ever seen.

To Josh Dunn, for many long conversations when I probably should have been working, and for giving me frankly one of the most horrific gifts I have ever received.

To Dasha Shchepanovska, without you this thesis would be a lot worse and my life a lot emptier.

To my D&D crew — Callum, Dasha, Helen, Jonathan, Lars, Robin and Zack — I thank for many hilarious evenings spent dealing with the no-win moral dilemmas I threw at you.

To my supervisors and advisors who have got me through the PhD: to my former supervisor David Glowacki, I have managed to complete this PhD despite your many shortcomings, and yet would not have had the freedom to work on what I have without you; to Adrian Mulholland, for supporting me through my PhD; and finally to Craig Butts, for helping me get over the finish line and been a great source of advice.

To current and former academics with the department, Andrew Orr-Ewing, Tom Oliver, Neil Allan & Natalie Fey, I'd like to thank for showing not all academia is bad. And to everyone else who I've met — Jillisa Thompson, Simon McKenzie, Silvia Amabilino, Jae Levy, Matt Bain, Rebecca Twidale, Angus Voice, Ella Gale, Ian Shepherd and Lisa May Thomas — thanks for contributing to my time here

To Basile Curchod and your group — Antonio Prlj, Daniel Hollas, Jack Taylor, Lewis Hutton, Yorick Lassmann and Ema Marsili — I'm sorry that I did not get to spend more time with you all. After the general lull in membership of the CCC, it is reassuring to know that there are people like you all to keep the spirit alive.

Outside of Bristol, I'd like to thank my friends from St Andrews — Laras Yuniarto, Finn Smith, Annabel Grout, Billy Brown, Cameron Bathgate, Andrew Woods, Boti Hajdara, Alec Christie and Grania Smith — for their continued friendship and the many evening zoom games that got us through lockdown. And to Jenny Winstone, for her support through the first few years of my PhD.

Finally, I would like to thank my family — my mother Wendy, father Rob, uncle Pete and my Nan Lynn — for being constant support regardless if they have any idea what I do. They'll be pleased to know I finally have a real job.

Contents

List of Figures

List of Tables

List of Abbreviations

AABB Axis-Aligned Bounding Box.

API Application Programming Interface.

BCC Body-Centred Cubic.

BXD Boxed Molecular Dynamics.

COM Centre of Mass.

CPK Corey, Pauling and Koltun.

CPU Central Processing Unit.

CSG Constructive Solid Geometry.

DNA Deoxyribonucleic Acid.

DSL Domain-Specific Language.

DSSP Define Secondary Structure of Proteins.

FPS Frames Per Second.

GPU Graphics Processing Unit.

HTML HyperText Markup Language.

HTTP HyperText Transfer Protocol.

ID Identifier.

IMD Interactive Molecular Dynamics.

IMD-VR Interactive Molecular Dynamics in Virtual Reality.

JSON JavaScript Object Notation.

LED Light-Emitting Diode.

LES Ligand Excluded Surface.

MD Molecular Dynamics.

MSS Molecular Skin Surface.

NMR Nuclear Magnetic Resonance.

NPT Isothermal-Isobaric Ensemble.

NURBS Non-Uniform Rational B-Splines.

NVE Microcanonical Ensemble.

NVT Canonical Ensemble.

OSC Open Sound Control.

PC Personal Computer.

PCA Principal Component Analysis.

PDB Protein Data Bank.

RPC Remote Procedure Call.

SAS Solvent Accessible Surface.

SES Solvent Excluded Surface.

SGMD Self-Guided Molecular Dynamics.

SMD Steered Molecular Dynamics.

SMILES Simplified Molecular-Input Line-Entry System.

SSAO Screen Space Ambient Occlusion.

TCP Transmission Control Protocol.

UDP User Datagram Protocol.

VR Virtual Reality.

XDR External Data Representation.

XML Extensible Markup Language.

YAML YAML Ain't Markup Language.

Chapter 1

Introduction

1.1 What is Molecular Dynamics?

Using computational methods to describe chemical systems has a rich history going back to the 1950s. Advantages of computational methods in comparison to experiment include the reduced cost and the potential insight such simulations provide. However, the two must always coexist, informing each other and leveraging their respective strengths.

Bulk properties of a given molecular system can be determined using computational methods. This is often predicated on sampling configurations of our system, determining the desired property for each of these states, and weighting them by the probability of that state occurring. One method of collecting a representative set of configurations is the **Monte Carlo** method. Specifically, the Metropolis Monte Carlo approach is commonly applied to molecular systems, where random perturbations or moves are made to generate a new state. The probability of accepting this as a valid move is based on the difference in energy from the Boltzmann distribution, which ensures that higher energy states are visited less often. As Monte Carlo is not describing the time evolution of the system, it is limited to calculating static properties.

An alternative approach arose in the 1950s, called **Molecular Dynamics** (MD). Molecular dynamics propagates the system by solving Newton's equations of motion. As molecular dynamics describes a passage through time, it is more suited to questions where dynamical properties based on the trajectories of atoms are of interest.[1] Generally, classical molecular dynamics treats each atom as a point mass generally referred to as a particle. Each particle has position $\{\vec{q}_i\}$ and momentum $\{\vec{p}_i\}$, which are propagated forwards in time in accordance with **Newton's second law**:

$$\vec{F}_i = m_i \frac{\mathrm{d}^2\vec{q}_i}{\mathrm{d}t^2} = \frac{\mathrm{d}\vec{p}_i}{\mathrm{d}t} \tag{1.1}$$

The force on each particle is usually expressible as the gradient of a potential energy function U, which itself is a function of the atomic coordinates:

$$\vec{F_i} = -\nabla_{\vec{r_i}} U(\{\vec{q_i}\}) \tag{1.2}$$

Combining equations 1.1 and 1.2 yields a set of N differential equations, where N is the number of particles. Large molecular systems are not tractable analytically, and hence numerical methods must be used to determine a solution. These equations can be propagated using explicit numerical integration such as velocity Verlet, which determine the positions and velocities at a time $t + \delta t$ from the positions and velocities at time t.

Whilst the underlying techniques of molecular dynamics have emerged largely unchanged through the years, it has instead evolved both in developing new forcefields (ranging from empirical to quantum mechanical to modern machine learning potentials) and in applying enhanced sampling methods for exploring the occurrence of rare events. The increase in computational power over the last century also has lead to molecular simulations becoming viable on personal computers, reducing the barrier to entry and making it a widely adopted technique for scientific discovery.

1.2 What is Virtual Reality?

Virtual reality (VR) describes any technology that can place a user in an immersive, visual digital environment. This is commonly achieved by the suspension of screens in front of the users' eyes in a headset, displaying a virtual environment. By combining this with pose tracking, where the position and orientation of the head control the view through the headset, the user is effectively immersed in a digital world.

Virtual reality has found widespread uses in many disciplines. In medicine, it has been adopted as an effective tool for improving operating room performance[2] and exploring human anatomy.[3,4] In industry, it has been incorporated into product development[5] and drug discovery.[6–8] Additionally, it has seen adoption as an educational tool, where it provides not just the ability to improve spatial reasoning, but improving engagement and immersion.[3] In chemistry, VR has been applied to many areas such as quantum chemistry,[9] NMR ensembles[10] and chemistry education.[11–14] The three-dimensional nature of molecular conformation means virtual reality is an ideal medium to explore the three-dimensional structure of molecules[15–18] and allows immersion in molecular simulations.[19,20]

The tracking of the orientation and position of the headset is generally done in one of two ways. Outside-in tracking uses external sensors positioned in the room to track the location of the headset. Inside-out tracking uses cameras on the outside of the headset, which record the surroundings and use the relative movement between frames to determine the motion of the headset. This newer form of tracking has been made possible by the increased computational power that can be incorporated directly into the headset.

This move towards incorporating more hardware into the headset has also led to a general shift away from tethered headsets. A tethered headset acts as a screen and tracker, whilst the actual calculation of what to display is performed on a separate computer. These headsets must be plugged into the GPU of the user's machine, and hence they are limited in how far the user can move around due to the physical cable. An untethered headset is effectively a small computer, similar to a smartphone. As all computation is shifted to the headset itself, they are more limited in what they can display. They also require some source of power within the headset and hence require charging between uses. However, they provide unparalleled freedom to move around, without the constraint of a cable.

Whilst VR applications often rely on dedicated hardware, the increasing processing power of personal smartphones has made them a widely available and affordable alternative for some VR applications. These headsets work by suspending a user's phone in front of the lenses. By displaying a specific view on the screen and utilising the inbuilt accelerometers, these devices provide access to VR for users without high-end personal computers. Whilst more limited than dedicated hardware, these developments allow virtual reality to be an affordable technology for many people.

An important drawback to consider when using virtual reality is **virtual reality sick-**

ness.[21] This is similar to motion sickness and affects certain people more than others. Generally, to reduce the effects of virtual reality sickness, the refresh rate of the display must be kept above a certain threshold. This emphasises the need for virtual reality applications to avoid expensive calculations and remain responsive to user actions.

In recent months, much has been made of the concept of the *metaverse*. The Meta Platforms, Inc. conglomerate, one of the Big Five technology companies, launched a social VR application known as Horizon Worlds. Given the profound impact that COVID-19 has had on global working practices within the last few years, virtual reality has experienced a resurgence as a method of interacting with other users remotely.

The software in our group has been developed as a framework for performing interactive molecular dynamics in virtual reality.[22–24] This work has been used to teach enzyme catalysis[14], simulate drug binding[25], generate data to train an atomic neural network[26] and explore reaction networks.[27] This breadth of applications shows that virtual reality is applicable to many of the challenges faced by today's scientists.

1.3 Thesis Overview

In Chapter 2, I introduce the work I have done in the development of Narupa iMD and other software. This starts with a brief overview of its development, and the major features and drawbacks of the various versions that have been released. I lay out my contributions to the project, which include the final versions of NarupaXR to Narupa iMD and other projects.

Leading on from this, Chapter 3 is a detailed introduction to rendering for molecular systems, based on my work implementing the visualisations in Narupa iMD. I describe the rendering pipeline, and the technique of raycasting impostors employed by many molecular visualisation applications. I discuss how existing representations such as ball-and-stick are rendered, as well as introduce new methods for rendering dotted and dashed three-dimensional lines, with applications for partial and directed bonds. Finally, I present my novel implementation of periodic systems visualisation, and describe a set of modifications that may be made to existing rendering workflows.

In my work on periodic systems, I argue that visual continuity is important in dynamic visualisation in virtual reality, as it prevents sudden jumps or changes that would distract the user and break immersion. This leads on to my work in Chapter 4, in which I introduce a new method for rendering ribbon representations of secondary structure. This method does not suffer from the dynamic discontinuities observed in existing implementations. I also discuss the history and development of the ribbon diagram, as well as look at how they are implemented in existing packages.

Finally, in Chapter 5 I discuss the background and theory of interactive molecular dynamics (IMD). I describe how interactions are incorporated into the equations of molecular dynamics, and propose improvements to the existing implementation of IMD as it appears in NarupaXR and Narupa iMD. This chapter also discusses how IMD trajectories can be recorded for subsequent analysis, and the information we can extract from this, such as calculating the work performed by our interactions. Finally, I introduce a novel form of a rotation-based interaction, which can be used for fine-grained control of molecular fragments. This method allows manipulation of molecules without distortion, and can be generalised to finite-sized anisotropic particles as well.

Overall, the thesis lays out the current state of the field of interactive molecular dynamics. The overall message of the thesis is that we should not simply duplicate our existing approaches into VR, but take advantage of this transformative medium. The menu and toolbar heavy interfaces which make up most software is ill-suited for VR, where the goal is clean and clear user interfaces, with data presented in a manner that will not overwhelm the user. Just as it has its limitations, the increased degrees of freedom that the controllers provide should be leveraged to provide inputs that would not be possible in a 2D interface. The goal is therefore to understand what works and what doesn't, and design software that takes advantage of VR.

1.4 Contributions

To the benefit of the reader, as some work presented here is collaborative, my contributions may be summarised as:

1. Developing, reviewing and writing a large amount of the Narupa iMD software. This includes contributing to the design and implementation of the communication protocol. I developed and wrote all visualisation code within the virtual reality frontend, as described in Chapters 2 and 3.

2. Developing two novel raycasting shapes for use in visualising molecular systems — a chain of spheres and a wheel shape.

3. Developing a novel approach for visualising periodic systems, cropping shapes as they cross the boundaries and ensuring they smoothly enter from the opposing side.

4. Developing and implementing a novel approach for rendering ribbon diagrams, which does not suffer from previous visual glitches that affect all other protein visualisations.

5. Proposing a new approach for applying rotational forces in interactive molecular dynamics, whilst still allowing internal motion.

1.5 Publications

My work and contributions have lead to the following publications:

Michael B. O'Connor, Simon J. Bennie, Helen M. Deeks, Alexander Jamieson-Binnie, Alex J. Jones, Robin J. Shannon, Rebecca Walters, Thomas J. Mitchell, Adrian J. Mulholland, and David R. Glowacki. "Interactive molecular dynamics in virtual reality from quantum chemistry to drug binding: An open-source multi-person framework". In: *The Journal of Chemical Physics* 150.22 (June 14, 2019). Publisher: American Institute of Physics Inc., p. 220901. ISSN: 0021-9606. DOI: `10.1063/1.5092590`

Rhoslyn Roebuck Williams, Xan Varcoe, Becca R. Glowacki, Ella M. Gale, Alexander Jamieson-Binnie, and David R. Glowacki. "Subtle Sensing: Detecting Differences in the Flexibility of Virtually Simulated Molecular Objects". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems.* New York, NY, USA: ACM, Apr. 25, 2020, pp. 1–8. ISBN: 978-1-4503-6819-3. DOI: `10.1145/3334480.3383026`

Alexander D Jamieson-Binnie, Michael B. O'Connor, Jonathan Barnoud, Mark D. Wonnacott, Simon J. Bennie, and David R. Glowacki. "Narupa iMD: A VR-Enabled Multiplayer Framework for Streaming Interactive Molecular Simulations". In: *ACM SIGGRAPH 2020 Immersive Pavilion.* New York, NY, USA: ACM, Aug. 17, 2020, pp. 1–2. ISBN: 978-1-4503-7968-7. DOI: `10.1145/3388536.3407891`

Alexander D. Jamieson-Binnie and David R. Glowacki. "Visual Continuity of Protein Secondary Structure Rendering: Application to SARS-CoV-2 Mpro in Virtual Reality". In: *Frontiers in Computer Science* 0 (July 12, 2021). Publisher: Frontiers, p. 63. ISSN: 2624-9898. DOI: `10.3389/FCOMP.2021.642172`

Chapter 2

The Narupa Framework

The Narupa framework is the software and protocols developed within the Glowacki group at the University of Bristol to perform interactive molecular dynamics. It consists of two core parts — a front-end application which allows a user to enter virtual reality to view and interact with a simulation; and a back-end server, which interfaces with a molecular dynamics engine and synchronises between the simulation and the front-end clients.

In this chapter, I will discuss the various components that make up the Narupa framework I have contributed to. After a brief overview of the development history of the predecessor of Narupa iMD, I will discuss co-location in virtual reality, comparing it to other approaches that have been tried and explaining how it is achieved.

I will then discuss one of the main purposes of Narupa — communication between simulation and visualisation software. With software for run molecular dynamics bering well established, and a plethora of molecular visualisation programs available, communication between the two is still somewhat limited and underdeveloped. I will discuss file formats commonly used for chemical data and how they are often ill-suited for fast transmission over a network. This will lead onto the development of Narupa iMD's communication framework, using gRPC and protocol buffers to send molecular data to clients, and how shared data is synchronised between users.

After describing how NarupaXR and Narupa iMD propagate and stream their dynamics, I will discuss benchmarks comparing the relative performance of the software defined above. Finally, I will discuss some integration with other packages, as well as the visualisation system employed in the Narupa framework that I developed.

Contributions

The work carried out on Narupa has been a collaborative effort between myself and other colleagues in the Glowacki group. My involvement in the framework started when the project was known as NanoSimbox, before being rebranded as NarupaXR. I was heavily involved in refactoring and tidying the codebase and implementing a trajectory playback feature. I implemented new controller models for the user which more closely mapped their physical counterparts to help new users use them. I also experimented with features that would later make it into later versions, such as using raytracing for visualisation and combining the topology and positions of atoms into the concept of a *frame* of a trajectory.

When we decided that a complete rewrite would be beneficial, I was involved in initial experimentation for a new communication protocol. I wrote a large part of the frontend application, as well as worked on the Python backend. In the frontend, I developed the visualisation stack that renders the system. I also worked on the user interface and communication between the frontend and backend, including addressing several long-term issues with synchronisation. I was involved in the design choices, such as the appearance of the user interface and menus. This new version is generally referred to as the Narupa framework, with the frontend being called Narupa iMD.

After that project had stagnated, I continue to develop and release versions of the software which I have developed alone — the `narupatools` Python package which extends the backend server and a modified version of the Narupa iMD frontend. This was necessary for me to continue developing on the project and to incorporate the work presented in subsequent chapters into software that is openly available to all. To be clear:

1. NarupaXR[a] refers to the first version of Narupa as described by O'Connor et al.[23]. This was a continuation of an existing project formerly known as NanoSimbox, and consists of a frontend written in C$\sharp$ using the Unity game engine, and a custom backend written in C$\sharp$.

2. Narupa iMD[b] refers to the second version of Narupa, described by Jamieson-Binnie et al.[29]. This was a complete rewrite based on ideas developed in NarupaXR. The backend consists of a set of python libraries which interface with existing molecular dynamics libraries.

3. `narupatools`[c] is a set of python libraries which extends the existing `narupa-server` package of Narupa iMD with new features. Servers run through `narupatools` are compatible with Narupa iMD clients.

[a]https://irl.itch.io/narupaxr
[b]https://narupa.readthedocs.io/en/latest/
[c]https://github.com/alexjbinnie/narupatools

2.1 Development of NarupaXR

The origin of NarupaXR lies in the development of danceroom Spectroscopy (dS).[30–32] This project used consumer-grade depth sensors to project the silhouettes of people into a three-dimensional simulation, and use these as potential energy landscapes (as shown in Figure 2.1). Whilst mainly used for art installations, the same technology could be used to manipulate small molecules such as peptides using the user's hands. dS also allowed multiple people to interact with the same simulation simultaneously, whilst being located in the same physical space. With these features, Glowacki et al.[32] postulated that this approach had not just educational applications but could be used to accelerate rare event sampling and find new dynamical pathways.

The development of danceroom Spectroscopy lead directly to the development of NanoSimbox, which used, in place of depth sensors such as the Microsoft Kinect, commercial virtual reality headsets in the form of the HTC Vive (and later HTC Vive Pro).[22] A core feature of this setup is the idea of **co-location**, in which multiple users could interact with the simulation whilst being in the same physical and virtual space. This enabled users to see each other within the context of the simulation, which has both the practical aspect of avoiding collisions and the psychological aspect of inhabiting the same space, which is beneficial for demonstrating and teaching purposes.

The NanoSimbox project was later forked and renamed to NarupaXR. The name Narupa was chosen as it combines the sanskrit word *arūpa*, which means formless, and the prefix *n-* for nano. This name is meant to avoid limiting the scope of the project to something specific such as molecules, and instead encapsulate its generality to visualising scales we cannot observe with our own eyes.

As the project had been continuously developed from danceroom Spectroscopy to Naru-

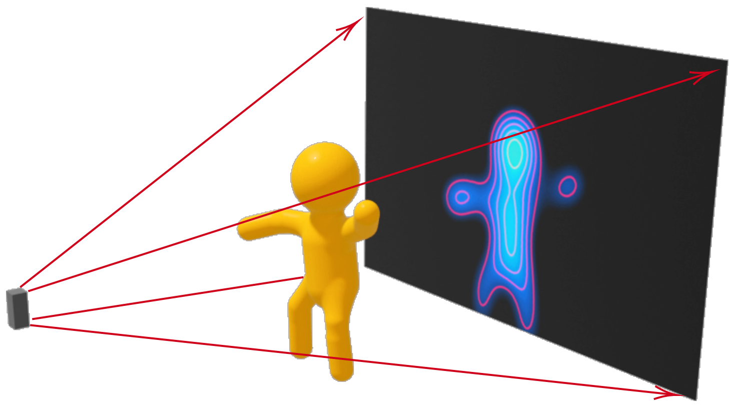


Figure 2.1: Projection of a user's silhouette into a potential energy surface using a depth sensor.

paXR, the NarupaXR codebase was cluttered with techniques and clauses that related to now defunct developments. Another issue that was encountered was the development of the server, which is where users may wish to add custom forces and integrators for their specific system. The server and client are both written in C$^\sharp$, an interpreted language similar to Java. However, it was found that generally scientists were unfamiliar with this language, and it provided a barrier to entry for users to adopt the software. To interface C$^\sharp$ with existing codes (often written in C++), custom bindings also have to be developed.

It was these issues that lead to the development of Narupa iMD, sometimes referred to as Narupa 2 to distinguish it from NarupaXR. This was a complete rewrite of the codebase, based on the insights gained during the previous years of development. Whilst the VR client was similar, using C$^\sharp$ and the game engine Unity, to that of NarupaXR, the backend was completely rewritten in Python. Python is an interpreted language, and has wide adoption in the scientific community. Many software packages also provide a Python wrapper or API, making it much easier to interface Narupa iMD with other software.

Game engines as scientific platforms

The development of molecular visualisation software is similar to the development of video games, with immersive three-dimensional environments and user interaction. Unlike the rendering of animated films, where the production of a film of a few hours can take many months whilst being rendered across a large number of computers, a video game must use commodity hardware found in a users computer to deliver a scene in real time. By addressing issues relating to high-performance rendering, the video game industry solves similar issues that face molecular visualisation programs.[33]

Often, visualisation techniques originate in video games. For example, ambient occlusion is a method which describes the darkening of creases and corners of a scene, due to less ambient light reaching them. Mathematically, calculating ambient occlusion exactly is expensive and generally limited to pre-rendered scenes. However, in the video game Crysis the cheaper approximation of Screen Space Ambient Occlusion (SSAO) was developed.[34] This technique has since found widespread adoption in both video games and in molecular visualisation, where it provides important depth information to help users understand the three-dimensional structure of their molecule.[35]

Writing a video game involves many parts, including interfacing with the computer's graphics API, reading in assets and representing the scene. It is for this reason that game engines exist, which provide a foundation which builds on the low-level code required to create a game and allows developers to focus on their application-specific concerns. Using a game engine also allows a developer to target a wide range of platforms — whilst different operating systems and hardware may have different capabilities and interfaces, a game engine can abstract this into a single generic interface that is exposed to the developer.

In recent years, scientific software has increasingly adopted the use of game engines such as Unity and Unreal. These programs provide comprehensive editors for laying out and creating scenes graphically rather than purely through code. They also benefit from a wide adoption in game development, which has lead to a thriving ecosystem of tutorials and assets. Adopting an engine such as Unity therefore has a lower barrier of entry than learning low-level graphics and user interface libraries. Another advantage is that game engines often allow programming in a higher-level language, such as Python or C$^\sharp$, whilst the lower-level code driving the engine is often written in C++.

When developing software for virtual reality, using a game engine helps support a range of different headset hardware and APIs. The Unity game engine has seen an especially wide adoption in molecular visualisation, especially when augmented or virtual reality compatibility is required. Examples of software that uses Unity include Narupa[23,29], UnityMol[33,36–40], Nanome[18], Molecular Rift[41] MOF-VR[42], CellexalVR[43], Peppy[44], BioVR[45], MDV[46], SimView[47] and InteraChem[48]. Meanwhile, the Unreal engine has seen less widespread adoption.

As an alternative to developing applications to run on a user's computer, molecular software may be run within a web browser. Instead of requiring an entire game engine, applications running in the browser can leverage a wide range of Javascript libraries, such as Three.js. With modern libraries such as WebXR there has been an increasing number of web-based virtual reality applications for chemistry, including ProteinVR[49], VRMol[50], BioSim$^{\mathrm{AR}}$[51] and MoleculARWeb[52].

When compared to a standalone software, web-based applications tend to have a smaller distribution size. They are not compiled — being mostly Javascript, they are easily modifiable and extensible. This also allows them to be easily distributed and embedded in other websites. For examples, viewers such as JSmol, ngl and Mol* have found widespread use for embedding three-dimensional molecular structures in a range of web resources.

2.2 Co-location in Virtual Reality

A unique aspect of Narupa's approach to IMD-VR is its support for **co-location**.[23,29] It allows users who were located in the same physical space to experience the same virtual world simultaneously, by utilising multiple headsets located within the same room.

It has been observed that the exclusivity of utilising a single headset whilst multiple people are physically present results in these people being excluded from participating in the experience, often being relegated to observing what the headset-wearing user is doing through a two-dimensional screen. Previous work to address this issue includes CoVR[53], which displays the view of a VR user using a projector, allowing others to view exactly what the user sees. Other approaches include *ShareVR*[54], which allowed a user in virtual reality to interact with others who are in the same physical space but are not wearing a headset, or *FaceDisplay*[55], which used touch-sensitive displays attached externally to the headset to allow external users to interact.

The issue of external observers not being able to experiance the virtual world can be addressed by the *ReverseCAVE*[56]. Here, translucent screens can be used to project the virtual environment around the user, allowing external bystanders to understand where they are relative to what they are experiencing. However, none of these approaches are as immersive as placing the observers within the same virtual space and allowing them to participate as equals.

Multiuser experiences in virtual reality often makes the assumption that the users are located in different physical spaces. Therefore, there is no need for a correlation between their relative physical positions and their relative positions within a virtual world. However, when multiple users wear headsets in the same room and interact in the same virtual world, it is important that the two positions correlate. For example, if a user is standing to the left of another, they should also appear at this position within the virtual world. This helps avoid collisions in the physical world. Another consideration is that if the users are speaking in real life, the audio will come from the same direction as would be expected by the position of their virtual avatars.

Co-location in NarupaXR and Narupa iMD is implemented using the lighthouse tracking system of the SteamVR ecosystem, as it allows the tracking of multiple headsets simultaneously. As each headset agrees on an upwards direction, as well as the location of the two lighthouses, we are able to correctly position the headsets relative to one another.

A similar approach to co-location has been explored at Purdue University.[57] This approach used a custom-made visualisation platform written in Unity, and used multiple Oculus Quest headsets. As described previously, the Oculus Quest headset uses an inside-out tracking, where cameras on the headset determine the orientation rather than external sensors. Early approaches at Purdue used various external trackers attached to the headset to synchronise the headsets within one virtual space. However, by marking a clear origin and forwards direction on the floor, each headset can be calibrated in the same manner. This achieves the same result as Narupa's co-location, but it requires careful calibration of each headset.

There are several different spaces (referring to a coordinate system of an origin and three axes) that must be considered when synchronising users:

1. Real space $\mathcal{R}$. This describes the actual location of users within the physical space they occupy.
2. Calibration space $\mathcal{C}_i$. Each user's headset is calibrated in a manner specific to that headset. This calibration step is performed before using the headset and determines the floor level and direction that the headset considers to be forwards. If the calibration is performed correctly, the floor height and upwards directions with agree for all users, and be aligned with the real physical space $\mathcal{R}$.
3. Virtual space $\mathcal{V}$. The virtual space is the coordinate space in which the virtual world will be defined.

The transformation (represented as a matrix) which transforms points from one space $\mathcal{A}$ to another $\mathcal{B}$ will be written as $T[\mathcal{A} \rightarrow \mathcal{B}]$.

When multiple people are viewing the same system (either remotely or in the same room) and co-location is not in use, a situation as shown in Figure 2.2 will occur. The virtual space $\mathcal{V}$ is positioned relative to each user's calibration frame $\mathcal{C}_i$, and hence will be located in different positions in real space. This means when avatars are synchronised between users, they will appear at somewhat arbitrary locations and may overlay with each other.

Co-location is achieved in Narupa iMD by assuming that all headsets are tracked by the same lighthouse base stations. As each of these base stations has a unique ID that is transmitted

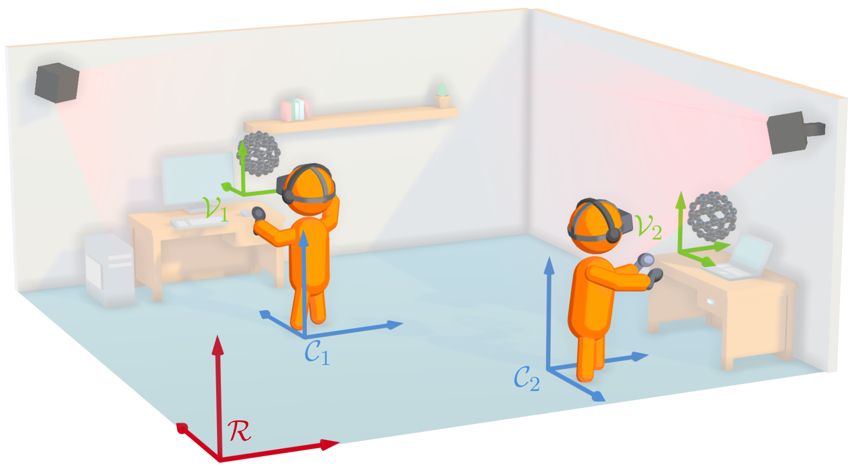


Figure 2.2: Two users occupying the same physical space $\mathcal{R}$, but interacting with different virtual spaces $\mathcal{V}_1$ and $\mathcal{V}_2$.

automatically to each headset, all users can agree on the location of the two sensors with the lowest IDs. By taking the midpoint of the two lighthouse locations and projecting it onto the floor, all users can agree on a single point in real space, and know where this is relative to their calibrated space. By aligning this coordinate system with the lighthouses, and locating the virtual space here, the virtual space of each user are now aligned in real space (as shown in Figure 2.3. This achieves the goal of co-location — aligning the virtual space even when the calibration space of each headset differs. The common reference space of $\mathcal{V}$ allows the users avatars to be synchronised in a manner in which they are aligned with their real-life locations.

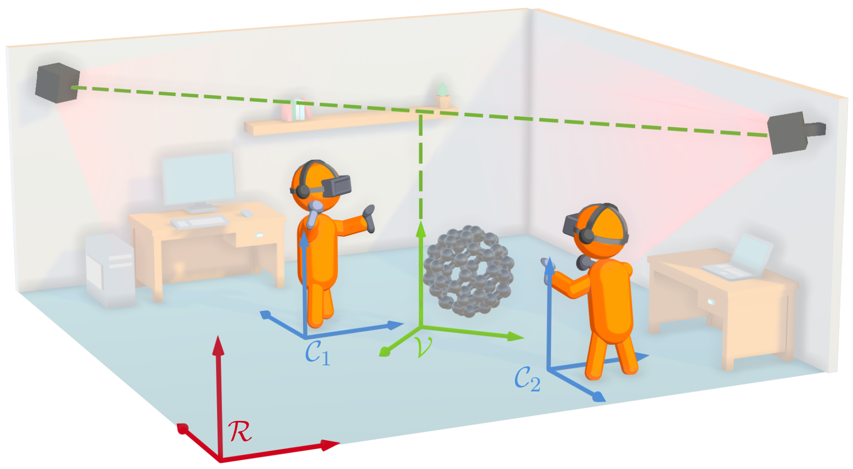


Figure 2.3: Two users occupying the same physical space $\mathcal{R}$ and interacting with the same virtual space $\mathcal{V}$.

2.3 Communication

Communication is a vital part of developing interconnected software, from communicating between two programs running on the same PC to transfering data between different computers that may be on the same network or on a different continent. The communication between two pieces of software can be broken into two parts:

1. The format in which the data is transmitted between the software. Converting from a native format used by the software to the form used for communication is referred to as **serialisation**, whilst converting back is referred to as **deserialisation**.
2. The method in which this data is transferred from one piece of software to another, whether they are on the same computer or separated.

Chemical file formats

When sending sending files between different software, or sharing files to other people across the internet, it has to be agreed what format these files with take. Over the years, many file formats have been developed to communicate molecular data.

The XYZ file format is one of the most widely utilised formats, though it lacks an exactly defined standard. It is a simple format, usually storing just atomic positions and elements. This makes it suitable for storing trajectories of small molecules, but usually needs to be accompanied by another file which better describes the structure of the molecule. These additional data describing the groupings and bonds within a chemical structure is referred to as the **topology**. The topology of a molecule contains both information on the bonds within a molecule, as well as information such as atom names or residues.

For protein structures, the Protein Data Bank (PDB) format has been widely adopted. In the PDB format, each line is preceded by a keyword such as `TITLE` or `ATOM`. The atomic data is stored in a *fixed column* manner, with only a specific number of characters in each line allocated for each specific piece of data. This applies certain limitations upon the format, such as a limit on the number of atoms it can store or the ability to store additional data. However, it has been widely adopted and is supported by most software handling protein molecules. As it is developed for protein structures, it contains topological information such as residues, secondary structure assignments and bond information.

Software packages that actually run simulations have generally used bespoke file formats specific to their use cases, such as the `.gro` file format of GROMACS[58] or the `.crd` and `.psf` formats of CHARMM[59]. This has lead to the proliferation of many different ways of describing molecular systems, each with their own pros and cons. It therefore necessitates that software have a wide range of importers and exporters, or users having to use software such as Open Babel[60] or MDAnalysis[61,62] to convert between them.

Human-readable and binary formats

These file formats such as PDB and XYZ are unique to molecular data and use specific syntax. This means different applications must implement specific importers and exporters that must both understand the raw data conveyed within the files, but also how it is laid out within these files. This can be addressed by file formats that use a markup language. These are well-described methods for laying out arbitrary data in a file, without dictating exactly what data should be stored. The most common of these includes JavaScript Object Notation (JSON), Extensible Markup Language (XML) and YAML, examples of which can be found in Figure 2.4. These are all examples of human-readable formats that can be used to serialise data.

A classic example of a human-readable format that is transferred between computers is HTML, which is stored in a format very similar to XML. This describes the format in which web pages are defined, using a standard set of tags and nodes to describe various features such as links and headers. It is this file that is distributed across the internet when you visit a website, and the browser acts as a front end that 'deserialises' this plain text file and interprets it as an interactive viewable webpage that is presented to the user.

Human-readable formats such as these are well suited if they are to be modified directly by the user, or where the size is not an issue. However, when sending large amounts of data across a network, smaller binary formats are preferred. In our web example, whilst the general layout of a web page is send as a human-readable HTML file, a video on a webpage will be sent using a highly compressed binary format.

Each common letter or character can be represented on a computer using a single **byte** — a group of eight **bits** each of which may be a 0 or a 1. A human readable 'spells out' the data in writing, in a similar manner to writing it down on a page. However, computers have more

```xml
<Family>
    <Person>
        <Name>Alice</Name>
        <Age>21</Age>
    </Person>
    <Person>
        <Name>Bob</Name>
        <Age>24</Age>
    </Person>
    <Person>
        <Name>Eve</Name>
        <Age>18</Age>
    </Person>
</Family>
```

(a) XML

```json
{
    "family": [
        {
            "name": "Alice",
            "age": 21
        },
        {
            "name": "Bob",
            "age": 24
        },
        {
            "name": "Eve",
            "age": 18
        }
    ]
}
```

(b) JSON

```yaml
--- # family
- name: Alice
  age: 21
- name: Bob
  age: 24
- name: Eve
  age: 18
```

(c) YAML

Figure 2.4: Comparison of markup languages for representing arbitrary data.

optimal ways of storing numerical data. For example, storing an integer on most platforms requires 4 bytes, but can represent a range of values from $-2\,147\,483\,648$ to $2\,147\,483\,648$. If this were to be written out in text form, 4 bytes could only store numbers from -999 to 9999. More importantly, when performing calculations and otherwise processing data, computers act on these internal binary representations of numbers rather than the text form.

This concept underpins binary serialisation formats, where values are stored not as a sequence of characters but as a well-defined sequence of bytes. This tends to be closer to how software already stores these values internally, and hence it is faster to read and write files of this type. As different computer systems can interpret data differently, binary formats must be very specific in how they store certain values. For example, when storing integers there is the concept of endianness, which describes the order in which the bytes which describe the integer are sorted (with big-endianness systems storing the most most significant byte first and small-endianness systems storing it last).

Transport of data

Regardless of the data exchange format used, to a computer all data fundamentally consists of a sequence of bytes, each consisting of 8 digits that may be 0 or 1. The communication of this exact sequence of bytes from one computer to another involves defining some method of communication. The specification of how exactly data should be interpreted at each end is referred to as a **protocol**. Defining how data is transported between computers is the job of the **transport layer**, with the two most common transport protocols being TCP and UDP. TCP (Transmission Control Protocol) defines communication based on connections, and can transmit streams of bytes between two computers in a reliable manner. Reliable in this sense means that if somehow a part of the message is lost in transmission, then the sender is informed and can resend it. This is in contrast to UDP (User Datagram Protocol), which does not have any sort of guarantee that the data will arrive at the destination. This makes UDP better suited for audio and video, where if a small amount of data is lost, it is not important as it is soon to be replaced by subsequent data regardless.

Dealing directly with these protocols involves writing code that has to deal with the exact meaning of every byte. This is usually hidden behind the higher-level application protocol. A prime example of this higher-level protocol is HTTP (Hyper Text Transfer Protocol), over which a vast majority of the internet is communicated over. This protocol usually employs TCP, and helps separate the user from the fundamentals of communications and focus on high-level application development. In the context of internet browsing, we have a server that sends data (in the form of web pages written in HTML) to a user's browser via the communication protocol HTTP, which uses TCP to communicate the bytes of the HTML page to the browser.

Another example of a higher-level protocol are WebSockets, which again use TCP as a low-level protocol to send messages between clients. As TCP transmits chunks of bytes, the

WebSocket layer handles the disassembly of a message to be sent via TCP and its reassembly into a message on the recieving end. The predominant difference between HTTP and WebSockets is that HTTP is mainly suited to a request/response based system, where no persistent connection is maintined, whilst WebSockets are good for event-driven communication between two clients. However, recent moves towards HTTP/2 and HTTP/3 are introducing new features to HTTP to remedy this.

Communication for interactive dynamics

Some of the earliest development for streaming molecular simulations came in the set of tools known as MDScope.[63] This linked together the molecular visualisation package VMD[64] with the molecular dynamics software NAMD[65], using a custom communication library known as MDComm. MDComm uses eXternal Data Representation (XDR), which is an example of a binary serialisation format, and the TCP standard to transmit this data between computers.

The next generation of communication between VMD and NAMD came with the replacement of MDComm by a custom protocol by Stone, Gullingsrud, and Schulten[66], commonly referred to as the VMD-IMD protocol. This aimed to allow highly efficient communication between server and client, as latency is a major issue when performing interactive molecular dynamics. This protocol has seen adoption in a wide range of molecular dynamics packages, such as NAMD, GROMACS[58], LAMMPS[67] and HOOMD[68]. In addition to VMD, the Python application Tios[69] allowed simulations to be streamed using this protocol.

The IMD protocol defines various messages, including sending energies or communicating new forces. However, it is very prescriptive and only supports receiving coordinates. Data such as molecular topology and periodic box dimensions are not transmitted, and hence must be provided separately. It is also limited to simulations where the number of atoms does not change. However, it is still currently the *de facto* standard for interactive molecular simulations, finding use in other inter-software communication packages such as MDDriver.[70]

MDsrv[71] uses the popular web-based NGL viewer to show trajectories streamed across the web. However, it does not use any special binary formats, but simply transmits standard chemical file formats over an internet connection to be displayed by NGL viewer. HTMoL[72] acts in a similar manner, however it only supports a small subset of MD formats (XTC, DCD and NC). However, these are specifically binary formats, and hence are well suited for streaming data across a network. These web-based approaches are mainly focused on streaming pre-existing trajectories which have already been written to file, rather than streaming data from simulations that are currently taking place.

NarupaXR uses a server that performs the integration of each MD timestep itself and relies on external engines for the calculation of forces. The communication between the VR client and server was an in-house solution referred to as the Nano API. The Nano API supports two connection types — TCP and WebSockets. These relatively low-level communication protocols mean that a lot of the complexities of serialising the data and communicating it also had to be developed and incorporated into the codebase. Using a bespoke approach such as this as the disadvantage of being harder for new developers to understand, compared to utilising a well-defined protocol.

The streaming of data such as atomic positions was achieved through a set of streams — a continuous open communication from the server to the client along which arbitrary bytes could be sent. As both the client and the server agreed on the meaning of these bytes, this allowed the efficient transmission of a large amount of data. To minimise the amount of data, the server converted each numeric value from a 4 byte representation (with around 6 to 9 significant figures) to a 2 byte representation (with around 3 to 5 significant figures). Whilst this loses precision, it is an acceptable trade off for halving the amount of data transmitted.

Additional communication within NarupaXR is performed using the Open Sound Control[73] (OSC) protocol, originally used for communication between musical devices. This could be used to both execute functions on the server from the client, such as adding a new restraint, or be broadcast from the server to all clients. This second approach could be used to inform clients when certain events occurred, such as a bond breaking. These OSC commands are used to implement common playback commands that the client can control, such as playing and pausing the interaction. For example, the `/play` command when sent by a client will resume a simulation if it has previously been paused.

When the development of Narupa iMD took place, many discussions were had about how to implement the communication between the server and the clients. Desirable properties were an extensible format that allowed transmitting arbitrary data, a small packet size to minimise bandwidth and latency when broadcasting, and an avoidance of low-level custom implementations previously used. Adopting existing packages to handle the networking gives more time to the developers to focus on the scientific side of the software, makes the code more reliable and makes it easier for new developers to understand how it works.

gRPC

For the communication side of Narupa iMD, we decided to use gRPC[74]. This is a high-level library based on HTTP/2 that handles establishing connections and transmitting data in a bidirectional manner. It supports a wide variety of programming languages, which makes interoperability between various programs within the Narupa framework easier.

Although gRPC can support any serialisation format, we chose to use the default format of protocol buffers, or protobuf. This is a binary serialisation format developed by Google, and used extensively in web services. We settled on this format due to its wealth of documentation, its strong cross-language support and its flexible schema. Protobuf works on the basis of 'messages', which are predefined in specific protocol files. These messages define what kinds of data are held and in what order. By adopting a message-based approach, if two applications know the message definition, they know for example that the second number in a message is a 'count' variable, without having to send the word 'count' in the message itself to identify it. This reduces the size of a complex message compared to sending arbitary data. Protocol buffers have found some use within computational chemistry, with the TeraChem electronic structure package using them for communication between loosely coupled computing resources.[75]

The RPC in the name gRPC refers to it being a Remote Procedure Call library. This concept comes from the idea that instead of just sending arbitrary messages, the protocol must also define functions or procedures that can be invoked. In the simplest sense, a simple procedure call takes some arguments (in the form of a message) and returns one or more results (again, in the form of a single message). Therefore, the messages are just part of the communication, and this approach can be seen as calling a function on a remote computer, with the result returned at some later point. This is a 'request-response' system.

The advantage of gRPC is that it also allows 'streaming' calls, which allow more than one message to be sent or received. This breaks down each function that can be defined into four categories:

1. Simple request-response, with one request being answered with one (optional) response.
2. Client-streaming calls, where the clients send repeated messages to the server and the server only responds when it's finished. This was used for sending user interactions to the server, before being superseded by a bidirectional stream.
3. Server-streaming calls, where the server sends repeated messages to the client. This encapsulates another pattern known as the 'publisher-receiver' pattern and is used to send the simulation data to the clients (where the server does not want any response back).
4. Bidirectional streaming calls, where both server and clients can send multiple messages to each other. This is the most general case and implements the synchronisation of data in the Narupa framework. The clients can update values and send them to the server, and the server sends everyone these values so all clients are in sync.

Using gRPC has allowed us to focus on developing the software itself, as it handles all communication, establishing servers and maintaining connections. As it uses the HTTP/2 protocol, it can utilise the same port as regular internet traffic and hence can work without configuring network hardware. It also has features designed for authentication, encryption and compression, which can be leveraged when the use case requires it.

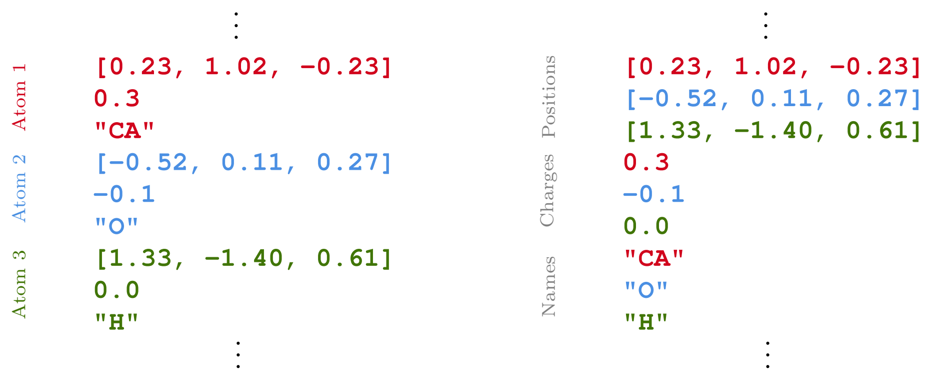
Frame data representation

The most common piece of data we wish to communicate using the Narupa protocol is a snapshot of a molecular trajectory. This draws a parallel between our use case and the concept of a video — a series of images spaced through time. Correspondingly, we refer to these individual snapshots of a system as a **frame**, allowing the use of corresponding jargon such as frame rate to describe the rate at which molecular frames are produced.

Unlike a video, where each frame is a single image consisting of a grid of coloured pixels, a molecular frame consists of different kinds of data. These frames are an example of a record or struct, a basic data structure consisting of one or more fields. For example, each atom may have fields including its position, the index of the residue it belongs to or its atomic charge. There are two general manners in which a set of records can be stored — as an *array of structures* (AoS) or a *structure of arrays* (SoA).

In the case of an array of structures, each atom is a separate data structure containing its properties, such as position and charge. Therefore, data relating to each atom are located close together in memory. In contrast, a structure of arrays (sometimes known as parallel arrays) stores separate arrays for each property. In this case, data of the same type (such as positions) are located close together in memory. Figure 2.5 illustrates this point.

The AoS approach is well suited if certain atoms have fields which other atoms lack. It is also more suitable where atoms are added or removed from the simulation, as only one array needs to be modified. Often, chemical file formats are laid out in this manner. For example, each line of a PDB file describes an atom and all its properties. However, the SoA approach is generally better for molecular simulations, where the number of atoms is constant. It is often the one encountered in simulation engines such as ASE[76]. The SoA approach allows the data to be easily filtered before sending to the client, by only sending relevant data when required.



(a) Array of Structures (b) Structure of Arrays

Figure 2.5: Comparison of structure of arrays vs. arrays of structure for storing molecular data.

Therefore, the Narupa framework adopts an SoA approach when describing a snapshot of a molecular system. This data structure is known as a `FrameData`, and consists of a set of single values (such as potential energy) and arrays (such as the atomic positions or bond array). Theses fields are uniquely identified by a name, which is defined by the protocol. Table 2.1 gives some examples of the fields that are defined in the Narupa framework, along with the data type associated with them. As these frames consist of arbitrary data, it is very easy for users to append new per-atom data to send to the client. In comparison, each field (such as positions or charges) was synchronised as a separate pre-defined stream in NarupaXR, limiting its extensibility to new kinds of simulations.

Protocol buffers contains a predefined data structure that is well suited for defining arbitrary data, called a `struct`. This stored arbitrary key-value pairs, in a similar manner to a dictionary in Python. Protocol buffers also have the ability to store lists. However, these lists are untyped — they can store arbitrary data such as strings, integers and other lists within the same list. When serialising data consisting of the same data type, such as the $3N$ coordinates of a system, each number must be prefixed by a byte indicating the following entry should be interpreted as a float. This both increases the size of the serialised message, and requires slowly deserialising the data item by item to reproduce the original list.

To address this, I developed the protocol used with the Narupa framework for storing and transmitting frames of a molecular simulation. Each frame consists of two parts — arbitrary key-value pairs that can hold arbitrary data as defined above, and a special set of key-value pairs which are optimised for storing repeated data. The difference between the two when storing three floats is illustrated in Figure 2.6. These array fields can store repeated floating-point numbers, integers and strings without the additional bytes indicating their type. This reduces the size of the messages transmitted (allowing larger systems) and allowed optimised reading and writing of these data.

Key	Meaning	Type
`particle.count`	Number of particles N	Single integer
`particle.positions`	x, y, z coordinates of each particle	Array of $3N$ floats
`particle.elements`	Atomic numbers of each particle	Array of N integers
`particle.residues`	Index of the residue the atom belongs to, in the corresponding array	Array of N integers
`residue.count`	Number of residues N_r	Single integer
`residue.name`	Name of each residue	Array of N_r text strings
`bond.count`	Number of bonds N_b	Single integer
`bond.pairs`	Pairs of particle indices	Array of $2N_b$ integers
`energy.kinetic`	Kinetic energy	Single floats
`energy.potential`	Potential energy	Single floats

Table 2.1: Examples of data that can be found in a Narupa iMD `FrameData`, and their corresponding keys.

2.4 Synchronising Data

The gRPC framework is well suited for a *request-response* pattern, in which a client sends a request to a server, which responds with a single reply. Another common pattern that is relevant for streaming molecular data is the *publish-subscribe* pattern. Here, a publisher produces messages without regarding exactly who requires them. Clients may then subscribe to a specific subset of these messages. When a publisher produces a new message, all current subscribers are sent the message.

This approach can be implemented using the gRPC server-side streaming paradigm. This is similar to *request-response*, however the server may respond with multiple messages. These messages may be separated by a large amount of time, and the client can act while waiting for additional messages. This is used for the frame subscription portion of the Narupa protocol. A client may register an interest in receiving frames of the molecular simulation using the `SubscribeLatestFrames` command. The server first replies by sending a `FrameData` which contains all information about the current system. Subsequent messages inform the client of updates to this using additional `FrameData`.

Of the data available, most of it does not vary with time. For example, for a fixed molecular system, values such as the atomic elements, bonding pairs or residue names do not change over the course of the simulations. Streaming all this information whenever the atomic positions change would be a waste of network bandwidth and limit the size of the simulation that can be simulated.

To address this, the updates that the server sends the client after the initial reply are partial changes, describing only the data which has changed since the last message. This is an approach commonly used in web interfaces, using the `PATCH` method of the HTTP protocol. In the example of a molecular simulation, subsequent messages usually only contain the new atomic positions computed by the molecular simulation engine. This is performed by key however — if a single position changes, all the atomic positions must be sent again. This could

$$\texttt{0a 18 }\underbrace{\texttt{00 00 00 00 00 00 f0 3f}}\ \underbrace{\texttt{00 00 00 00 00 00 00 40}}$$

$$\underbrace{\texttt{00 00 00 00 00 00 08 40}}$$

$$\texttt{0a 21 0a 09 11 }\underbrace{\texttt{00 00 00 00 00 00 f0 3f}}\texttt{ 0a 09 11}$$

$$\underbrace{\texttt{00 00 00 00 00 00 00 40}}\texttt{ 0a 09 11 }\underbrace{\texttt{00 00 00 00 00 00 08 40}}$$

Figure 2.6: Comparison of encoding three floating-point number (each of 8 bytes), using a standard protobuf list (bottom) and a specific repeated field of floats (top).

be improved in the future by allowing more granular specification of what has changed.

This process occurs on the server between the molecular simulation engine and the frame publisher. Here, the simulation engine informs the publisher that some data has changed and hence a new `FrameData` must be sent to the clients. It is hence the simulation engine's job to track what data has changed and hence what data must be sent to the publisher. When new data is received by the publisher, it first merges it onto a `FrameData` which represents all previously received `FrameData`. This means the publisher has an object which represents the merging of all simulation data received so far, and hence has a complete snapshot of the system at this point. It is this object which is first sent to a new client.

On the client side, the received `FrameData` is merged onto the existing one. Using these methods, the servers can keep the clients informed about the current state of the simulation whilst minimising the amount of data transmitted. The process of merging does not contain a history of the changes — by merging all the differences together, each client has a copy of the `FrameData` that represents the current state of the system. Figure 2.7 illustrates how synchronisation between the server and client is achieved.

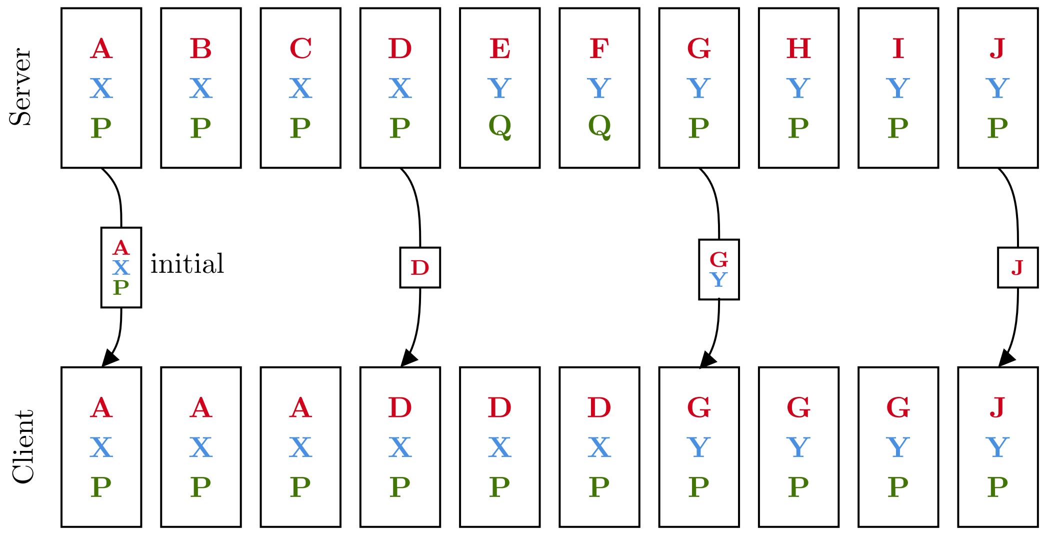


Figure 2.7: Synchronisation of frame data between a server and a client. The server's state changes according to the dynamics, while periodically the fields (denoted by different colours) that have changed (denoted by different letters) are sent to the client to merged into the client's frame. Note that the red field changes faster than the frames are produced, and hence certain values are never sent to the client. Similarly, the green field changes to Q temporarily, but this is never seen by the clients as it occurs between frames.

Shared State

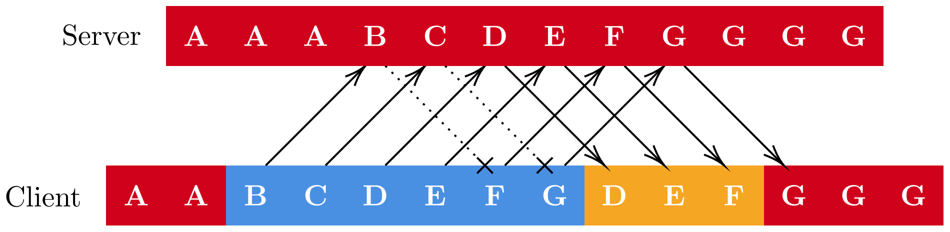
Transmitting the state of the molecular simulation is unidirectional — the server determines the state of the simulation and transmits this to the clients. However, other data needs to be shared between the server and the clients which is not associated directly with the simulation. This includes the positions of the users' avatars (their headsets and controllers), interactions applied to the simulation by the user, and the position of the simulation box relative to the virtual space. This data is bidirectional — it may be influenced by either a client or a server.

All this data is grouped into a single data structure known as the **shared state** in Narupa. This is effectively a dictionary of arbitrary key-value pairs which is synchronised between the server and the clients. The server shares this to the clients in a similar manner to how molecular frames are transmitted — it groups together all changes since the last update and sends these to all clients. However, the use of a bidirectional stream allows clients to make modifications themselves.

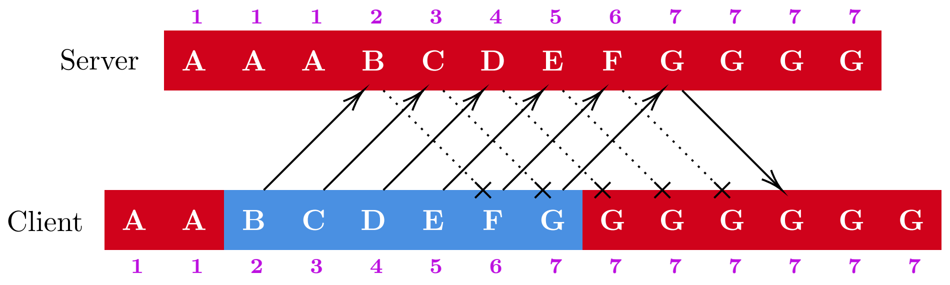
Usually, only one party (either a user or the server) modify a certain value. For example, only a user writes to its own avatar location and its own interactions, whilst other parties merely read these values. However, the simulation box is modifiable by any user, by using their controller to grab the box and manoeuvre it. As Narupa iMD is a multiple-user experience, there has to be an approach that prevents race conditions where more than one user attempts to move the box at the same time. This is achieved using a feature of the shared state known as **locking**, where a client may request that one or more keys are locked. If the key is unlocked, then that client is granted an exclusive right to modify that value until the client requests it is unlocked.

One issue which I addressed was a glitch which occurred when a user was moving the simulation box. As you move the simulation box, its position is tied to the position of your controller and this is sent to the server to ensure all users see the box in the same position. When the user had finished moving the box and let go, the simulation box was teleporting briefly back to a previous position before moving back to where the user has last positioned it. This is because of the finite time it takes for messages to go to and from the server. When the user was releasing the box, the server was only just sending them where the box has been a few moments prior.

To address this, I implemented a protocol where each time the user sends an update to the server, they also send a monotonically increasing identifier called an *update count*. Therefore, they can ignore updates to the box's position if they are older than the update count when the box was last sent. Figure 2.8 illustrates this technique.



(a) After changing a value (blue region), the client observes some of the previous values (orange region) due to the delay in communication.



(b) By tracking update counts when changing values and synchronising them as value, the client can ignore the server until it sends back the last update count the client sent.

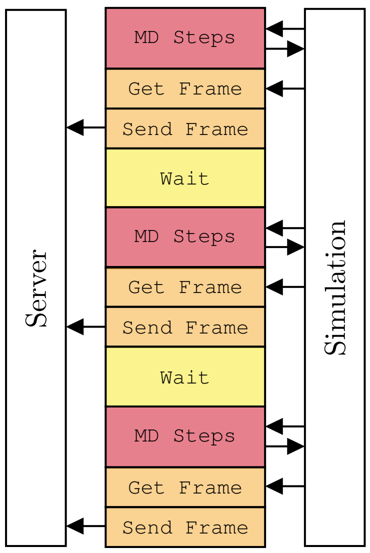
Figure 2.8: Comparison of synchronising a value between the client and server with and without update count tracking.

2.5 Streaming Molecular Dynamics

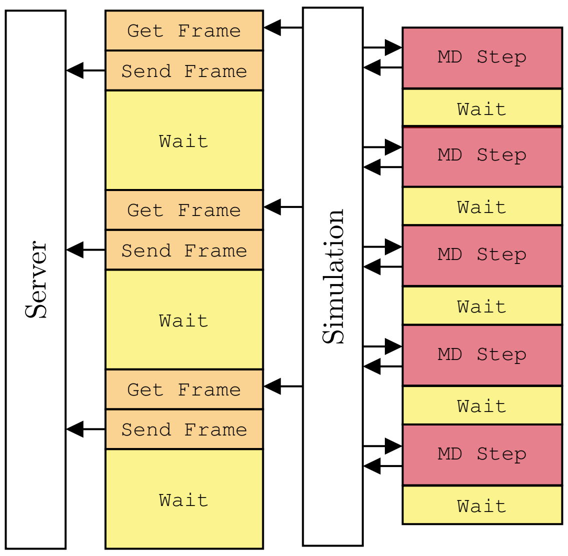
Molecular dynamics consists of the propagation of a system forwards in time in discrete steps, governed by a dynamical equation. Running molecular dynamics on a computer therefore consists of repeated steps forward in time, each advancing the system forward by some timestep dt. Often, when trajectories are recorded the simulation is interrupted at an interval, for example every 100 steps, and the current state of the system is written to disk.

It is this approach which was used in NarupaXR, and subsequently carried over to Narupa iMD. Here, the user specifies a *frame interval* — how many steps of the simulation should pass between publishing a molecular dynamics snapshot to the VR clients. By choosing a good value such that an adequate amount of frames are produced, this results in a simulation that runs as fast as possible, while streaming the current state of the simulation to any observers at an acceptable frame rate.

However, there are limitations to this approach. Choosing a frame interval relies on prior knowledge of how fast the simulation will run, something which is heavily dependent on the nature of the system and the computer it is being run upon. Another consequence of running the simulation as fast as possible is that it may be too fast to visualise and interact with effectively. This can be addressed by lowering the timestep, but this artificially increases how slowly the dynamics runs without necessarily improving the results obtained. A third problem occurs when certain activities (such as multiple users interact with the system simultaneously) cause the MD steps to take more time. This is observed by the users as a sudden slow down of the system, which can be distracting and undesirable.



(a) Core MD loop of NarupaXR and Narupa iMD.



(b) Core MD loop of `narupatools`.

Figure 2.9: Core MD loops of NarupaXR, Narupa iMD and `narupatools`.

Instead of increasing the timestep, a rate-limiting loop is added to the molecular dynamics. This ensures that the dynamics does not run above a given number of steps per second, by inserting small pauses between the steps. This allows a user to specify the speed they wish the simulation to run in a comprehensible unit, such as picosecond of simulation per second of real time. It also provides a buffer zone — if calculating an interaction causes the simulation to slow down somewhat, then the pauses will be made smaller to compensate and hence the user will not notice a slow down. For simulations which would run much faster than the user would desire, inserting pauses slows down the simulation without performing unnecessary additional timesteps.

Rather than specifying some arbitrary system-specific frame interval to state how many steps should pass between frames, a more intuitive specification would be to state how many frames should be produced per second. This can be adjusted based on parameters such as how smooth the simulation should be and how large it is. To achieve this in `narupatools`, extracting snapshots of the simulation and sending them to the users is performed on a second thread. This runs in parallel, at a different rate to the main simulation loop. This decouples the rate at which the simulation runs from the rate at which frames are produced. Figure 2.9 illustrates the difference between the two approaches.

Benchmarks

The main simulation engine that NarupaXR and Narupa iMD can interface with is OpenMM[77,78]. OpenMM is a GPU-enabled simulation engine written in C++, that supports a wide range of forcefields and even the addition of custom forces. The toolset available for OpenMM makes it readily usable for protein-ligand systems.

Whilst NarupaXR provides its own implementation of an integrator written in C$^\sharp$, Narupa iMD can instead leverage the Atomic Simulation Environment (ASE) package[76]. ASE is a Python package that can perform many common molecular dynamics tasks, however it delegates the calculation of forces to an external package. In this case, running OpenMM through ASE involves ASE propagating the system, and OpenMM computing the corresponding forces.

Finally, Narupa iMD also has direct support for OpenMM. Here, both the integration and the forcefield calculations are performed by OpenMM, and Narupa iMD only interrupts to add interaction forces and to extract the state of the system to send to any clients.

Figure 2.10 gives the simulation rate of a Neuraminidase-Oesteltamivir protein-ligand simulation in OpenMM, when ran through the various softwares including NarupaXR and Narupa iMD. The methods compared are:

1. Using NarupaXR, running the `narupa-server` application.
2. Using the OpenMM and ASE OpenMM runners in Narupa iMD. For the ASE runner, modifications were made so the simulation ran using Velocity Verlet using the correct

timestep, as by default the runner uses a Langevin integrator.

3. A benchmark calling OpenMM directly without the use of Narupa.

This protein-ligand system is the model system used for testing Narupa iMD, and has previously been used in studies by Deeks et al.[25]. The 5988 atom system is parameterised with the Amber 14ffSB forcefield in implicit solvent, and ran using a timestep of 0.5 fs.

The simulation was ran using either a Verlet integrator (when ran through OpenMM or NarupaXR) or a velocity Verlet integrator (when ran through ASE). Each simulation was run for a short period of around 4 seconds, and this was repeated 10 times for each setup.

NarupaXR uses a custom integrator written in C$\sharp$. There is a large overhead with running NarupaXR, however the additional calculations when handling interactions from users are insignificant.

In comparison, Narupa iMD implements the interactions in Python. Whilst this allows the definition of new forces easily, it comes with a large amount of overhead. Regarding running the simulation, it may run in two manners — using the Python integrator in ASE and using forces from OpenMM, or running the full simulation in OpenMM and uploading the interactions from Python. Figure 2.10 clearly shows that the performance of molecular dynamics ran through

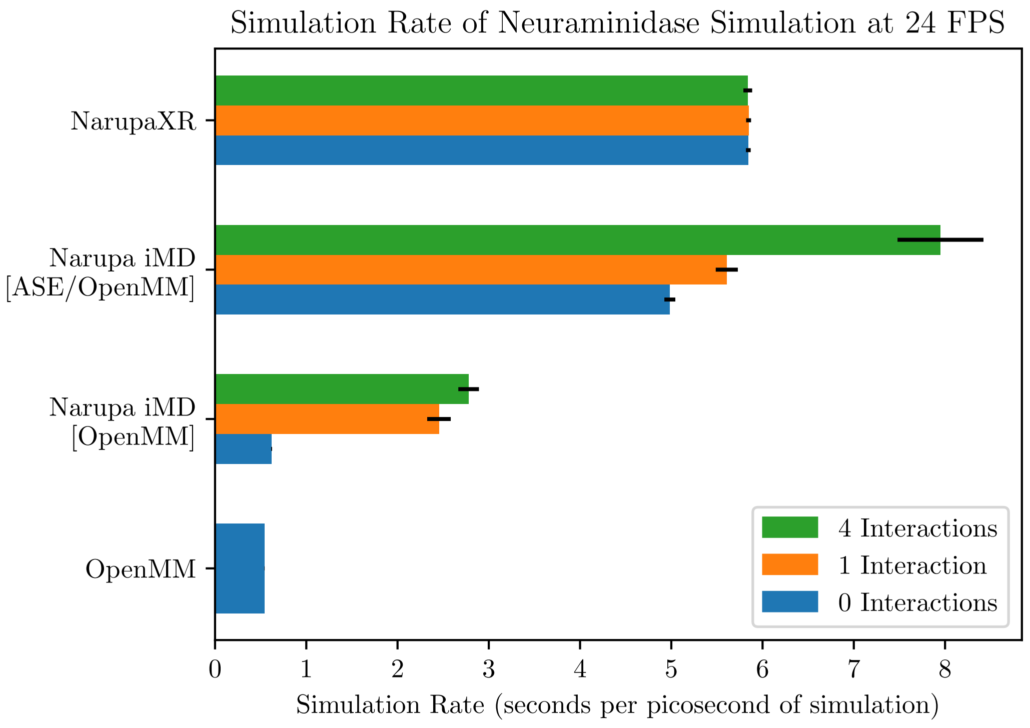


Figure 2.10: Simulation rate of a Neuraminidase system in OpenMM, using different software in or based on Narupa.

ASE is comparable to that of NarupaXR, however it is more modular and modifiable than NarupaXR. When run through OpenMM directly, performance is close to native performance when not interacting with the system. This indicates that at 24 frames per second, streaming the simulation does not incur a significant performance cost.

For the ASE runners, it can be seen that each interaction results in a similar cost. However, for the pure OpenMM systems, the initial introduction of interactions significantly lowers the rate, after which additional interactions have a smaller effect. This is because when interactions need to be applied, the positions must be copied off the GPU. The forces are then calculated in Python, and then these new parameters must be reuploaded to the GPU. As this must occur after every molecular dynamics step, this massively reduces the benefit of running simulations using CUDA and OpenMM.

A possible way of reducing the overhead of interactions is to not recalculate them at every timestep. Instead, the interactive forces can be chosen to only be updated at a fixed interval, for example every 10 dynamics steps. This reduces the accuracy of the interaction, but may be acceptable when the exact force applied by the interaction is not important. This is enabled by default in Narupa iMD. This approach is similar to that of multiple time-step method (MTS) of Streett, Tildesley, and Saville[79]. Here, forces are split into those between nearby neighbours and long-range forces. As long-range forces vary less, these can be recalculated at a lower frequency than short-range forces.

Network Traffic

Assuming that the simulation is not bottlenecked by the speed at which it can run the simulation, then the amount of network traffic that can be exchanged between the server and client may become a limitation. This is referred to as bandwidth, and is usually measured in megabits per second, or Mbps.

For example, YouTube recommends that a 4K video requires 20 Mbps, whilst a 1080p video requires 5 Mbps.[a] Between November 2019 and March 2021, the median download speed in the United Kingdom was 50.4 Mbps, whilst the median upload speed was 9.8 Mbps.

Figure 2.11 shows the bandwidth measured from the server using the same system as for figure 2.10. It can be seen that Narupa iMD has a rate of approximately 15 Mps, whilst NarupaXR had a lower rate of 7 Mbps. The vast majority of network traffic can be accounted for as the transmission of atomic coordinates from the server to the client. For this sytem, there are 5988 atoms, each with a three-dimensional position consisting of a 4 byte float for each component. Therefore, transmitting the positions for each frame requires 0.57 Mbps. At a rate of 24 FPS, this will require 13.8 Mbps.

The factor of half difference between NarupaXR and Narupa iMD can be explained by the different precisions the two use. NarupaXR converts each 4 byte float value to a 2 byte half-precision floating point number, and hence only 6.9 Mbps is required for 24 frames per second. Utilising half-precision floating point numbers is a common approach in computer graphics and neural networks where high precision is not required. Converting Narupa iMD to use half-precision is a possibility in the future, though protocol buffers do not support this natively.

[a]https://support.google.com/youtube/answer/78358

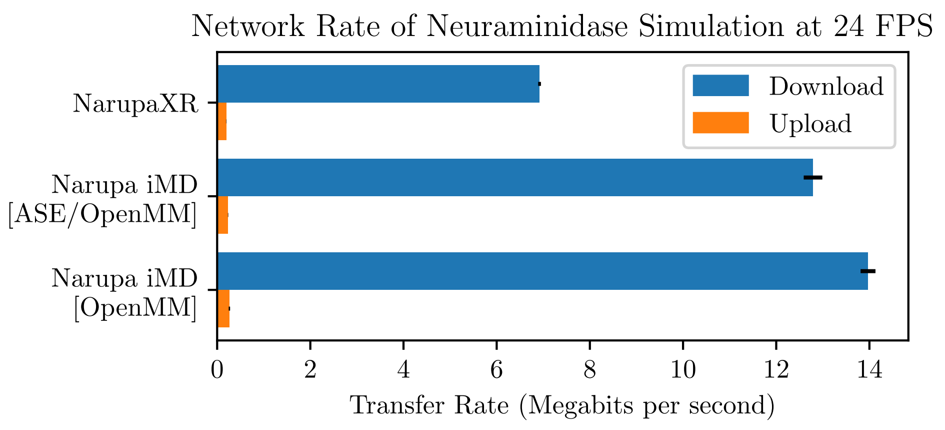


Figure 2.11: Network transfer rate of a Neuraminidase system in OpenMM, using different software in or based on Narupa.

2.6 Integrations and Expansions

The Narupa framework is not intended to replace existing molecular dynamics software. It is instead meant to act as the glue which ties together existing frameworks and allow the exploration of these systems using a new medium.

One of the flaws of NarupaXR is that it reimplements many of the methods and techniques found in other packages, such as Verlet integration and harmonic restraints. In comparison, Narupa iMD can interface with any molecular dynamics package where custom forces may be passed to the simulation engine. Currently, Narupa iMD supports the OpenMM[78], LAMMPS[67] and ASE[76] packages natively. Chapter 5 goes into more detail on how interactive molecular dynamics is made possible with this simulation engines.

Unit Consistency

The wide range of molecular simulation software available has lead to a corresponding breadth of different unit systems in use, as shown in Table 2.2.

Both NarupaXR and Narupa iMD use the unit system used by OpenMM, namely the nanometer, picosecond and atomic mass unit.

As energy is a derived quantity based upon the distance, time and mass, it is important that the four units are **internally consistent**. This is important for molecular dynamics as it involves quantities such as forces, which may be expressed either as a gradient of an energy or a product of a mass and the acceleration. For example, in Narupa a force calculated as a gradient would have units of $\mathrm{kJ\,mol^{-1}\,nm^{-1}}$ whilst a force calculated using Newton's second law would have units of $\mathrm{Da\,nm^2\,ps^{-2}}$. As this system of units is internally consistent, these two units are the same. However, in other packages such as mdanalysis, the use of Å as the unit of distance would result in these two calculations having different units.

	Distance	Time	Mass	Energy	Consistent
OpenMM	nm	ps	Da	$\mathrm{kJ\,mol^{-1}}$	Yes
CHARMM	Å	†	Da	$\mathrm{kcal\,mol^{-1}}$	Yes
GROMACS	nm	ps	Da	$\mathrm{kJ\,mol^{-1}}$	Yes
NAMD	Å	†	Da	$\mathrm{kcal\,mol^{-1}}$	Yes
ASE	Å	†	Da	eV	Yes
MDTraj	nm	ps	Da	-	
MDAnalysis	Å	ps	Da	$\mathrm{kJ\,mol^{-1}}$	No
LAMMPS	Å	fs	$\mathrm{g\,mol^{-1}}$	$\mathrm{kcal\,mol^{-1}}$	No
XYZ File	Å	-	-	-	-
PDB File	Å	-	-	-	-

Table 2.2: Units used by different software, packages, and file formats.

RDKit integration

RDKit[80] is a popular cheminformatics package written in C++, with a Python wrapper. As part of its functions, it can perform geometry minimisation using two forcefields, the Universal Force Field (UFF)[81] and the Merck Molecular Force Field (MMFF94).[82,83] As well as exposing methods for minimising the structure using these two force fields, RDKit also provides ways of getting the energy and gradients associated with the two forcefields. `narupatools` implements an ASE calculator, exposing the UFF and MMFF94 forcefields can be used to run molecular dynamics.

RDKit also supports generating 3D molecular structures from a SMILES string. SMILES, or simplified molecular-input line-entry system, is a common notation for writing a chemical formula, which defines atoms and bonds but not 3D structure.[84] These two features make it very easy to generate simple molecular systems that support a wide variety of elements and bonding patterns. This will make this an easy option for preparing simple teaching material and demos for arbitrary molecules.

Dynamic Visualisation using *nglview*

NGL viewer is a popular Javascript-based molecular visualiser for the browser.[85] It also has a Python interface called *nglview*,[86] which allows molecular visualisation to be embedded directly into a Jupyter notebook. These visualisations can normally show either a static structure or a predetermined trajectory. `narupatools` provides additional *nglview* integration, by allowing dynamics to be viewed directly. The function `show_dynamics` takes a dynamics object and returns an NGL widget showing the current structure. However, it has also attached a callback to the dynamics, such that when a dynamics step is taken, the widget is refreshed with the latest structure. This interactive NGL widget allows users to view the dynamics of their simulation before entering virtual reality, to confirm the system has been set up correctly.

Narupa Builder

One offshoot project I was involved in was the *Narupa Builder*.[7] This is an offline VR application (without the need for a server) which allowed users to create small molecules, in a similar manner to 2D software programs such as GaussView.

My main contribution to Narupa Builder was to transition it to use a shared codebase with Narupa iMD, and a corresponding overhaul in its user interface. I heavily used radial menus, where pressing a button opens the various options (such as adding an atom or removing a bond) as a circle of menu items around the user's controller. They can then select an option by physically moving their hand over the desired option. This is contrasted with other approaches involving a laser pointer emerging from the controller to interact with a more traditional 2D menu floating at a distance from the user.

An achievable and interesting future project would be enabling the builder to be used in a multiplayer context, enabling it to be used to modify a molecule stored on a server. Given the great strides made in Narupa iMD and `narupatools` to make the client-server framework extensible, it would even be possible to seamlessly integrate the building mechanics directly into Narupa iMD. One can imagine a situation where a simulation may be paused midway, a ligand modified using a molecular builder, and the simulation resumed with these new atoms — all within the same piece of software.

Figure 2.12 illustrates several screenshots of the Narupa builder in use. A video showing the operation of the Narupa Builder may be found at the following link: `https://doi.org/10.6084/m9.figshare.22083554.v1`.

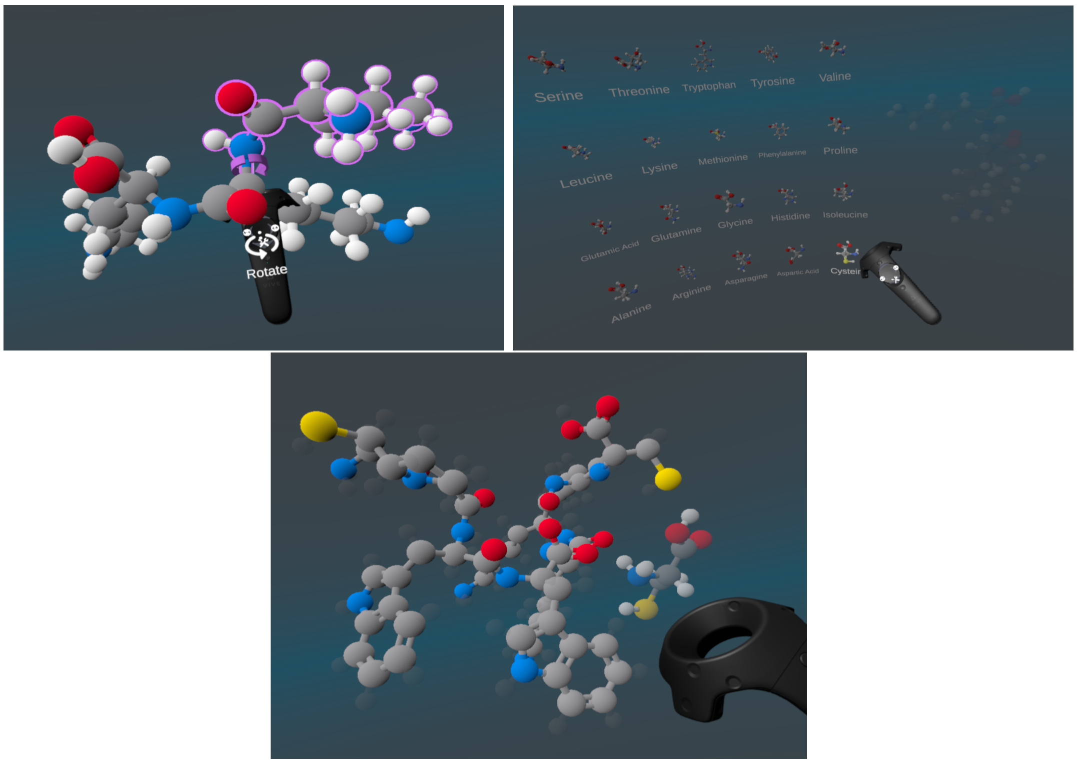


Figure 2.12: Various screenshots of the Narupa Builder in use.

2.7 Visualisation

As discussed in depth in Chapters 3 and 4, there are many different methods for molecular visualisation. With the wealth of systems and properties that can be explored, there is a corresponding abundance of visualisation styles available.

Regarding the general style of representation, these may be broken down into three general categories. Firstly, there are the simple geometric styles most exemplified by ball-and-stick. This is the *de facto* representation of molecules. Variations upon this include the stick or liquorice representation, where the spheres are not prominent, or the simpler sphere representation, where the bonds are not present.

The second category of representations would be the cartoon or ribbon diagrams, commonly used to represent proteins and nucleic acids. These representations are aimed at highlighting the general arrangement and structure of a polymer, without atomistic details.

Finally, there is the category of molecular surfaces. These representations highlight the overall three dimensional shape of a large macromolecule, and highlight the existence of cavities and pores which are so important in drug docking.

General molecular representations such as ball-and-stick as well as molecular surfaces are discussed in Chapter 3. Chapter 4 goes into detail on ribbon diagrams, their history and their implementation.

These representations are influenced by the atomic positions, as well as bonding between the atoms. Some of these representations may also be scaled to represent different data. For example, the spheres representing individual atoms may be scaled to denote a variety of atomic radii, including the Van der Waals radii, the covalent radii or the ionic radii. Together, this describes the physical geometry of the representation.

In order to convey additional information, colour is employed. This allows per-atom data to be conveyed in a way independent of the representation used. For atomistic representations, the element of each atom is often conveyed using specific colours. These colours are generally shared between different software, yielding a common language that different scientists can understand. Molecular surfaces may be coloured to convey important information for interactivity, such as molecular charges, while a ribbon diagram may be coloured to provide information about the residues involved in the structure.

Existing Software

Presenting the wide variety of options available for visualisation is a problem encountered by all molecular visualisation software. For example, VMD allows the colouring and render style to be varied separately, but does not give fine-grained control over the radii of the individual atoms. As visualisations can often feature many options, all possibilities may not be covered by the graphical user interface. Often, advanced visualisation options may be controlled by an additional scripting language or command line built into the software. VMD, ChimeraX and PyMol all feature this approach, using scripting languages such as Python or TCL.

Some attempts have been made at defining specific visualisation languages to allow greater customisation. Palmer[87] and Hultquist and Raible[88] defined the Pdbq and SuperGlue languages respectively. These were interpreted languages which allowed rapid development of new visualisation techniques. ViSlang[89] is a more recent approach at developing a language for describing scientific visualisations.

These are examples of domain-specific languages (DSL). In contrast to a general language such as Python or C++, a DSL is a language specialised for a specific task. For example, markup languages such as HTML can be considered to be a DSL, as its a language for writing web pages specifically. Using a DSL to control the visualisation allows a maximal amount of flexibility, but generally has not been widely adopted in molecular visualisation software.

Another approach is to define a format expressed in JSON which describes the visualisation setup, such as DXR. DXR is a toolkit for Unity for immersive data visualisations for AR and VR.[90] This was developed to encapsulate the low-level programming required to perform data visualisation, such as parsing data, creating objects and updating visualisations based on new data.[90] DXR encodes a visualisation specification, inspired by similar approaches to language-agnostic visualisations such as Vega-Lite.[91]

The approach used by DXR is similar to that adopted by the visualisation system I have developed in Narupa iMD. This approach used in Narupa iMD is designed to be modular, allowing the reuse of specific components between different visualisations. It also acts as a testbed for rapid development of new representations, and hence was very useful in developing the novel visualisation approaches described in Chapters 3 and 4.

In NarupaXR, each user could explore different visualisations independently. Whilst this can be a useful feature, it can lead to confusion where different users are discussing the same system but with different visualisations. In the Narupa framework, visualisation is instead set up at the server level and synchronised globally. As servers are often run using Jupyter interactive notebooks, which can execute arbitrary Python code, there is no need for there to be a scripting language built into the visualisation frontend. Through use of the `narupa-server` and `narupatools` Python libraries, complex visualisations can be set up through Python and synchronised to the clients using the shared state.

Visualisation in Narupa iMD

The visualisation system I have implemented in Narupa iMD is conceptually a node-based system. Each individual logic block is a **node**, which take in arrays of data and either generates new data or renders specific objects to the screen. For example:

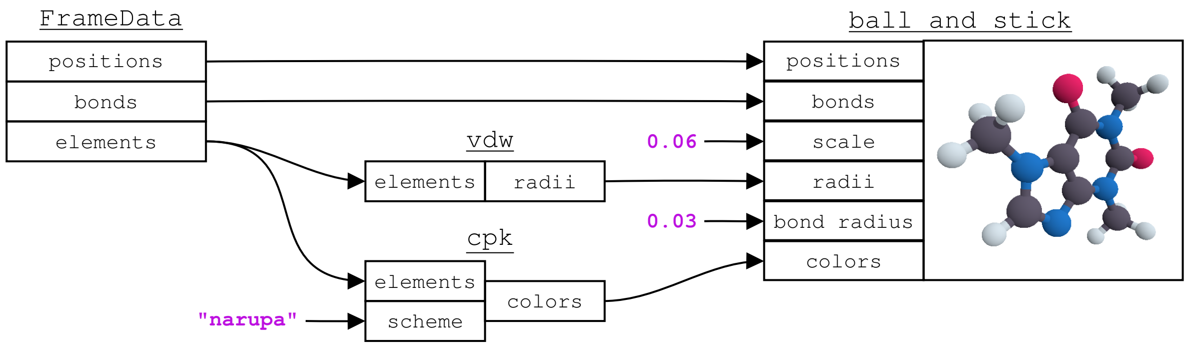
1. A van der Waals node would take in the atomic elements (an array of integers with the name `particle.elements`) and outputs an equally sized array of floats with name `particle.scales`, where the nth value is the van der Waals radii of the nth atom.

2. A ball and stick node takes the positions of each atom as well as atom-specific data, such as radii and colours. It uploads this data to the GPU and renders the various shapes making up the representation, namely spheres and cylinders.

The input to the visualisation system consists of a specification of parameters for each node, as well as the `FrameData` representing the system to be rendered. These nodes can therefore be seen as transformations applied to a source frame, reinterpreting raw data in various ways

```
{
    "color": {
        "type": "cpk",
        "scheme": "narupa"
    },
    "scale": "vdw",
    "render": {
        "type": "ball and stick",
        "particle.scale": 0.06,
        "bond.scale": 0.03
    }
}
```

(a) Example of a visualisation input for Narupa iMD, in JSON format.



(b) Example of a visualisation pipeline generated from Figure 2.13a.

Figure 2.13: Standard ball-and-stick visualisation as defined in Narupa iMD, given both in its JSON form which is transmitted from the server to the client, and the subsequent nodes generated on the frontend which results in a visualisation of a caffeine molecule.

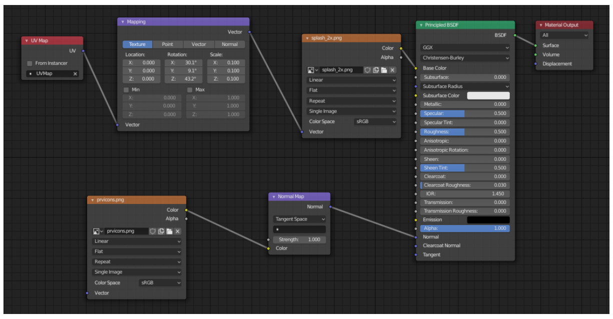
and using it to drive a set of primitive visualisation nodes. These primitive visualisation nodes include visualising spheres, cylinders and hyperboloids.

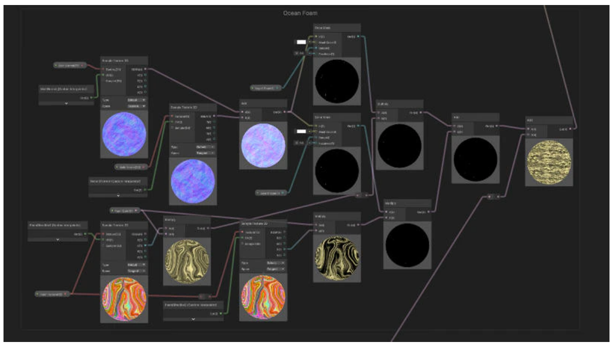
An example of a visualisation pipeline consisting of four nodes is given in Figure 2.13. Two of the nodes are responsible for the scaling and colour of each atom. These nodes get the corresponding data from the source frame, which in this case is the element of each atom. The final node takes information from the others and renders specific geometric primitives to represent the system.

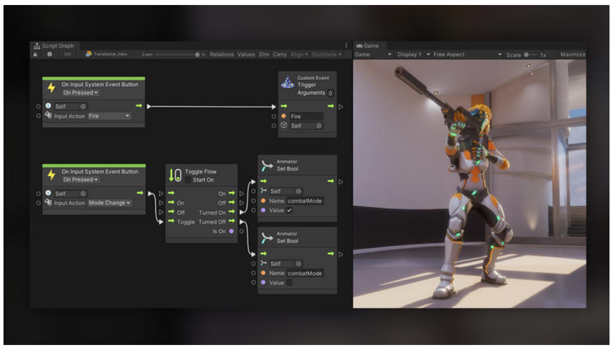
The use of a node based system allows several optimisations to be made. As the flow of information is well defined, when certain data is changed only the corresponding nodes are refreshed. For example, in a general simulation the elements and bonds are invariant. If a visualisation such as Figure 2.13 is used, when the positions are altered the `vdw` and `cpk` nodes do not require recalculation. This prevents nodes with a high computational cost from being executed when their input has not changed. Similarly, this pipeline can avoid unnecessary uploading of data to the GPU. In this case, only the positions array has to be reuploaded, whilst the radii, bonds and colours remain unchanged

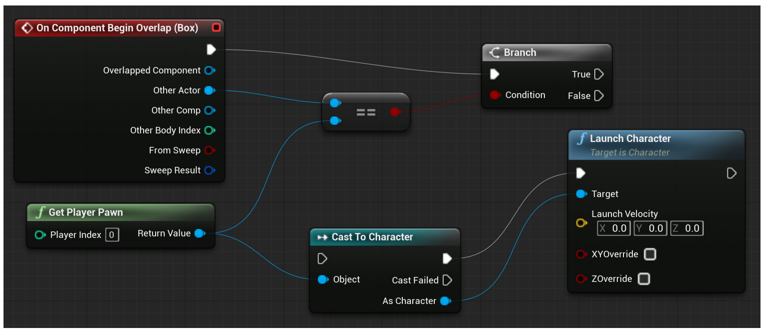
The nodes which actually render specific objects to the screen are adapted to only use data if it is available. For example, if van der Waals radii are specified, an array of scales will be uploaded to the GPU and each sphere will then be scaled individually. If no radii are specified, a fixed value will be uploaded instead. These optimisations are made in the shader programs which dictate how the nodes render objects to the screen, and avoid uploading redundant data to the GPU.

This node-based approach to visualisation is seeing increasing use within software geared towards rendering (Figure 2.14 shows some examples in relevant software). Software such as Unreal, Unity and Blender all provide node-based approaches allowing non-technical users to create rendering code. Unreal also allows node-based programming for general game logic. The advantages of using nodes is that it allows the encapsulating and reusing of small logical pieces, each representing a transformation or generation of one or more pieces of data. This is well suited to molecular visualisation, where there are separate parts that may be varied including the scale, colour and shape of the final render.


(a) Blender nodes.


(b) Unity Shader Graph.


(c) Unity Visual Scripting.



(d) Unreal Engine Blueprints.

Figure 2.14: Examples of node-based visual programming in modern software for 3D modelling and game development.

2.8 Conclusions

This chapter has been a summary of the work over the past three years on the next generation of Narupa. Though a great proof of concept, NarupaXR carried technical debt due to its development history, as well as existing in a programming language not familiar to many scientists.

Narupa iMD has from its origins been written to be extensible, easy to understand and widely available. In addition to its use for interactive molecular dynamics, which will be explored in Chapter 5, work is also being done in applying it to various other use cases.

The Narupa builder was not optimised for large systems, and scales poorly with the number of atoms. This is mostly due to the lack of any spatial hashing or other structures that would allow for efficient calculation of collisions between atoms, which are computed when large fragments are moved together. Other groups are exploring interfacing protein design codes such as ISAMBARD[92] and Rosetta[93] with visual frontends using the Narupa framework, and would greatly benefit from development work on the performance of the builder.

As part of my PhD, I also was involved in developing an offshoot of Narupa iMD in collaboration with Hyundai. This work interfaced with Monte Carlo simulations for examining nanoparticles. The ease with which this project was completed is a testament to the extreme flexibility that the Narupa framework provides, by allowing certain parts of the pipeline to be readily interchanged.

As shown by figure 2.11, small choices such as the choice of encoding a position in either 2 or 4 bytes can have a dramatic effect on the amount of data that needs to be transmitted. The protocol buffer serialisation format does not natively support half-precision floating point numbers, however, the gRPC communication layer is encoding agnostic. This means that in the future we could move to using a more efficient encoding strategy for sending data between servers and clients. Possibilities include adopting formats that more closely resemble binary formats used to store molecular dynamics strategies, such as HDF[94]. As is often the case in network communication, a balance must be found between minimising the amount of data transmitted and minimising the performance cost of packing and unpacking these messages.

Whilst Narupa iMD more readily supports different kinds of interactions (and `narupatools` expands this with stateful interactions), they incur a significant performance penalty. One possibility is to shift the commonly used interactions to be implemented in C++. A further step could be to create an OpenMM plugin (also in C++) that would interact with the simulation on a lower level. Currently, interactions are applied using the `CustomExternalForce` class of OpenMM, which whilst very generic, suffers in performance relative to bespoke force classes.

In conclusion, the work myself and others have contributed in the development of the Narupa framework has lead to an extensible and modifiable suite of software that can be used for performing interactive molecular dynamics, viewing immersive trajectories in co-located virtual reality and building molecules intuitively.

Chapter 3

Molecular Visualisation

A fundamental part of Narupa is the display of molecular systems, specifically in virtual reality. Molecular visualisation is the field describing how we portray our numerical simulations in an intuitive and visually distinct manner. The challenges of molecular visualisation are only amplified when considered in virtual reality, where a drop in frame rate can lead to motion sickness. Taking full advantage of the parallel and programmable nature of the GPU is therefore essential when visualising dynamic molecular systems. The large range of scales that a user may resize their system to be, often over a short period of time, provides additional challenges when considering the resolution of the graphics that must be produced.

In this chapter, I will start with an overview of the history of molecular visualisation, rooted in its origins in the physical models of the 19th and 20th centuries. These early pioneers have shaped the graphical representations we now use to view molecules.

I will discuss how the computer graphics pipeline operates and the application of the ray casting technique to molecular visualisation. I will start with the common shapes where their intersection with a ray can be solved analytically, including spheres, cylinders and hyperboloids. I will then describe other techniques used for more complex shapes, such as signed distance fields.

Much work has been done in the field of visualisation for biomolecules, however the field of materials science has seen relatively fewer advances. Periodic systems pose unique challenges for visualisations, as atoms may pass from one side of the simulation box to another by crossing these boundaries. This gives the appearance of sudden and unexpected movement, which is particularly an issue in virtual reality. I will discuss my implementation of a visualisation technique for periodic boundaries which, while often used to portray periodic systems in chemistry textbooks, is yet to be employed in digital visualisation software.

3.1 Introduction

Molecular Representations

The field of molecular visualisation is the digital analogue of the physical models we build to represent the atomic world. From the early works of Dalton in the 19th century, atoms have often been represented as spheres or circles. These spheres would come to be joined by lines, giving us the basis of structural formulae used today. While the circles have since been dropped when writing structural formulae on paper, these diagrams became the direct inspiration for physical models.[95] These early models, aptly described as ball-and-stick, are the *de facto* representation of molecules in 3D, being ubiquitous in chemistry textbooks, research papers and molecular modelling kits. Whilst originally rigid constructions (such as Figure 3.1), later approaches to these physical models allowed flexibility for rotation of these models about certain bonds.[96]

Later, another representation appeared — that of the space-filling model.[97,98] Here, the spheres are scaled up to represent the van der Waals radii of each atom, hence obscuring the bonds. The larger radii of each atom can better indicate the volume of the molecular structure.

Finally, another common representation of molecules is the so-called liquorice representation, which emphasises bonds instead of atoms. This can be seen as a three-dimensional analogue of the 2D stick diagrams and models. Irving Geis's painting of myoglobin is a great

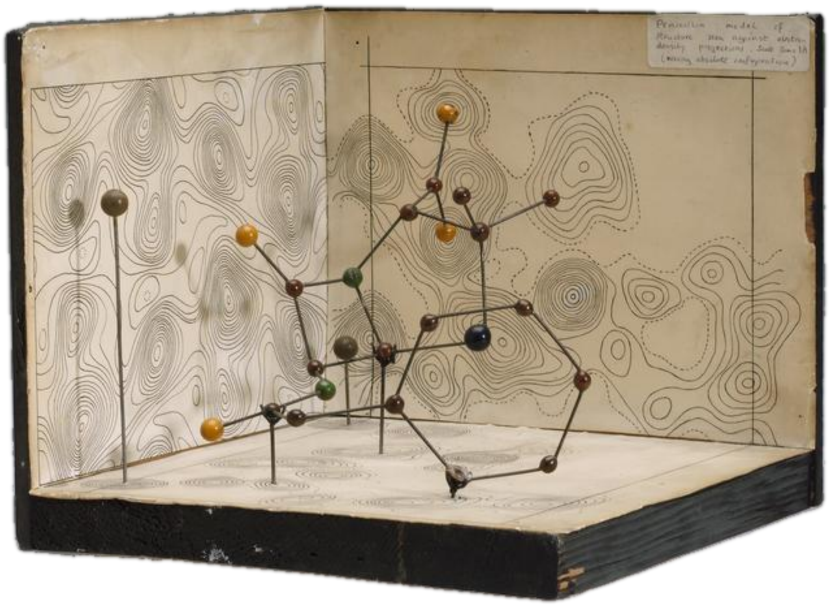


Figure 3.1: Molecular model of penicillin by Dorothy Hodgkin, built in 1945. Used under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 Licence.

example of an early liquorice diagram before the advent of widespread computer visualisation.[99]

With these early models, the association of elements with certain colours originated with Hoffmann using a set of croquet balls during a lecture at the Royal Institution.[95] This convention is commonly referred to as the CPK scheme, after Corey, Pauling and Koltun, who stated the colours in their papers on physical space-filling models.[97,98] These colours include oxygen as red, nitrogen as blue and carbon as grey or black. Consistent colours across many applications and visualisations provide a common visual language that allows users to identify common elements.

Crucially, representations must not unduly bias our opinions. For example, von Baeyer's argument that cyclohexane must adopt a strained planar structure (rather than the conformations that it is now known to adopt such as chair and boat) may have been influenced by his interaction with molecular models with flexible bonds that allowed such a structure.[100]

As graphical capabilities of computers have evolved in the latter half of the 20th century, these physical models have transitioned into the digital domain. Figure 3.2 illustrates the three most common atomistic representations, as rendered in Narupa iMD. Early software such as ORTEP[101–103], RasMol[104,105], Raster3D[106] and Molscript[107] gave researchers the ability to create illustrations of their systems. These static renders have evolved further, with modern techniques allowing the visualisation of millions of atoms at once. This evolution has also allowed the visualisation of dynamic systems and trajectories. The current forerunners in molecular visualisation include desktop applications such as VMD[64], PyMOL and Chimera[108,109], and web-based programs such as NGL Viewer[85,86], Jmol[110] and the Mol* Viewer[111].

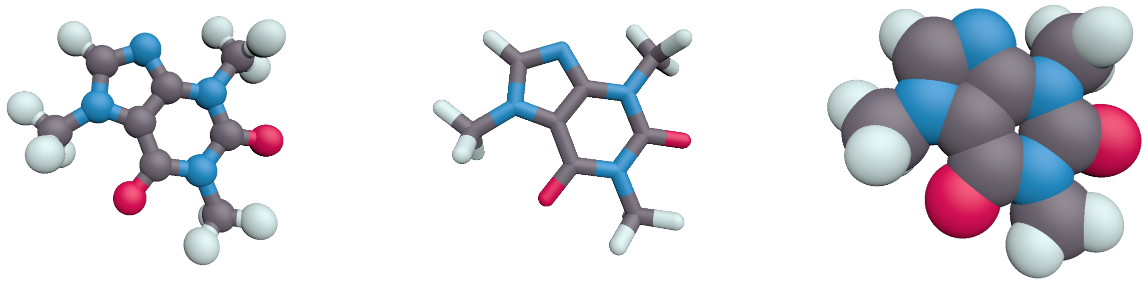

(a) Ball and Stick (b) Liquorice (c) Space Filling

Figure 3.2: Illustration of a caffeine molecule represented using the three common atomistic visualisations. The atoms are coloured based on the CPK color scheme.

Complex Visualisations

The family of simple atomistic representations (ball-and-stick, liquorice and space-filling), while well suited to smaller molecules, can become confused and busy when visualising large macromolecules such as proteins. This has led to representations that are specifically designed to highlight various larger-scale features. Two methods commonly employed for protein visualisation are ribbon diagrams, which highlight the protein backbone and secondary structure, and molecular surfaces, which highlight the cavities within the protein structure.

Ribbon diagrams highlight the general structure and alignment of the polymer chains that make up a protein. This structure can be most simply seen by stripping out all atoms except the alpha carbons at the centre of each element of the polymer. These may then be connected by straight bonds using the liquorice representation, depicting the protein structure as an angular series of lines. From this representation, various techniques have been developed to render a smooth curve that illustrates the protein's conformation.[112] Chapter 4 goes into greater detail of the history, development and application of such ribbon diagrams.

Molecular surface representations highlight the structures at the surface of the protein, as well as cavities that may host a substrate. Here, the steric arrangement and molecular makeup of the surface are of vital importance, revealing areas in which the substrate might bind both chemically and sterically. A good approximation of the molecular surface is to use the aforementioned space-filling model, with the spheres of each atom overlapping to give an indication of the overall surface.

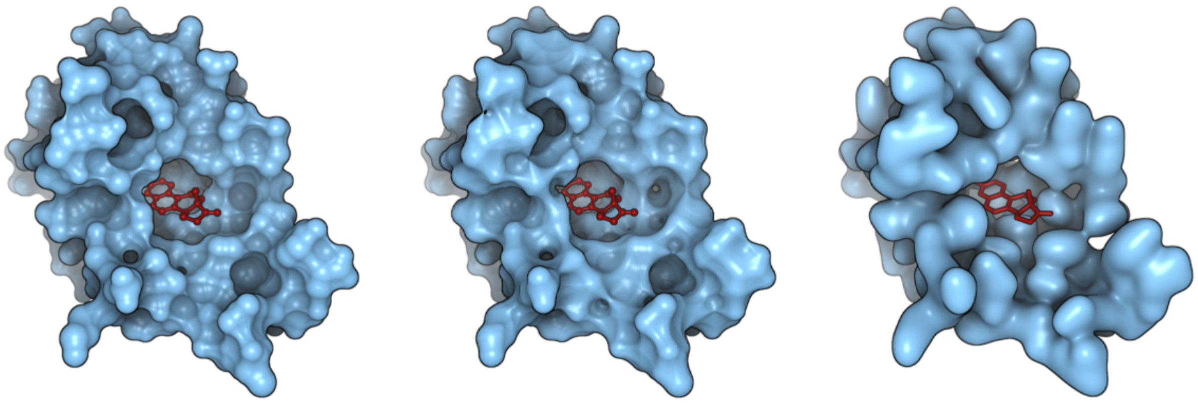

(a) Solvent-excluded Surface. (b) Molecular Skin Surface. (c) Gaussian surface.

Figure 3.3: Comparison of three molecular surfaces used in visualisation: the solvent excluded surface (SES), obtained by rolling a probe sphere over the van der Waals surface; the molecular skin surface (MSS), defined from a weighted set of points and the Gaussian surface, defined as an isosurface of a set of atom-centred Gaussians. Adapted from B. Kozlíková et al. "Visualization of Biomolecular Structures: State of the Art Revisited". In: *Computer Graphics Forum* 36.8 (Dec. 1, 2017). Publisher: Blackwell Publishing Ltd, pp. 178–204. ISSN: 01677055. DOI: `10.1111/cgf.13072` under the Creative Commons Attribution 4.0 license.

Surface representations are often used to illustrate the permeability of the protein's structure to a solvent. This is achieved by modelling the solvent as a probe sphere, which is rolled across the van der Waals surface. The centre of this probe sphere defines a new surface, called the solvent accessible surface (SAS).[114] This can be seen to be the same as increasing the radius of the space-filling representation by the probe radius.

A related surface to the SAS is the solvent excluded surface (SES) seen in Figure 3.3a.[115,116] Here, the surface is defined as the region carved out by the probe's surface as it rolls across the van der Waals surface. This has the effect of filling in gaps in the surface where a solvent molecule would not be able to access.

Analytic approaches to computing the SES involve decomposing the surface into the various geometric patches that make it up: spheres, where the probe is in contact with a single atom; torus 'spindles', where the probe is simultaneously touching two atoms; and spherical patches, where the probe is in contact three atoms at once. The SES is geometrically well-defined but due to its shape it can result in sharp points or singularities that can make them challenging to render. Algorithms for generating analytic SES surfaces include contour buildup[117–119] and reduced surfaces[120–122].

The evaluation of molecular surfaces can proceed either algebraically or via numerical discretization. In the latter approach, a grid of points is evaluated to determine if they lie within or outwith the surface. Given a grid of values, the marching cubes algorithm is commonly used to convert this into a triangular mesh.[123] This approach is commonly used to render volumetric data such as isosurfaces. However, marching cubes scales poorly with the resolution of the mesh, and hence a trade-off between appearance and speed must be made.

To avoid the issues of singularities that appear in the SES, alternative surfaces may be used. One such surface is the molecular skin surface (MSS) (Figure 3.3b), which is defined using a weighted combination of spheres.[124] Algebraically, the surface may be decomposed into quadric surfaces and hence is well suited for GPU ray casting that is discussed further in this chapter.[118,125]

Another surface-like approach is to use a Gaussian isosurface (Figure 3.3c), consisting of a set of Gaussians positioned at each atom. This approach is similar to the visualisation known as metaballs. Recent work has shown how screen-space calculations can render the Gaussian surface of large systems in real-time.[126]

The MSS and the Gaussian surface are purely mathematical surfaces and do not have any biological meaning. However, the parameters used to define them may be tweaked such that they provide a good approximation of a more rigorously defined surface such as the SES.

Lastly, the ligand-excluded surface (LES) is a more advanced form of the SES, where in place of a spherical probe representing the solvent, the full dynamic structure of the ligand is used to carve out the surface.[127] However, the complexities involved in calculating this prohibit its use in dynamic visualisation.

Even with earlier visualisation software, it was possible to create images incorporating many thousands of atoms with computationally-demanding effects such as translucency and shadows. However, creating a still image can take as much time as one is willing to allow, whilst an dynamic display of molecules must render each image at interactive frame rates. We can draw parallels between this and the link between computer graphics in movies and video games. A movie has many months to render all of its frames, which it can distribute over many computers. In comparison, video games must produce graphics at interactive rates on a personal computer. Therefore, there is a lag between when cutting-edge visual effects can be applied to static renders and these techniques being incorporated into real-time trajectory visualisation.

The Graphics Pipeline

Rendering is the process in which a computer generates an image to display to the user. On modern computers, the later stages of rendering take place on specialised hardware known as Graphical Processing Units (GPUs). Unlike the core hardware on which most computer software runs (the Central Processing Unit, or CPU), GPUs are specifically designed for the highly parallel task of generating images to display to the user. The sequence of steps which the GPU takes to render an image, based on instructions received from the CPU, is known as the **graphics pipeline**.[128] Whilst some of these are low-level processes that are inaccessible to the user, many stages of the pipeline can be modified by writing small programs known as **shaders**. Figure 3.4 shows the main steps involved, highlighting the step which can be modified by shaders.

The first step of the pipeline involves defining what geometry should be rendered. When defining the 3D objects in our scene, the standard method is to use a data structure known as a **mesh**. This is a representation of a 3D object and consists of vertices and triangles. The use of triangles is fundamental in rendering, as they are planar and lend themselves to **interpolation**. Interpolation is the process in which data that is assigned to each vertex (such

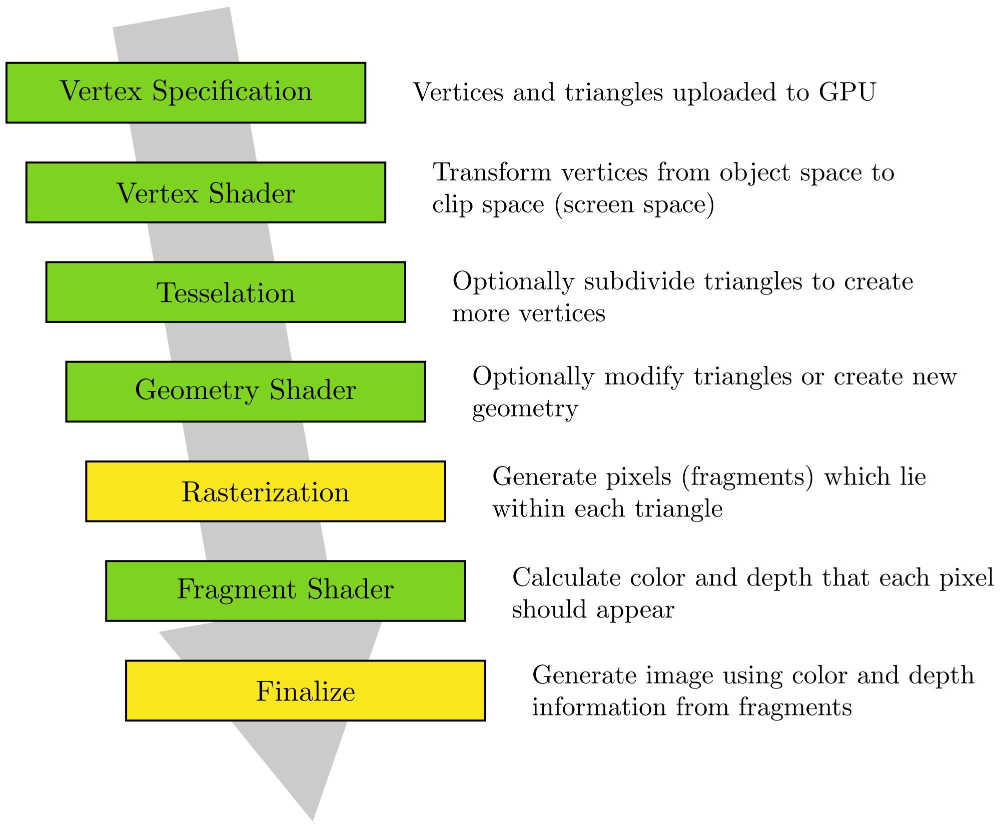

Vertex Specification
Vertices and triangles uploaded to GPU
Vertex Shader
Transform vertices from object space to clip space (screen space)
Tesselation
Optionally subdivide triangles to create more vertices
Geometry Shader
Optionally modify triangles or create new geometry
Rasterization
Generate pixels (fragments) which lie within each triangle
Fragment Shader
Calculate color and depth that each pixel should appear
Finalize
Generate image using color and depth information from fragments

Figure 3.4: Overview of the steps in the graphics pipeline. Steps that can be modified using a shader are designated in green, whilst other steps are designated in yellow.

as a colour or a surface normal direction) is used to calculate data at any point within the triangle, by weighting each corner depending on how close the point is. The interpolation of surface normals allows a smooth shape such as a sphere to be represented by discrete polyhedra made of triangular faces.

It is important to note that the concept of a vertex in rendering is related but not identical to the standard mathematical one. For example, a mesh of a cube requires 24 vertices, though the mathematical object of the cube has only 8. This is because at each vertex three faces meet, each with a separate surface normal. For each square face, the square requires two triangles, giving a total count of 12 triangles and 24 vertices as shown in Figure 3.5. This duplication of vertices is required because we are using triangles as our primitives.

The first programmable part of the graphics pipeline is the **vertex shader**. Shaders are small programs that are compiled and run on the GPU, with the vertex shader being responsible for manipulating the vertices that have been uploaded as part of the mesh. If our scene contained many cubes, it would be inefficient to define the position of every vertex. Instead, a mesh is usually defined in some internal coordinate system known as **object space**. For example, we may define a model cube with sides of length 1 and centred about the origin. To render other cubes, we transform our model cube by rotating, translating and scaling each vertex. This transformation is performed within the vertex shader and takes each vertex from object space (relative to its internal coordinate system) into **world space** (relative to a global coordinate system), as shown in the first step of Figure 3.6a. More details on the transformations used within the graphics pipeline may be found in Appendix A.

We also need to define from what angle and direction we are viewing our virtual world. This is defined by a **camera**, which is also positioned and rotated relative to the global world coordinate system. This is our view into this scene and is used to project the 3D scene as defined within the computer onto the 2D grid of coloured pixels that is presented to the user. Therefore, further transformations are performed on the vertices of our mesh. Firstly, the

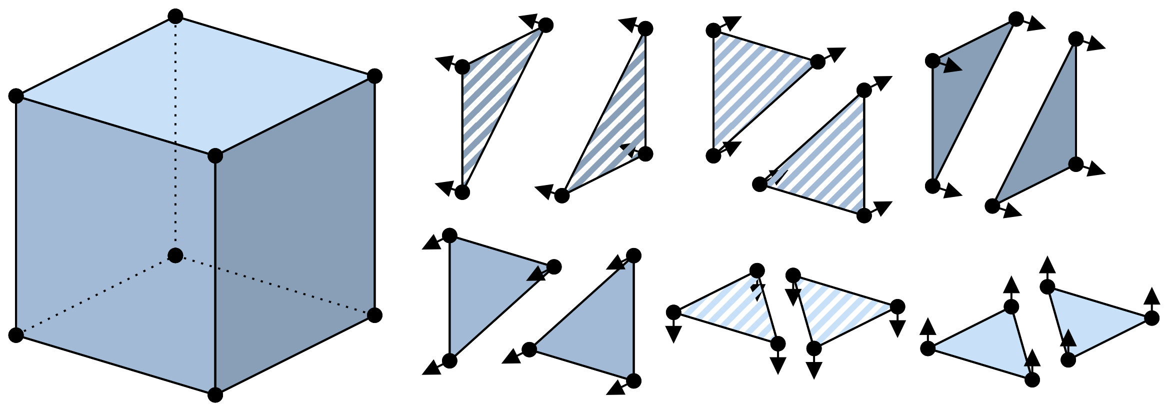


Figure 3.5: Mesh of a cube broken down into 12 triangles and 24 vertices. Each of the six faces consists of two triangles.

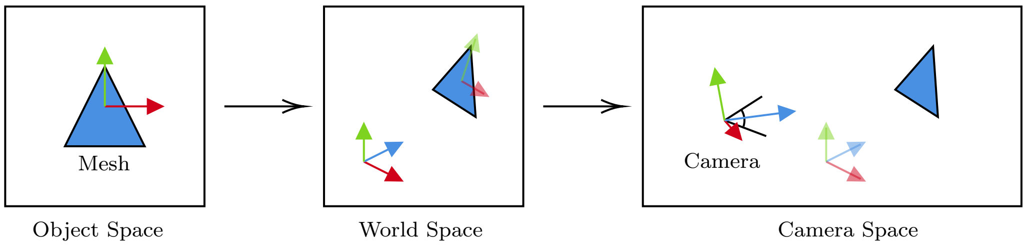
vertices in world space are transformed to be defined relative to the camera (the second part of Figure 3.6a). Like the human eye, the camera has a field of view, which defines the amount of the scene in front of the camera that is visible. For rendering purposes, there also has to be two **clip planes** — cutoffs defining the minimum and maximum distances objects are viewable from. Together, the field of view and clip planes define a rectangular frustum (see Figure 3.6b). Up to this point, the transformations generally do not distort the meshes, merely translate, rotate and uniformly scale them. The next transformation however is a perspective transformation, which transforms the camera's frustum into a unit cube. This results in objects further away from the camera being scaled smaller than those closer to the camera, hence causing objects further away to appear smaller as is expected. More information on how this transformation is carried out may be found in Appendix A. This final space is known as **clip space**. In clip space, the objects are now correctly positioned for rendering, with them being projected onto the flat face towards the camera to yield an image. The depth of each vertex within clip space is used to determine which objects are closer to the camera, and hence which objects appear in front of each other.

The above steps are all computed by the vertex shader, which takes in data from the CPU and outputs a vertex defined in clip space, along with any other data the shader requires. The pipeline then takes each triangle (which has now been projected onto a 2D screen) and works out which pixels would be inside it. This step is known as **rasterisation**. Finally, for each pixel within each triangle, another program known as the **fragment shader** is run (see Figure 3.6c). This takes in the interpolated data from the three vertices that form the triangle, such as the surface normal and colour, as well as global data such as the viewing angle and the direction of the lighting, and calculates what colour to set the pixel to. This calculation can use simple approximations of how surfaces are shaded, or complex calculations to give effects such as specular highlights or rim lighting.

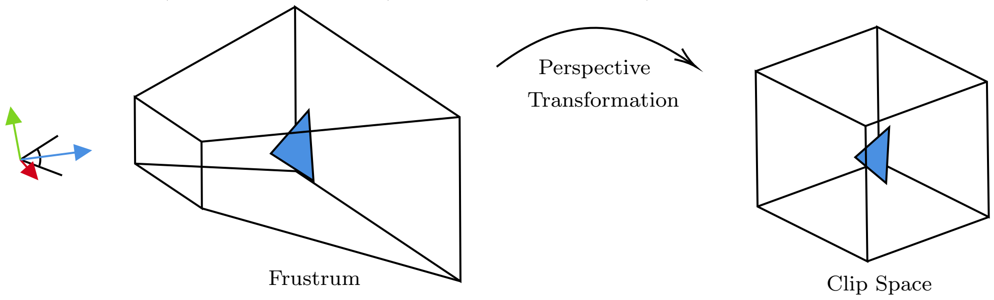
In recent years a new form of rendering has been finding increasing use within video games and visualisation — **ray tracing**. Unlike traditional rendering, where each object is drawn onto the screen one by one, ray tracing adopts an approach that acts like human vision in reverse. Everything we can see with our eyes has originated at a light source, which has then reflected and refracted through the objects around us before reaching our eyes. Ray tracing seeks to emulate this, by firing a ray for each pixel from the camera into the three-dimensional scene we have defined. The intersections of these rays with the objects are calculated, reflected and propagated through the scene. This allows advanced techniques such as realistic lighting, reflections and shadows to be incorporated into a scene. There have also been special GPUs designed specifically with hardware aimed at optimising this kind of approach.[129–131]

Although ray tracing has been applied to visualise molecular systems,[132,133] it is still limited by performance. Ray tracing can often be sped up by using voxel-based approaches.[134,135] This approach is best used for static structures, with the space occupied by the molecule subdivided into a 3D grid, with each cell being called a voxel. Therefore, to compute a ray intersection with the molecule, only the spheres that lie within voxels passed through by the ray need to be considered. The use of this approach can allow the visualization of billions of atoms but is ill-suited for dynamic visualization.[134]

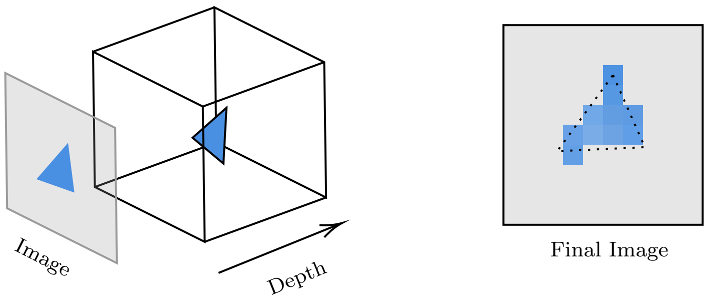
The rate-limiting step in traditional triangle-based rendering may lie either with the uploading and processing of vertices or with the individual processing of each fragment. Determining where this bottleneck lies can be done by varying the resolution of the final image. Here, the number of fragments will be reduced, and hence if performance is still an issue then the first part of the shader is the limiting factor. This is usually optimized by reducing the number



(a) Transformation of a mesh from object space (relative to its own internal coordinate system) to camera space (relative to the camera).



(b) Transformation of the mesh from camera space to clip space. This accounts for perspective, where objects further away are shrunk relative to objects closer to the camera.



(c) Interpretation of clip space as a two-dimensional image, with the z coordinate representing depth. The final image is obtained by sampling the xy face of clip space on a discrete grid of pixels.

Figure 3.6: Various steps involved in transforming a triangle to a set of pixels on the screen.

of draw calls, which are instructions from the CPU to the GPU specifying what to draw and where. Therefore, the aim is to reduce the number of vertices used, and where possible group them into batches that can be rendered in one call.

When rendering many copies of the same mesh, a common technique is **geometry instancing**. This can be used to render the same mesh many times, with the exact position and rotation of the mesh determined on the GPU within the vertex shader. This is very optimal for objects where a full 4×4 transformation matrix would be overkill. For example, a sphere is uniquely defined by a position and a single scaling factor. Defining a full 4×4 transformation matrix for each sphere is therefore unnecessary. Instancing reduces the number of draw calls required by the GPU, however, the vertex shader must still run for every vertex of the mesh for every instance that needs to be rendered. Hence minimising the number of vertices is also of vital importance.

An approach to deal with this is texture-based impostors, which are often used within video games to handle a large number of distant 3D objects. Instead of rendering these objects fully, when viewed at a distance they are replaced by a 2D square containing a picture that is at approximately the correct angle. A range of pictures of the model are precomputed from various directions, and the appropriate one is chosen to display on the screen.[136]

Using the programmable nature of the modern graphics pipeline, a middle-ground can be applied that uses similar concepts to ray-tracing within the traditional graphics pipeline. This approach, known as **ray-casting impostors**[137] or true impostors,[138] has found widespread use in molecular visualisation.[35,36,85,118,122,125,139–145] It is well suited for simple algebraic surfaces, for which there is an analytic solution for their intersection with a ray. This category includes spheres, cylinders, cones and hyperboloids, all of which are commonly used in molecular visualisation. Ray casting also allows certain effects to be easily applied, such as outlines.[35] Even for polyhedra, standard instancing can be slower than ray casting when the number of particles enters the millions.[146]

3.2 Ray casting

In principle, ray casting combines the two common pathways for rendering a scene — triangle-based rasterisation and ray tracing. The first stage of the pipeline is unchanged, with a mesh made of triangles transformed from object space to clip space. However, at this point we can leverage the fact that the fragment shader can return other data apart from the color of the pixel. It can also return a custom depth, which is normally calculated from the position of the triangle. The position of the triangle on the screen is unchanged (and cannot be within the fragment shader), but by changing its depth we can change how it intersects with other objects. We are effectively pulling or pushing each pixel into or out of the screen.

Another feature of a fragment shader is that it may **discard** a pixel. This effectively skips rendering this point, and is often employed for cutting holes in textures using alpha clipping. In ray casting, it can be used to discard points that would lie within the low-vertex mesh, but not within the object we wish to render. Combining depth modification and discarding pixels, arbitrary shapes can be created as long as they lie within the bounds of the original triangle. Figure 3.7 shows how these two techniques work, in reference to projecting an image in clip space as shown in Figure 3.6c.

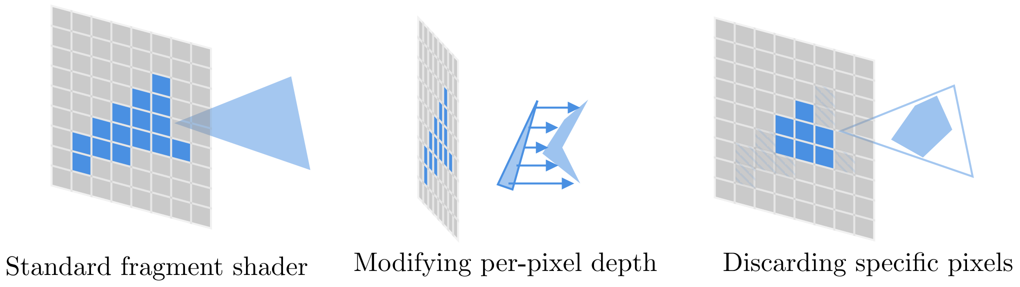


Figure 3.7: Techniques used in the fragment shader to perform ray casting.

Ray casting therefore trades off the number of vertices by making the calculations performed per pixel more complex. However, for simple shapes, there is a performance gain when considering ray casting yields perfectly smooth surfaces, which would require an excessive number of vertices to reproduce with standard shading.[147] As the amount of intersections calculated is proportional to how many pixels the shape occupies on the screen, a shape such as a sphere can be viewed at short distances without artifacts. This contrasts with the traditional representation of a sphere as a many-faced polyhedron, as when viewed at short distances the sharp edges may become apparent. This is especially important in virtual reality, where the ability of the user to manipulate the size and position of the system at speed means that the system can vary very quickly from occupying a small fraction of the screen to occupying a majority of it.

The containing shape (referred to as the ray-tracing area by Toledo and Lévy[137]) should enclose every point which would lie in the surface. The two approaches are to use a simpler mesh such as a cube to enclose the surface, or to utilise a rectangular quad that is rotated to always face the camera. The goal is to achieve a balance that results in the best performance. If the mesh is not tight-fitting enough, there will be many pixels where the ray casting calculations will occur and not result in any intersections. However, using a more complex tight-fitting mesh results in more vertices that will slow down the pipeline. Figure 3.8 demonstrates a couple of common approaches.

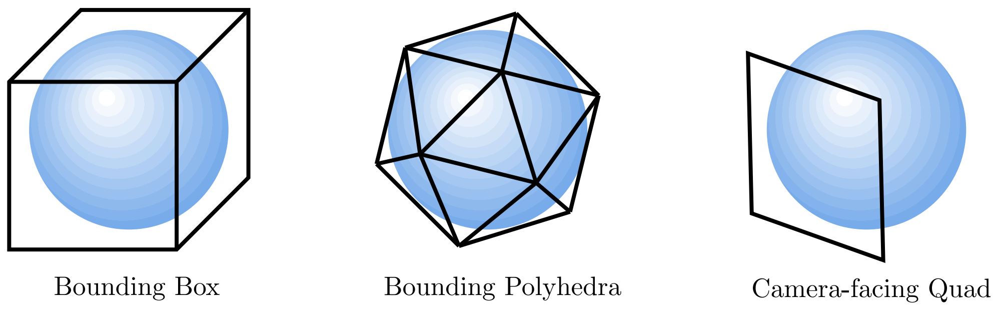


Figure 3.8: Various approaches to enclosing a surface within a bounding mesh.

Given a surface that we wish to render (such as a sphere) and a low-vertex mesh that encloses it on-screen, we must now determine two things — should we discard or keep the pixel, and if we keep it, how much should we adjust the depth. This can be determined using the principle of ray tracing. We define a ray, which originates at the camera and proceeds towards a point on the bounding mesh's surface. A ray is a representation of an infinite line, with an origin $\vec{p}$ and direction $\vec{d}$. The ray is parameterised by a scalar t, measuring the progression along the ray:

$$\vec{r}(t) = \vec{p} + t\vec{d} \tag{3.1}$$

To answer our two questions, we may simply compute the value of t at which an intersection occurs with our surface. If no such value exists, there is no intersection and we discard the pixel. If such an intersection exists, we adjust the depth of that pixel to match the intersection. We may also calculate the normal of the surface at this point, and use this to calculate the shading based on the direction of a light source. All together, these gives us a pixel-accurate representation of our surface.

In general, a surface S can be written as a general form:

$$S(\vec{x}) = 0 \tag{3.2}$$

where $\vec{x}$ is a general point in space. Therefore, to find where a ray would intersect the surface, points must be found that satisfy both the ray equation (equation 3.1) and the surface equation (equation 3.2). This is shown pictorially in Figure 3.9. The points at which the two intersect are given by the solutions of the equation:

$$S(\vec{p} + t\vec{d}) = 0 \tag{3.3}$$

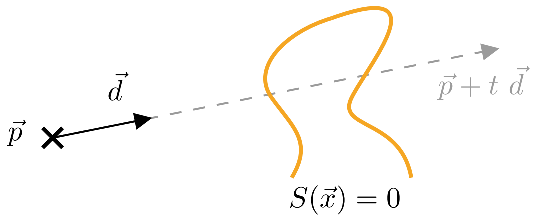


Figure 3.9: Intersection of a ray with an arbitrary surface.

Solving equation 3.3 depends on the form that $S(\vec{x})$ takes. An infinite plane is the most simple surface, as it yields a linear equation in t that results in one intersection. The next set of surfaces are those that have at most two intersections, which are a common use case for ray casting.[137,148] One important category of surfaces that has at most two intersections are the **quadric surfaces**, and include spheres, ellipsoids, hyperboloids, paraboloids and cones. Quadric surfaces are the three-dimensional analogues of the conic sections (circles, ellipses, parabolas and hyperbolas), and can be generally expressed in terms of a 3×3 matrix $\vec{Q}$, a vector $\vec{P}$, and a scalar R:

$$S(\vec{x}) = \vec{x}^T \mathbf{Q} \vec{x} + \vec{P} \cdot \vec{x} + R = 0$$

The solution of equation 3.3 for this category of surfaces yields a quadratic in t, which can be solved analytically. This is why ray casting is a common solution employed in molecular visualisation — the spheres and cylinders commonly used for atomistic representations can be intersected with rays analytically.

For more complex functions, the solution to equation 3.3 may lead to cubics and other higher-order polynomials. For example, ray casting a torus (which appear in the geometric form of the solvent-excluded surface) involves solving a quartic in t. Commonly, a numeric solution must be sought for these higher order equations, using root-finding methods such as Newton-Rhapson or root-isolation methods such as Sturm's theorem.[149]

The equations for visualising spheres, cylinders and other quadric surfaces are well described in the literature, and find widespread use in modern molecular visualisation. In the following section, I will discuss the equations used for some simple shapes, and further work I have performed in defining new shapes to enhance molecular visualisation.

Sphere

One of the simplest surfaces to calculate a ray intersection with is a sphere. A sphere is a surface consisting of all points that lie some distance R (the radius) from its centre $\vec{c}$:

$$S(\vec{x}) = \|\vec{x} - \vec{c}\|^2 - R^2 = 0 \tag{3.4}$$

Therefore, the two intersections occur when the distance between the centre and the ray (equation 3.1) is equal to R:

$$(\vec{p} - \vec{c}) \cdot (\vec{p} - \vec{c}) + 2t(\vec{p} - \vec{c}) \cdot \vec{d} + t^2 \vec{d} \cdot \vec{d} - R^2 = 0 \tag{3.5}$$

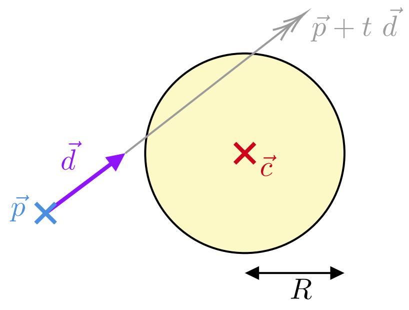


Figure 3.10: Intersection of a ray with a sphere. The sphere is defined by a radius R and centre $\vec{c}$, whilst the ray is defined in terms of its origin $\vec{p}$ and direction $\vec{d}$.

Figure 3.10 shows the relevant quantities for this problem. To simplify matters, the offset between the ray's origin and the centre of our surface $(\vec{p} - \vec{c})$ can be condensed as a single vector $\vec{q}$. This yields the ray-sphere intersection equation in the form as computed by Narupa, which is quadratic in t.

$$(\vec{d} \cdot \vec{d})t^2 + 2(\vec{q} \cdot \vec{d})t + \vec{q} \cdot \vec{q} - R^2 = 0 \tag{3.6}$$

The number of solutions to equation 3.6 is equal to the number of intersections the ray would have with the surface. For a sphere, the only possibilities are 0 (the ray misses the sphere), 1 (the ray passes by the sphere at a tangent) or 2 (the ray enters and leaves the sphere). Generally in rendering, we do not need to consider the edge case (in this case, where the intersection touches the surface), as the likelihood that there is exactly 1 solution is extremely small. To determine the values of t at which an intersection occurs, equation 3.6 can be solved using the quadratic equation:

$$t = -\frac{\vec{q} \cdot \vec{d}}{\vec{d} \cdot \vec{d}} \pm \sqrt{\underbrace{\left(\frac{\vec{q} \cdot \vec{d}}{\vec{d} \cdot \vec{d}}\right)^2 - \frac{\vec{q} \cdot \vec{q} - R^2}{\vec{d} \cdot \vec{d}}}_{\Delta}} \tag{3.7}$$

Depending on the sign of the discriminant Δ, there are either two real values of t ($\Delta > 0$) or two complex values ($\Delta < 0$). As t cannot take complex values, this corresponds to the case where the ray did not intersect the sphere. Therefore, in an implementation of a ray-sphere intersection, the discriminant is calculated first. If it is negative, then the ray has missed the sphere and hence that pixel should be discarded. If it is positive, then the intersection occurs at the closer of the two intersections (where equation 3.7 has a negative second term).

Performance

Now that I have laid out how to perform ray casting using the example of a sphere, we can compare its performance to traditional mesh-based rendering.

The approaches to visualising a sphere include:

1. Ray casting a sphere contained within a cube. As we do not require surface normals, we can represent each corner as a single vertex and hence this shape has 8 vertices.

2. Ray casting a sphere using a quad (a square made of 2 triangles and 4 vertices) that faces the camera.

3. A mesh of an icosahedron, with 12 vertices. This makes a poor approximation of a sphere except at great distance, and is included purely as a benchmark.

4. Two successive iterations of an icosphere, with 42 and 205 vertices respectively. This is achieved by taking an icosahedron and subdividing each face into four triangles, before

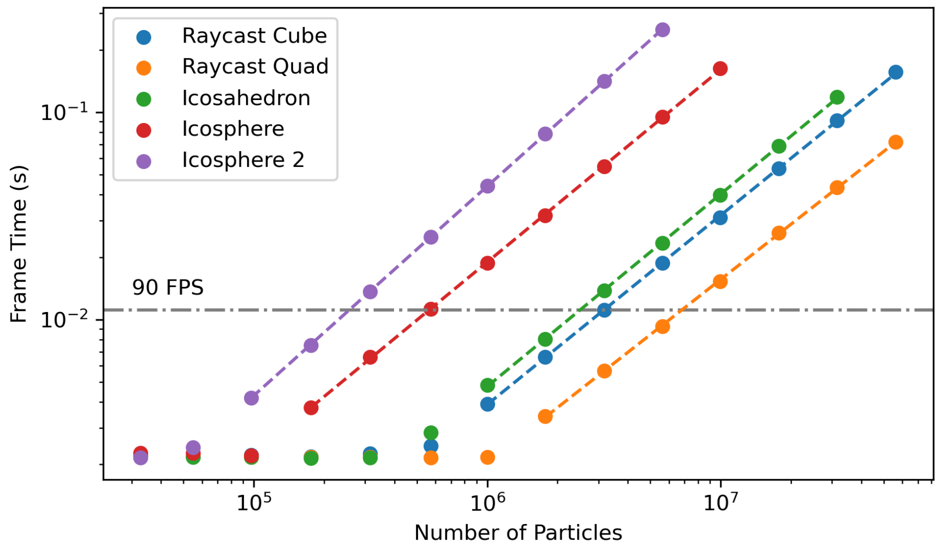


Figure 3.11: Time taken to render each frame using various representations of spheres.

projecting out onto a sphere. This is a common approach to generating spheres of arbitrary resolution, and was employed in NanoSimbox.

Figure 3.11 illustrates the time taken to render a single 1024 $\times$ 1024 image of a regular cubic lattice of particles, each with a randomly assigned scale and colour. The graph shows that a log-log plot produces a straight line fit, with approximately equal gradients but differing intercepts. This is indicative of the linear relationship between frame times and the number of particles — doubling the number of particles doubles the amount of work the GPU has to perform. These lines plateau at around 200 FPS, as there is an upper limit on the number of frames that the program allows to be produced per second.

By inspecting where these lines cross a given line, we can compare their relative performance by seeing how many particles can be rendered at a given frame rate. Figure 3.12 shows how many spheres can be rendered using procedural instancing, using both ray casting approaches and standard mesh approaches. The results clearly show that reducing the number of vertices has a large effect on performance than the decrease due to a more complex fragment shader required for ray casting.

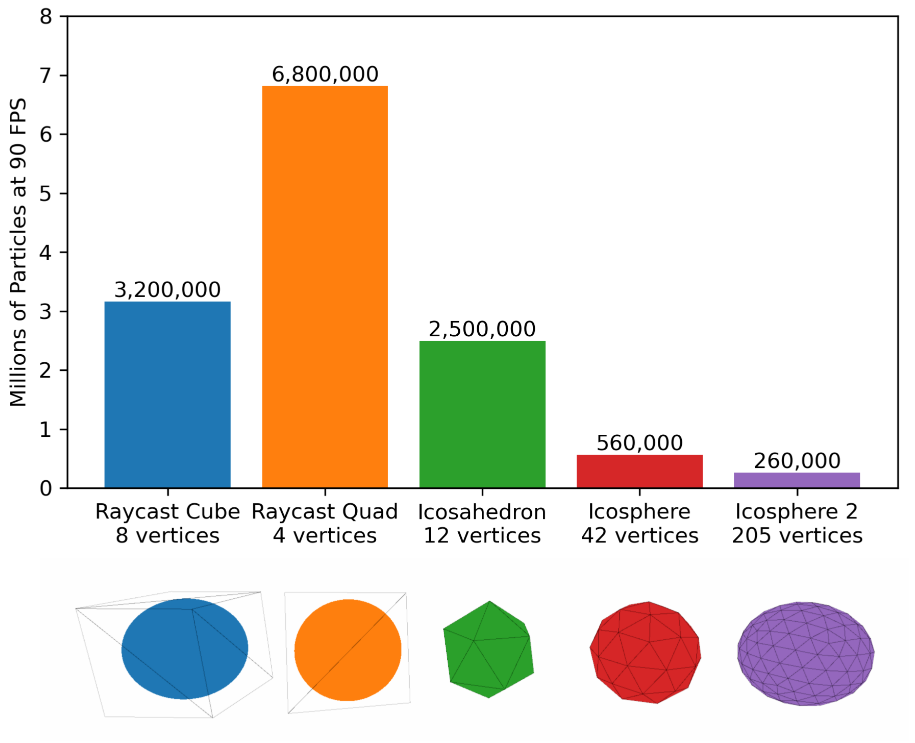


Figure 3.12: Number of spheres that can be rendered at 90 FPS using various approaches.

Cylinder

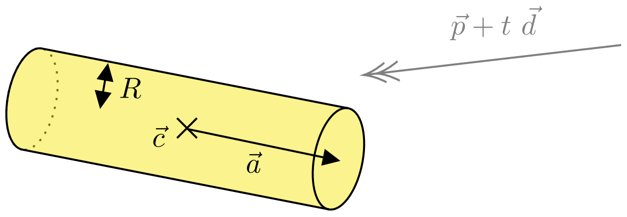


Figure 3.13: General problem for the intersection of a ray with a cylinder. The cylinder is defined in terms of its centre $\vec{c}$, its radius R and its axis $\vec{a}$, whose length determines the length of the cylinder.

With just spheres, space-filling models can be implemented in a molecular visualiser. However, to incorporate bonds into our picture we must implement cylinders. The equations used for a cylinder are superficially similar to that of a sphere, but the radius is now the distance of the ray from an axis, rather than a single point. A cylinder is defined by not just a centre $\vec{c}$ and a radius R, but an axis $\vec{a}$ which determines its direction. The component of the ray that is perpendicular to the axis may be found from the **vector rejection** of $\vec{r}$ from the axis $\vec{a}$:

$$\vec{q}_\perp + t\vec{d}_\perp = \vec{q} + t\vec{d} - \frac{(\vec{q} + t\vec{d}) \cdot \vec{a}}{\vec{a} \cdot \vec{a}}\vec{a} \tag{3.8}$$

Here, $\vec{q}_\perp$ indicate the component of $\vec{q}$ that is not parallel to the axis $\vec{a}$.

By projecting the ray onto its component perpendicular to the cylinder's axis, it can be treated in the same manner as the sphere and hence solved using equation 3.7 (using $\vec{q}_\perp$ and $\vec{d}_\perp$ instead of $\vec{q}$ and $\vec{d}$). However, this defines an *infinite* cylinder that stretches to infinity in both directions. This can lead to graphical artifacts when viewed at angles along the axis of the cylinder. To ensure the cylinder has a finite length, the two end planes of the cylinder can be used to determine the range of values of t that are valid. This is then used to crop the calculated values for an infinite cylinder such that they lie between the two. The length of the axis $\vec{a}$ is used to determine the distance from the centre to each of the two planes marking the ends of the cylinder.

Hyperballs

In two dimensions, instead of quadric surfaces we have the conic sections, traditionally thought of as slices of a double cone. These are namely the circle, ellipse, parabola and hyperbola. The three-dimensional generalisation of a hyperbola is known as a hyperboloid (we shall be considering the degenerate case of a hyperboloid of revolution). Depending on the shape of the hyperbola, there are two different possible hyperboloids: either a hyperboloid of one sheet or a hyperboloid of two sheets. These can be seen in Figure 3.14.

A novel use of hyperboloids is as a representation of a bond between two atoms.[142] These *hyperballs* are a representation that has seen adoption in some common molecular visualisation packages.[36,85,150,151] By smoothly blending between a connected state (the hyperboloid of one sheet) to a disconnected state (the hyperboloid of two sheets), they are ideal for representing systems in which bonds can be broken.[142]

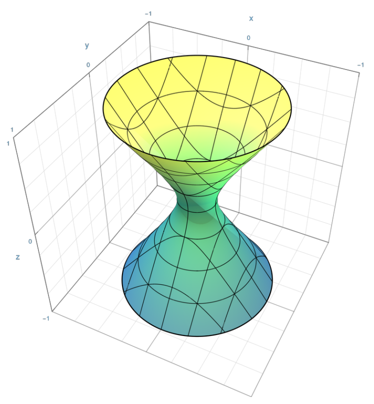
The expression for a hyperboloid that connects two spheres can be reduced to a two dimensional problem, as there is a cylindrical axis of symmetry passing through the centre of both spheres. In this two dimensional problem as given in Figure 3.15, the spheres become circles positioned at $\pm\frac{d}{2}$ along the z axis:

$$\left(z+\frac{d}{2}\right)^2 + r^2 = R_1^2 \tag{3.9}$$

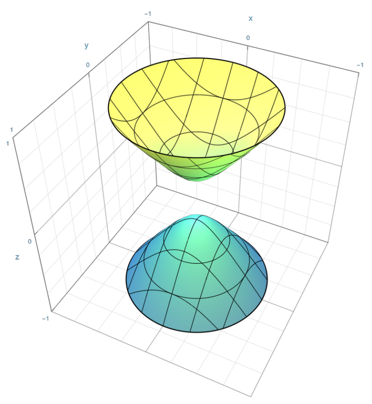
$$\left(z-\frac{d}{2}\right)^2 + r^2 = R_2^2 \tag{3.10}$$

Without loss of generality, we can define a hyperbola that is symmetric about the r axis using the following equation:

$$r^2 = \gamma^2 (z - z_0)^2 + \Gamma \tag{3.11}$$



(a) Hyperboloid of one sheet



(b) Hyperboloid of two sheet

Figure 3.14: Three-dimensional plots of the two general cases for hyperboloids of revolution.

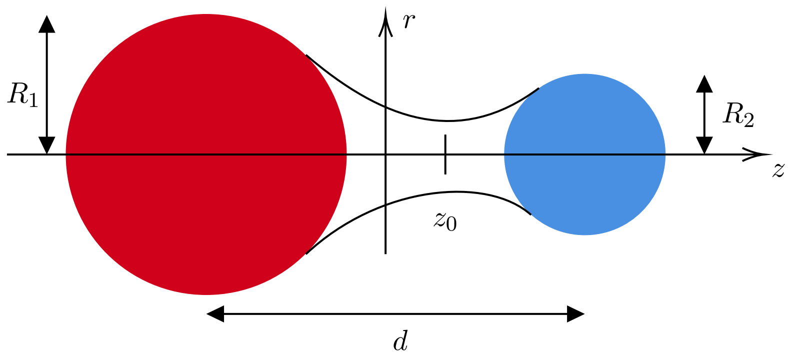


Figure 3.15: Reduction of the problem to two dimensions, with cylindrical (z) and radial (z) axes. The problem is now connecting two circles by a hyperbola.

This hyperboloid has three parameters: Γ, γ and z_0. These parameters can be fixed by considering that the hyperbola must touch each sphere at only one value of z. This yields two equations that fix two of the parameters. We choose γ to be the free parameter that will define the shape of the hyperbola, with the other two values determined by:

$$\begin{aligned}\epsilon &= \frac{\gamma^2}{1+\gamma^2} \\ z_0 &= \frac{R_1^2 - R_2^2}{2d\epsilon} + \frac{d}{2} \qquad (3.12)\\ \Gamma &= R_1^2 - \epsilon z_0^2 \qquad (3.13)\end{aligned}$$

Projecting this result back into three dimensions, we can consider the hyperboloid to be defined in terms of a centre $\vec{c}$, an axis $\vec{a}$ and a parameter γ. Equation 3.11 now becomes:

$$\left\|\vec{q} + t\vec{d}\right\|^2 - (1+\gamma^2)\left\|\vec{q}_{||} + t\vec{d}_{||}\right\|^2 - \Gamma \tag{3.14}$$

This adopts a similar form to that for a sphere or cylinder, and again results in a quadratic equation in t that can be solved analytically. One useful simplification is to note that the parallel projection of a vector is given by $\vec{q}_{||} = (\vec{q}\cdot\vec{a})/(\vec{a}\cdot\vec{a})\vec{a}$. We also note that the magnitude of $\vec{a}$ is irrelevant, as it cancels on top and bottom. Given this, we choose the magnitude of $\vec{a}$ to be equal to $\sqrt{1+\gamma^2}$. This simplifies the resultant quadratic equation to not contain γ:

$$t^2(\vec{d}\cdot\vec{d} - (\vec{d}\cdot\vec{a})^2) + 2t(\vec{q}\cdot\vec{d} - (\vec{d}\cdot\vec{a})(\vec{q}\cdot\vec{a}) + \vec{q}\cdot\vec{q} - (\vec{q}\cdot\vec{a})^2 - \Gamma \tag{3.15}$$

Like in the case of the cylinder, we have to crop the shape at either end to prevent it continuing to infinity.

Figure 3.16 shows how the hyperballs visualisation appears for a caffeine molecule. Hyperballs can be used for covalent bonds purely as an aesthetic choice over the more traditional ball and stick, however it is especially suited for systems where bonds are created and destroyed as it can maintain visual continuity.

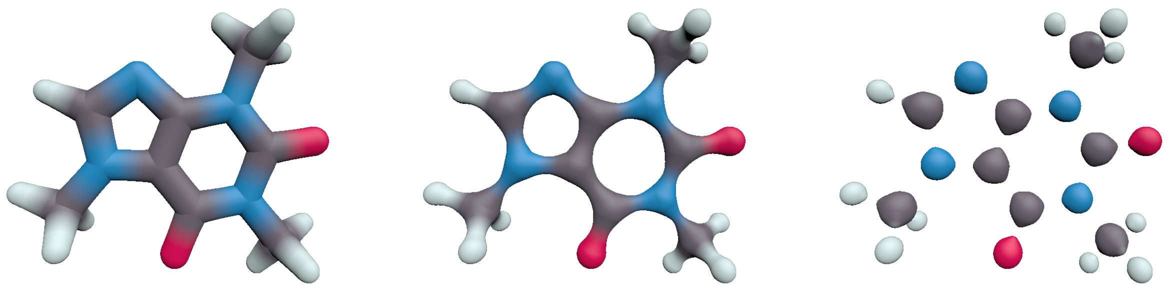

(a) $\gamma = 0.2$ (b) $\gamma = 0.6$ (c) $\gamma = 0.9$

Figure 3.16: Caffeine molecule represented using the hyperballs visualisation.

Chain of Spheres

Up to this point, I have described common ray casting shapes employed in existing molecular visualisation. I will now move on to new primitives that I feel can be employed effectively to communicate additional information. These approaches can be easily incorporated into software that already employs ray casting for its other primitives.

In the case of a sphere, I propose a novel extension that can be used to visualise an infinite chain of spheres. This is a novel application that allows three-dimensional dashed and dotted lines to be visualised, which are often used to draw rulers and partial bonds between atoms.

Instead of a single sphere with radius R and centre $\vec{c}$, now consider an infinite number of spheres parameterised by some integer $n \in \mathbb{Q}$. The centre of each of these spheres is given by:

$$\vec{c}_n = \vec{c} + n\vec{a}$$

The intersection of the ray with the n-th sphere yields a quadratic equation not only in t but also in n:

$$\begin{aligned} \|\vec{r}(t) - \vec{c}_n\|^2 &= R^2 \\ 0 &= n^2\vec{a}\cdot\vec{a} - 2n\vec{a}\cdot\vec{q} + \vec{q}\cdot\vec{q} - 2nt\vec{a}\cdot\vec{d} + 2t\vec{d}\cdot\vec{q} + t^2\vec{d}\cdot\vec{d} - R^2 \end{aligned}$$

As before, the values of t in which an intersection can occur can be found by:

$$t = -\frac{(\vec{q}-n\vec{a})\cdot\vec{d}}{\vec{d}\cdot\vec{d}} \pm \sqrt{\underbrace{\left(\frac{(\vec{q}-n\vec{a})\cdot\vec{d}}{\vec{d}\cdot\vec{d}}\right)^2 - \frac{(\vec{q}-n\vec{a})\cdot(\vec{q}-n\vec{a}) - R^2}{\vec{d}\cdot\vec{d}}}_{\Delta}} \qquad (3.16)$$

In the case of a single sphere, the discriminant was a fixed value in terms of the ray and the centre $\vec{c}$. The sphere was only intersected if this discriminant was greater than 0. Therefore, as equation 3.7 has a discriminant that depends on n, there are only certain values of n for which an intersection occurs. The discriminant Δ is a quadratic in n:

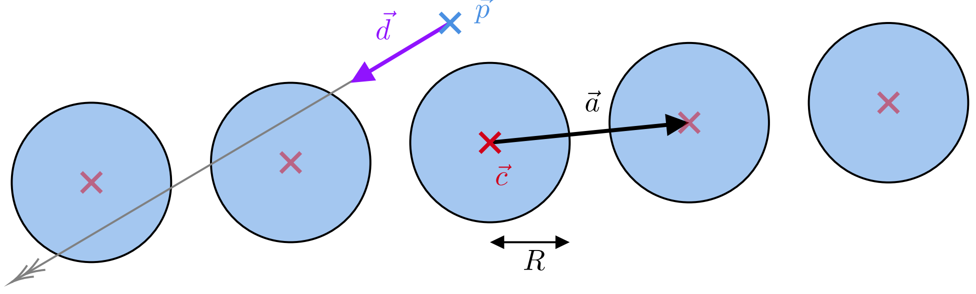


Figure 3.17: The intersection of a ray with an infinite number of spheres spaced at integer multiples of $\vec{a}$ from a centre $\vec{c}$

$$\Delta = n^2\left(\frac{(\vec{a}\cdot\vec{d})^2}{(\vec{d}\cdot\vec{d})^2} - \frac{\vec{a}\cdot\vec{a}}{\vec{d}\cdot\vec{d}}\right) + 2n\left(\frac{\vec{q}\cdot\vec{a}}{\vec{d}\cdot\vec{d}} - \frac{(\vec{a}\cdot\vec{d})(\vec{q}\cdot\vec{d})}{(\vec{d}\cdot\vec{d})^2}\right) + \frac{R^2}{\vec{d}\cdot\vec{d}} + \frac{(\vec{q}\cdot\vec{d})^2}{(\vec{d}\cdot\vec{d})^2} - \frac{\vec{q}\cdot\vec{q}}{\vec{d}\cdot\vec{d}} \tag{3.17}$$

$$= -n^2\frac{\left\|\vec{a}\times\vec{d}\right\|^2}{\left\|\vec{d}\right\|^4} - 2n\frac{\vec{d}\cdot(\vec{a}\times(\vec{q}\times\vec{d}))}{\left\|\vec{d}\right\|^4} + \frac{R^2}{\left\|\vec{d}\right\|^2} - \frac{\left\|\vec{q}\times\vec{d}\right\|^4}{\left\|\vec{d}\right\|^2} \tag{3.18}$$

where vector identities have been applied to simplify the terms. This quadratic is always 'concave-up' (due to the negative coefficient of the n^2 term), and hence either there are no intersections ($\Delta < 0$ for all n) or there is a finite range of n in which there are intersections ($\Delta > 0$).

Solving equation 3.18 for when the discriminant is 0 determines the range of n over which an intersection occurs, $[n_0, n_1]$. Given this range, it is then simple to compute which sphere was hit first. If the direction of $\vec{a}$ (in which n increases) and $\vec{d}$ are aligned, then the lowest integer value of n indicates which sphere is hit first. If $\vec{a}$ and $\vec{d}$ are in opposite directions, then the largest integer of n would be hit first. Given this value of n, the problem is now of intersection of a single sphere, which was handled previously.

Three-dimensional dotted lines can be employed to indicate various types of special bonds, such as hydrogen or ionic bonding. It is also possible to animate these bonds, by having the spheres slowly scroll in a certain direction. This can be used to show the directionality of a hydrogen bond, with the movement indicating the direction of charge from negative to positive. Dotted lines may also be used to indicate partial bond orders. Figure 3.18 gives one example of using three-dimensional dashed bonds, to indicate hydrogen bonds within a protein-ligand complex.

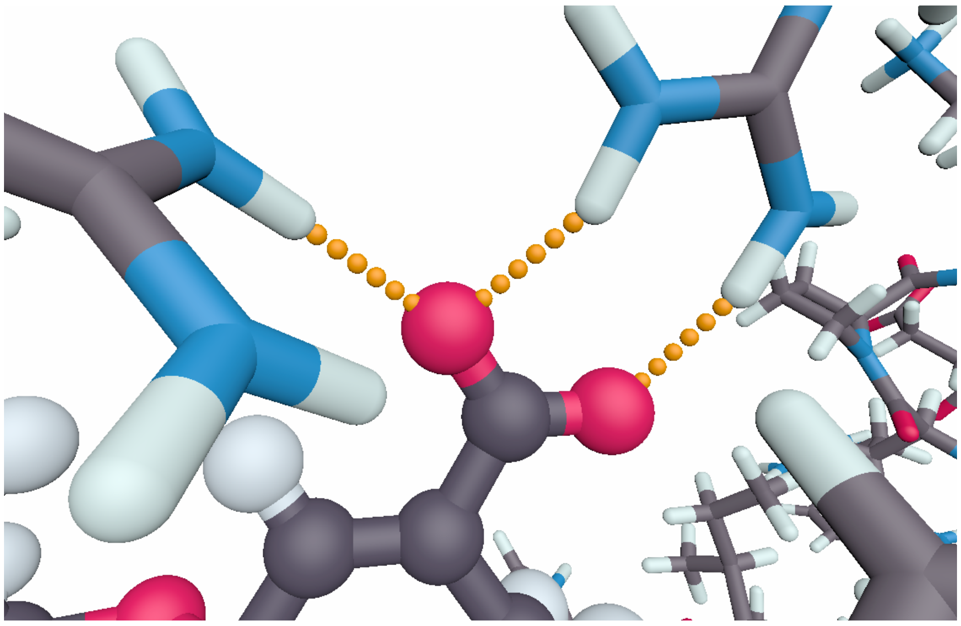


Figure 3.18: Example of a use for a ray casted chains of spheres, to visualise hydrogen bonds in a protein-ligand system. Unlike a flat dotted line, using three-dimensional spheres gives a more consistent visual style, and makes bonds appear more tangible. The spheres can be made to move along the bond edge, indicating the directionality of the hydrogen bonds.

Signed Distance Fields

Instead of calculating an intersection between a ray and the surface by solving the intersection equation, numeric techniques may also be applied even when the intersection cannot be solved analytically. For certain surfaces, we may be able to calculate the **signed distance function**, which is a function $f(\vec{x})$ which indicates the distance from the closest point on the surface to the point $\vec{x}$, and is positive outside the shape and negative inside. We may then apply the technique of **ray marching** to calculate an intersection. Whilst not a widely employed in molecular visualisation, signed distance fields have been used for protein visualisation.[152]

To perform ray marching, we start with a ray at its origin (with $t = 0$) and calculate the closest distance to the surface. We can then advance this ray in small increments until either the signed distance field becomes negative or the ray has travelled too far. However, a far more optimal method is **sphere tracing**.[153] Here, we use the fact that the signed distance function gives us the distance to the closest point, and hence gives us a minimum bound on how far we can advance before hitting the surface. Therefore, at each point we evaluate the signed distance field, and advance by this amount along the ray (as shown in Figure 3.19). By repeating this process, we can trace a wide range of surfaces including cones, capsules, torii and prisms. Various geometric techniques such as volumes of revolutions and elongation may also be performed with minor modifications. The surface may also be 'inflated' by an arbitrary radius, allowing various curved shapes such as rounded boxes to be generated.

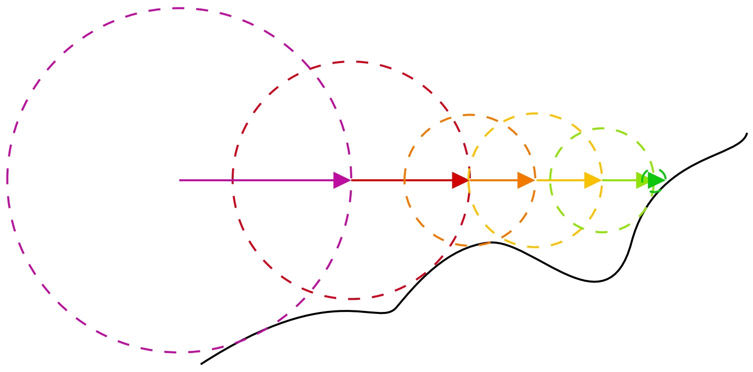
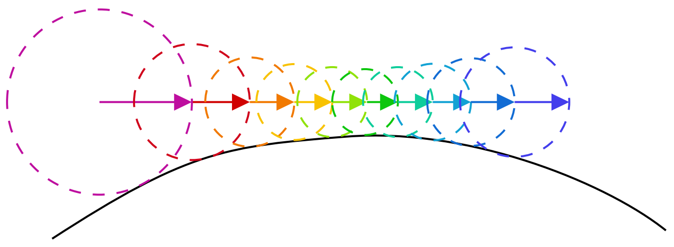


Figure 3.19: Example of sphere marching a signed distance field.

Sphere marching is an effective technique, though it performs poorly near the edges of objects. As seen in Figure 3.20a, as the ray passes tangentially to a surface, sphere tracing can result in many iterations spent passing by the surface without intersection.

For convex shapes, sphere tracing may be improved. By computing the closest point (either directly or by using the gradient of the signed distance field), we know that there are no intersections on this side of the tangent plane.[153] Figure 3.21 compares standard sphere tracing to convex sphere tracing for the wheel shape. The large number of iterations that occur at the edge of objects in sphere tracing is a well-known pitfall of sphere tracing.[154]


(a) Sphere tracing missing a surface, but taking many iterations.

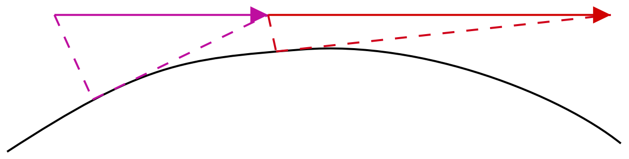

(b) Sphere tracing a convex object using tangent planes, taking minimal iterations.

Figure 3.20: Comparison of ray tracing using standard sphere tracing and exploting the property of a convex shape using tangent planes.

In the oxDNA coarse-grained model of proteins, each DNA base is often visualised as three parts: a spherical backbone, a connecting cylinder and an ellipsoid representing the DNA base. One of the terms in the oxDNA forcefield is a stacking interaction, which is related to the relative horizontal alignments of subsequent DNA pairs.

An alternative approach to the spheroid commonly used for depicting the DNA base in an oxDNA simulation would be to use a wheel shape, as shown in Figure 3.22b. This shape is a short cylinder with rounded sides, or equivalently it can be seen as a torus with the center filled in. Mathematically, there is no easy method for determining the exact intersection of a

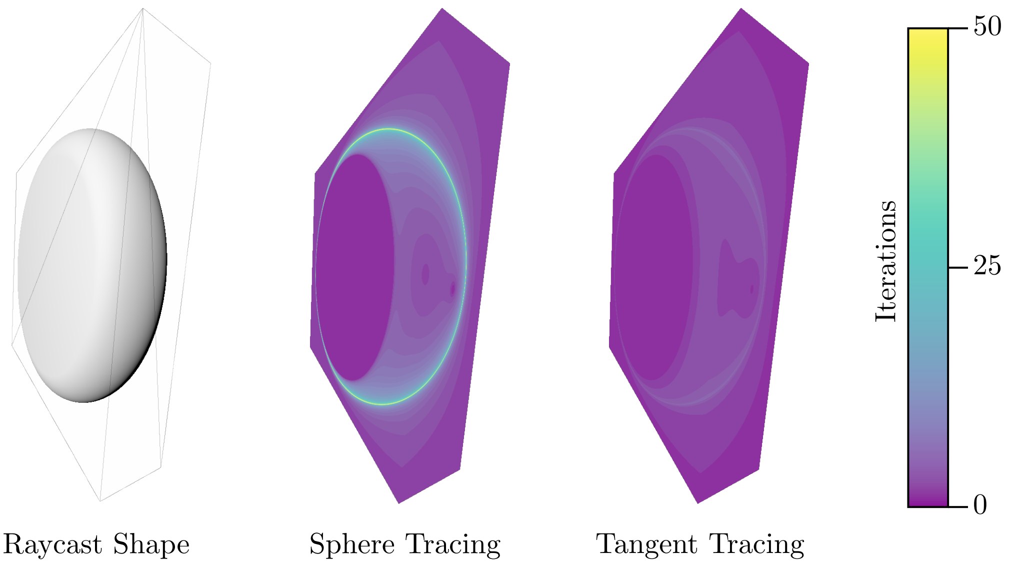
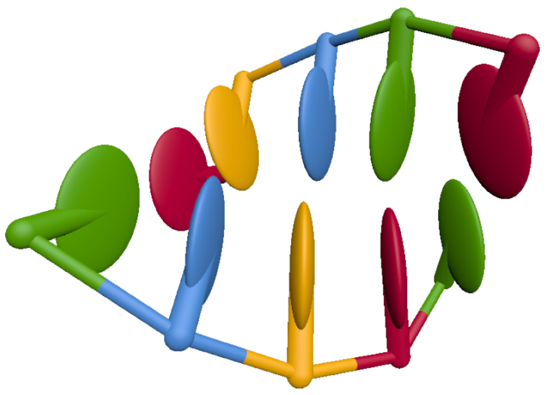


Figure 3.21: Comparison of sphere tracing and tangent tracing for a wheel shape. The colour of the image indicates the number of iterations required before conversion to a given accuracy.

ray and this shape. This could be done by considering it a combination of a cylinder and a torus, but the solution for the torus would require solving a quartic equation.

A better approach to this shape is ray marching. The shape can be seen as the set of points which are some distance d from a disk embedded in three-dimensional space. As we can determine the closest point on the surface, we may employ the tangent tracing method proposed in the previous section. As shown in Figure 3.21, tangent tracing for this shape involves far fewer iterations when the curved edges of the shape are considered.

Figures 3.22a and 3.22b compare these two approaches for visualising DNA bases for oxDNA. As the wheel shape has two large flat surfaces, it is far clearer where the top and bottom of the base are. In comparison, it can be difficult to determine the exact orientation of the ellipsoids, and hence it is less clear how subsequent bases are aligned. This is important in the oxDNA force field, as there are forcefield terms which are based on the relative alignment of subsequent base pairs.


(a) Spheroid-shape base pairs.

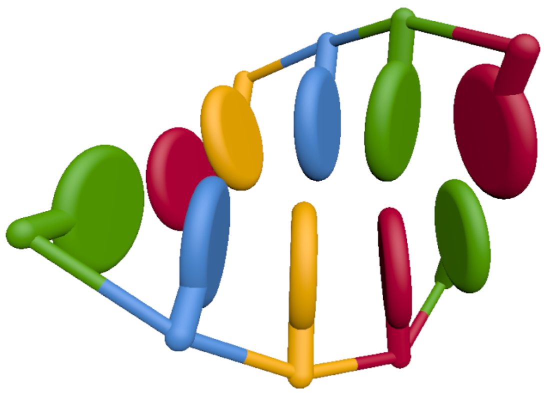

(b) Wheel-shaped base pairs.

Figure 3.22: Comparison of spheroid- and wheel-shaped base pairs for visualising a DNA helix. The flat faces of the wheel shapes gives a stronger visual indication of their direction compared to the smooth spheroids.

3.3 Periodic Boundary Conditions

Whilst molecular dynamics can simulate systems that contain millions of individual particles, generally they still must lie within a finite volume (called the **simulation box**). The use of this finite box results in boundary-related issues when attempting to simulate bulk or long-range properties. How these boundaries of the simulation are handled can be broken down into three approaches:

1. **free**, where there are no boundaries and particles move freely. If looking at a single molecule in isolation, this is the default. However, in a diffuse system, the particles will gradually spread apart.

2. **hard**, where the particles are limited to the simulation box, either by preventing particles from moving outside the box or by reflecting their velocities when they reach the edge.

3. **periodic**, where the simulation box is treated as one of an infinite number of identical copies tessellated together. A particle leaving one side of the box can be considered to be entering from the other side.

Using periodic boundary conditions, each particle now interacts with an infinite lattice of other particles, which means each pairwise interaction potentials is an infinite sums. However, if these interatomic potentials are short-ranged and can be truncated at a distance less than half that of the shortest box side, each particle can be considered to interact only with the closest copy of each other particle. This is known as the **minimum image convention**.

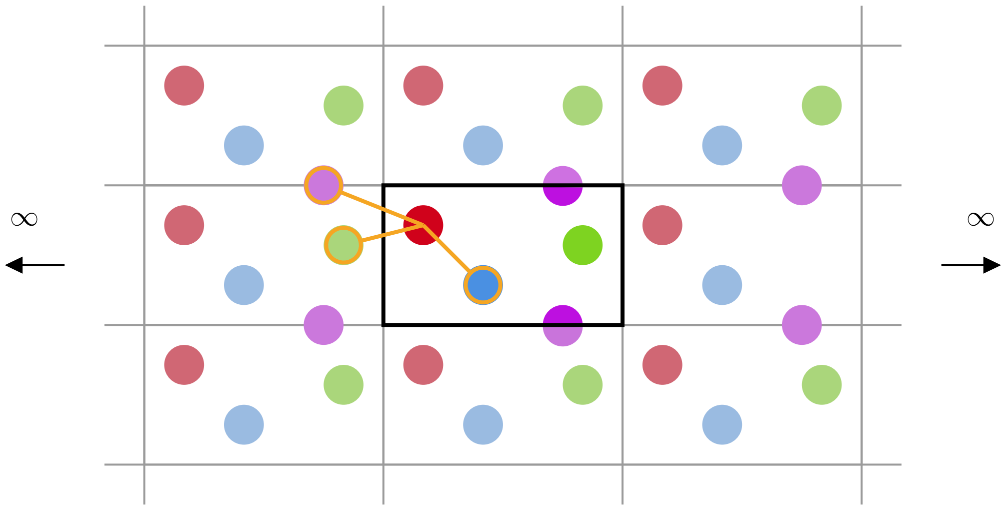


Figure 3.23: Illustration of a 2D periodic system consisting of four coloured particles. The orange lines indicate the minimum image convention for determining the atoms with which the red atom will interact with.

Figure 3.23 illustrates this approach for a simple two-dimensional system of four atoms. Here, though all four atoms exist within the core simulation box, the red atom interacts with periodic images of the green and purple atoms.

The implementation of periodic boundary conditions within a molecular dynamics software can involve either automatically wrapping the particle coordinates back within the unit cell after each timestep, or leaving them unrestricted. In the second case, the wrapping of particles to lie within the simulation box occurs only when they are visualised, rather than within the logic of the simulation.

Though widespread in molecular simulations, the visualisation of periodic systems has several issues. These artifacts include:

1. Particles (often visualised as spheres) which cross the boundary teleport instantaneously from one side of the simulation box to the other as they cross the periodic boundary.

2. Bonds between particles that are initially close by may suddenly span the simulation boundary when one of the two bonded particles crosses the boundary, as illustrated in Figure 3.24.

The first issue does not exist for static structures, and can be tolerated for dynamic trajectories when viewed on a two-dimensional screen. However, in virtual reality, sudden jarring movements of objects may lead to disorientation and confusion. The second issue is often tackled on the visualisation end, by removing bonds that are above a certain cutoff. However, I argue that this is a dangerous decision to allow a visualisation software without user input, as the existence of bonds within a system may be very important. By removing certain bonds

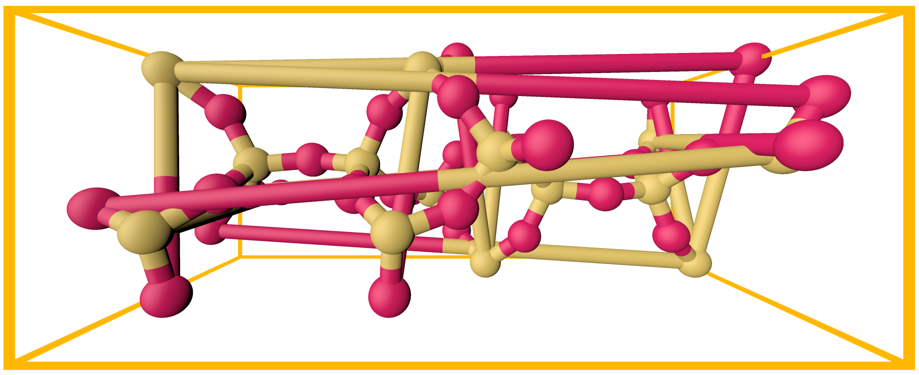


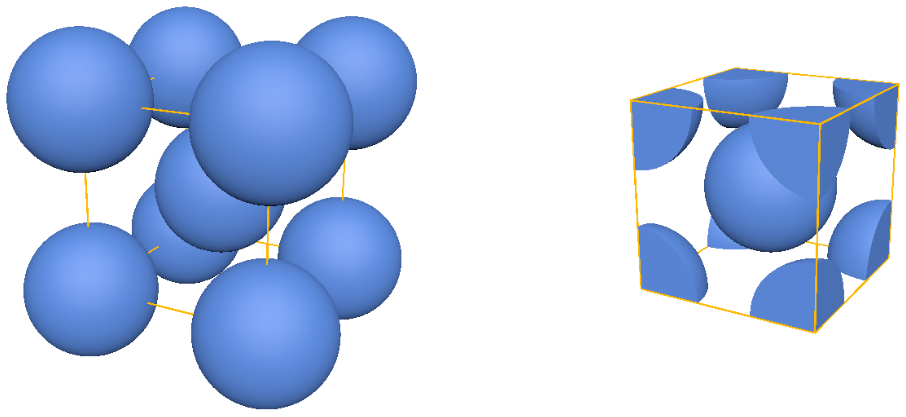
Figure 3.24: Ball-and-stick representation of an BRE zeolite unit cell, illustrating the issue of bonds that are linking two atoms at opposite ends of the cell. When viewed as a dynamic simulation, atoms will also teleport from one side of the cell to the other. These issues appear not just in Narupa iMD, but other visualisation software such as VMD, PyMOL and ChimeraX. Oxygen atoms are in red and silicon atoms in beige.

that cross periodic boundaries, atoms that are normally tetrahedrally arranged may suddenly have three bonds. The user must then have to understand why these bonds are not visible and remember they still exist.

Another way to approach the second issue is to simply draw more copies of the simulation box. This approach can be used in most existing applications such as VMD, PyMOL, ChimeraX and nglview. However, this still will have to omit bonds that lie at the edge of the unit cell. Duplicating the cell also does not prevent the issue of atoms appearing and disappearing, instead merely shifting these problems to the edges of the furthest cells. Therefore, a different approach is required.

For inspiration, we may turn to the depictions often used to describe atomic packing fractions in undergraduate textbooks.[155–157] and illustrated in Figure 3.25. In this approach, the atoms are cropped to be within the unit cell, resulting in slices such as hemispheres. This approach aids in the understanding of how many atoms appear in the unit cell, as the total volume of the atomic spheres is unchanged from that of a non-periodic system. In the case of dynamic systems, it also has the advantage of avoiding the sudden movement of atoms. Instead, as a sphere moves across the boundary, it is smoothly sliced away whilst the removed section slowly reappears at the opposing side of the cell.

This novel approach to visualising periodic systems is made possible by the use of ray-casting impostors to represent atoms and bonds. To implement this, the following three modifications can be made to existing ray casting shaders to enable this functionality.



(a) Uncropped.

(b) Cropped.

Figure 3.25: Comparison between a cropped and uncropped simulation box for a typical BCC unit cell. The cropped approach contains exactly two spheres by volume, whilst the uncropped version gives an erroneous impression there are 9 atoms in the cell.

Minimum image convention for bonding

The common representation of a bond is a simple pair of integers, indicating the indices or IDs of the two atoms that the bond connects. Therefore, the actual direction and length of the bond are not explicitly stated, and instead implied by the current positions of its two endpoints. In a non-periodic system, the standard method for defining the vector $\vec{b}$ along which a bond lies would be to take the difference in positions of the two atoms:

$$\vec{b} = \vec{p}_B - \vec{p}_A \tag{3.19}$$

To handle bonds in a periodic system, however, we may consider bonds in a similar way that interactions are considered in periodic systems — using the minimum image convention. The reason to do this is that bonds are usually just an indication of the topology of a molecule. However, the bonds that appear in a topology are also often representing a corresponding term in a forcefield (which would be applying the minimum image convention).

Therefore, a more correct bond vector would also utilise the minimum image convention. First, we must define the periodic simulation box. The box is defined by an origin $\vec{p}$ and three lattice vectors: $\vec{a}_1$, $\vec{a}_2$ and $\vec{a}_3$. We can group these vectors together into an affine transformation matrix (see Appendix A) $\mathbf{H}$:

$$\mathbf{H} = \begin{pmatrix} \vec{a}_1 & \vec{a}_2 & \vec{a}_3 & \vec{p} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.20}$$

This transformation matrix converts a unit cube (with side lengths of 1 and a corner at $(0, 0, 0)$. This matrix and its inverse hence allows conversion between positions in simulation space and positions in **fractional coordinates**, which are measured relative to the unit cube.

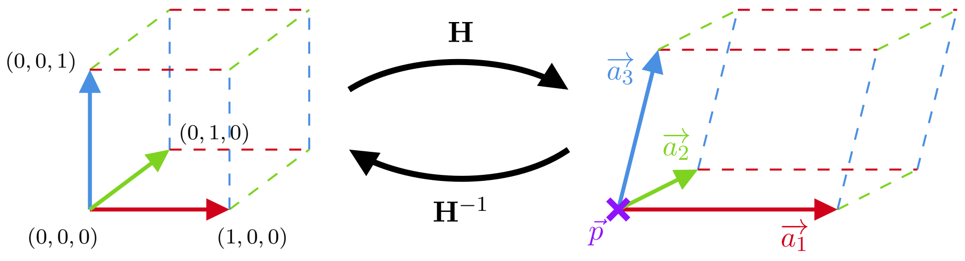


Figure 3.26: Conversion between fractional coordinates (unit cube) and simulation coordinates (simulation box) using $\mathbf{H}$.

In a periodic system, the point $\vec{p}_B$ may be translated by any integer multiple of the three primitive vectors and still represent the same particle. The minimum image convention would be to define the bond vector to be the shortest possible vector $\vec{b}$ such that:

$$\vec{b} = \vec{p}_B + u_1\vec{a}_1 + u_2\vec{a}_2 + u_3\vec{a}_3 - \vec{p}_A \tag{3.21}$$

Calculating the minimum image convention for the bond is therefore a minimisation problem, minimising the length of $\vec{b}$ in equation 3.21 with respect to the image vectors u_1, u_2 and u_3. To perform this for arbitrary triclinic simulation boxes, three steps are required

1. Transform the vector $\vec{p}_B - \vec{p}_A$ into fractional coordinates using $\mathbf{H}^{-1}$. This is the coordinate system of the simulation box, which now takes the form of the unit cube with a corner at $(0, 0, 0)$ and side lengths 1.
2. Wrap this vector to be in the interval $[-0.5, 0.5)$ along each axis. This can be achieved by rounding each component to the nearest integer and subtracting this from the value.
3. Transform this vector back into real space using the matrix $\mathbf{H}$.

Figure 3.27 illustrates the application of the minimum image convention for bonds. Compared to Figure 3.24, there are no longer bonds that stretch across the whole periodic cell. However there are still issues, with the bonds only being attached at one end. To fix this, we must render duplicates of objects near the simulation box faces around the other side.

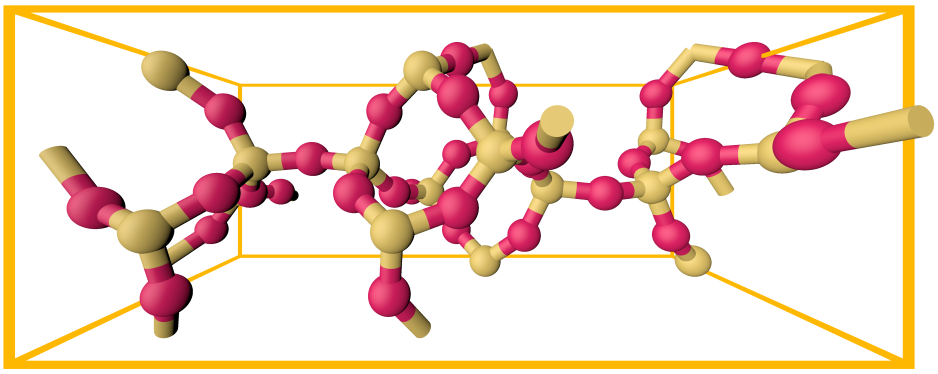


Figure 3.27: Ball-and-stick representation of an BRE zeolite unit cell, with bonds using the minimum image convention. This ensures that bonds no longer cross the whole cell when instead they should cross the boundary. Oxygen atoms are in red and silicon atoms in beige.

Repeated rendering of objects

Instead of stretching across the simulation box, the above method will ensure that each particle A connects to the nearest copy of particle B. However, this only shows the bond across one boundary, where it should ideally be shown both crossing one side and emerging on the other side. To address this, objects which are spanning periodic boundaries need to be duplicated and visualised on the other side of the simulation box.

For this duplication, the **geometry shader** will be used. This stage of the graphics pipeline acts after the triangles of the mesh have been generated but before individual pixels have been shaded. It allows the duplication and manipulation of individual triangles before rasterization and is well suited for simple duplication and translation of the various mirrors.

In a naive approach, we would have to generate 26 duplicates of each object. Not only is this expensive and wasteful (as we intend to later crop any part of an object not in the cell), this is also limited by hardcoded limits on how much additional geometry can be created by the geometry shader. Therefore, only the duplicates which would lie within the simulation box should be created.

By determining across how many faces of the unit cell an object lies across, we can determine which duplicates need to be drawn. As different geometries may be drawn, such as spheres, cylinders and hyperboloids, it is desirable to use a method that is applicable in a wide variety of cases. Therefore, we can abstract away the exact geometry of an object by determining a **bounding volume**, which is a volume that completely contains the object in question. The idea behind this is that it is better to be overzealous and draw some unnecessary copies than it is to omit ones that are important. By definition, if an object's (perhaps complex) geometry intersects with another volume, then so will its bounding volume.

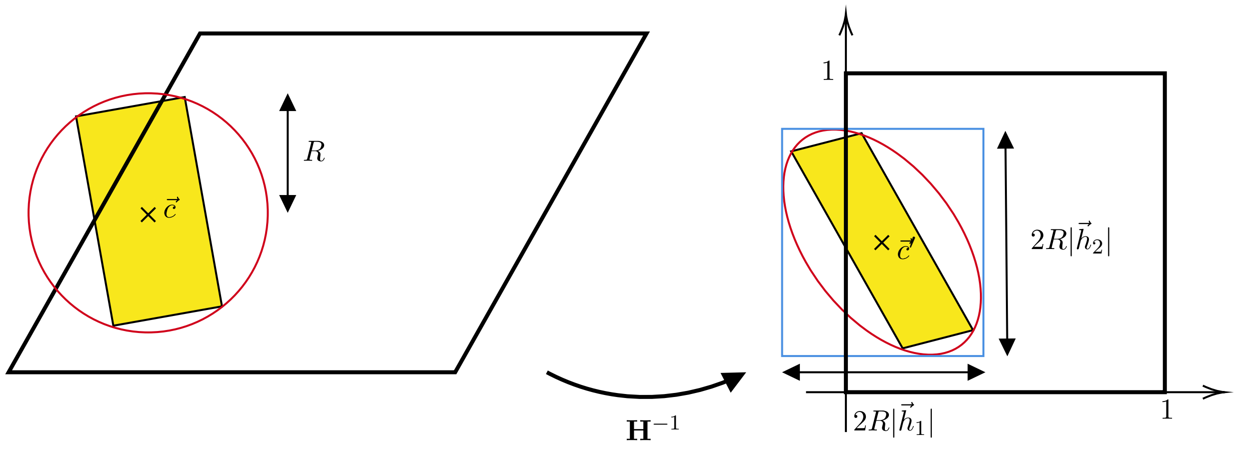


Figure 3.28: 2D example of how an object (yellow) in a non-orthorhombic simulation box can be enclosed within a bounding sphere in simulation space (red), which when transformed into scaled coordinates becomes an ellipse. In turn, this ellipse can be enclosed within an AABB (blue).

For the class of geometry that we deal with, a **bounding sphere** is most easily determined. This is a sphere (of centre $\vec{c}$ and radius R), in which the object we wish to draw is entirely contained. When transformed into fractional coordinates by the matrix $\mathbf{H}^{-1}$, this becomes an ellipse as shown in Figure 3.28.

For determining which edges are crossed by the object, the axis-aligned bounding box (AABB) that contains this ellipsoid can be determined. This is a cuboid that encloses the ellipsoid and defines the minimal and maximal coordinates along the three axes. The central value is defined by the transformed centre of the sphere, and the bounds can be defined in terms of the lengths of the rows of $\mathbf{H}^{-1}$:

$$\mathbf{H}^{-1} = \begin{pmatrix} \vec{h}_1 \\ \vec{h}_2 \\ \vec{h}_3 \end{pmatrix}$$

$$(\mathbf{H}^{-1}\vec{c})_x - \left\|\vec{h}_1\right\| R \leq x \leq (\mathbf{H}^{-1}\vec{c})_x + \left\|\vec{h}_1\right\| R$$

$$(\mathbf{H}^{-1}\vec{c})_y - \left\|\vec{h}_2\right\| R \leq y \leq (\mathbf{H}^{-1}\vec{c})_y + \left\|\vec{h}_2\right\| R$$

$$(\mathbf{H}^{-1}\vec{c})_z - \left\|\vec{h}_3\right\| R \leq z \leq (\mathbf{H}^{-1}\vec{c})_z + \left\|\vec{h}_3\right\| R$$

For each of the 26 possible duplicates, we can determine if their AABB would overlap with the simulation box and hence should this duplicate be drawn. By filtering this out, the amount of additional geometry is minimised as shown in Figure 3.29.

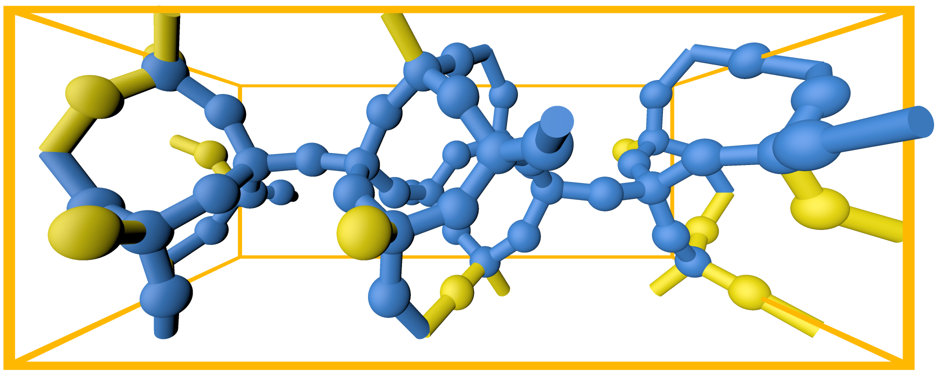


Figure 3.29: Ball-and-stick representation of an BRE zeolite unit cell, indicating the original geometry (blue) as well as additional geometry (yellow) drawn using the geometry shader. Note this does not correspond to drawing 26 duplicate cells, and only geometry near the border is duplicated.

Intersection with simulation box

Using the above techniques, all geometry that should appear in the simulation box is rendered correctly. However, it both fails to solve the problem with objects teleporting from one side to another and introduces a new issue where duplicates of atoms at either side of a periodic cell give an incorrect impression about the number of atoms present to the user. This is an issue that also appears when visualising unit cells when taught to undergraduates. Naively, a body-centred cubic cell contains 9 atoms, with one at each corner and one at the centre of the cell. However, of each atom in the corners, only 1/8th of the atom actually would lie within this specific cell. Therefore, the correct number of atoms that lie in a body-centred cell would be 2.

The approach used to visualise this in some textbooks is to **clip** the sphere, such that the only part of the sphere which appears is that which lies within the cell. This can be performed by using concepts from **constructive solid geometry** (CSG).[158] In CSG, simple primitive objects are combined using Boolean operations, namely union, difference and intersection. This is commonly used in modelling software, as it treats objects as solid entities rather than simple surfaces.

Treating both the simulation box (a parallelepiped) and the object we are rendering (a sphere, cylinder or other simple surfaces) as solid objects, we wish to compute a new object which represents the intersection of these two. The intersection of two solids will be the portion of the solid that lies within both solids. Specifically, here we will only consider objects that have two intersections (namely spheres, cubes, cylinders, hyperboloids and parallelepipeds). We can define their extent purely by two values — the value of t when the ray enters the shape, and the value of t when the ray exits the shape.

In most of these cases, the ray will always enter the solid before leaving it. This broad category will be labelled *finite* shapes, and include spheres, cubes, parallelepipeds and cylinders. Even in the case of a cylinder of infinite length, there is only one viewing direction (directly along the axis) where the ray would never leave or enter the solid. These shapes are contrasted with the hyperboloid, where at certain angles the ray would exit the solid and enter at a later point. In these situations, the points at $t \to \pm\infty$ should be considered to also lie within the solid.

This distinction between the two categories is important when it comes to calculating intersections. Firstly, intersections between two *finite* solids will always yield a new *finite* solid. However, the intersection of a hyperboloid with a *finite* solid may actually yield a complex set of intersections. As the hyperboloids are already clipped (they are clipped at either end to prevent them from tending to infinity), a second clipping with the box will lead to graphical artifacts when rendering hyperballs.

This issue can be resolved by computing the intersection of the hyperboloids clipping planes with the simulation box (yielding a new pair of intersections with a *finite* object), followed by the intersection test with the hyperboloid itself. At this final stage, it is unimportant that there are now four intersection points, as only the closest point is needed for rendering.

To perform the clipping, a general solution for the intersection of two objects (where both have two intersections, and one is *finite*) needs to be computed. Labelling the two objects A (our geometry in question) and B (a finite object such as the simulation box), there are four intersection points to consider:

1. t_A^{in}, the value of t when the ray enters object A.
2. t_A^{out}, the value of t when the ray exits object B.
3. t_B^{in}, the value of t when the ray enters object B.
4. t_B^{out}, the value of t when the ray exits object B.

Assuming the ray intersections both A and B, then the possible geometry of the intersection may be one of 12 possibilities. These are laid out in Figure 3.30. As highlighted above, there is one specific case (the fourth along on the second row) in which the intersection may not be written as a simple entry and exit value of t. However, here the only goal is to determine the value of t of the first intersection, designated $t_{A\cap B}^{in}$, as well as which shape was intersected.

If the entry point $t_{A\cap B}^{in} = t_A^{in}$, then the clipping has had no effect. However, if $t_{A\cap B}^{in} = t_B^{in}$, then the object has been clipped. The normal should be replaced by that of object B to ensure the lighting is correct.

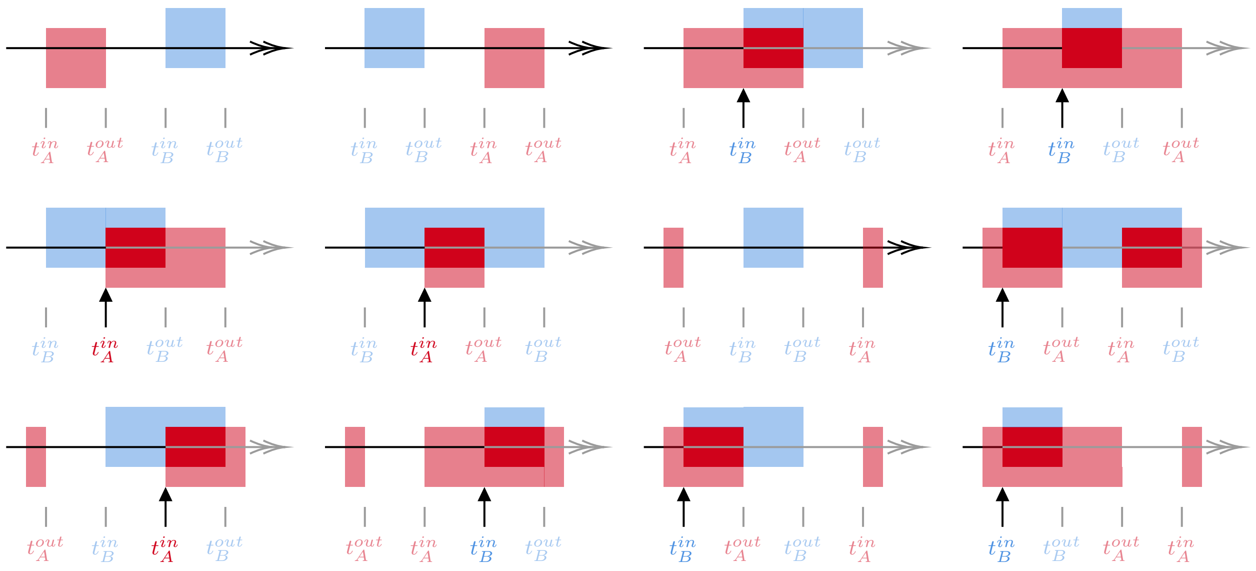

t_A^{in} t_A^{out} t_B^{in} t_B^{out}
t_B^{in} t_B^{out} t_A^{in} t_A^{out}
t_A^{in} t_B^{in} t_A^{out} t_B^{out}
t_A^{in} t_B^{in} t_B^{out} t_A^{out}
t_B^{in} t_A^{in} t_B^{out} t_A^{out}
t_B^{in} t_A^{in} t_A^{out} t_B^{out}
t_A^{out} t_B^{in} t_B^{out} t_A^{in}
t_B^{in} t_A^{out} t_A^{in} t_B^{out}
t_A^{out} t_B^{in} t_A^{in} t_B^{out}
t_A^{out} t_A^{in} t_B^{in} t_B^{out}
t_B^{in} t_A^{out} t_B^{out} t_A^{in}
t_B^{in} t_B^{out} t_A^{out} t_A^{in}

Figure 3.30: Possible intersections (striped region) between two objects A (blue) and B (red), assuming the ray passes through both objects.

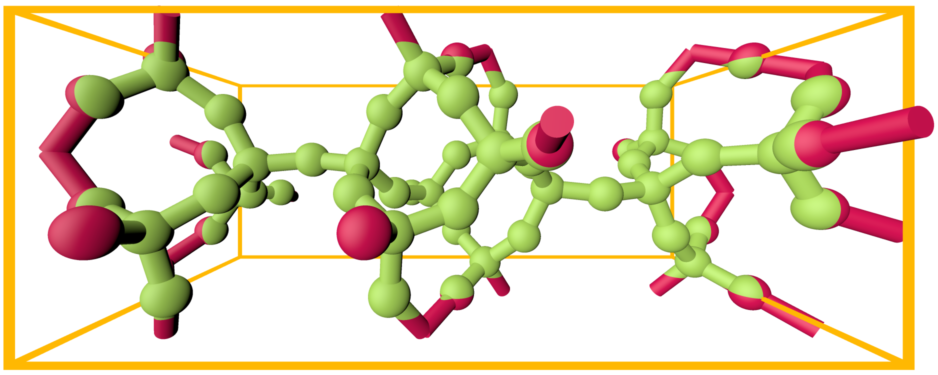


Figure 3.31: A periodic cell of the BRE zeolite structure, showing which parts will be clipped (red) and which lie inside the unit cell (green).

This technique can be applied to my periodic boundary visualisation by replacing each shape with the geometric intersection of it with the parallelepiped of the simulation box. Figure 3.31 illustrates how objects which lie partly across the periodic boundaries will be split.

By discarding the clipped parts, we finally have the full result of applying my periodic boundary visualisation as shown in Figure 3.32. This brings together all three techniques — the bonds have been replaced by their minimum image convention; geometry near the edges of the simulation box has been duplicated on the other side; and finally, the geometry has all been flipped such that it lies in the simulation box.

For viewing this approach in motion, see the video at `https://doi.org/10.6084/m9.figshare.22083572` for a short simulation without my modifications applied, and the video at `https://doi.org/10.6084/m9.figshare.22083575`

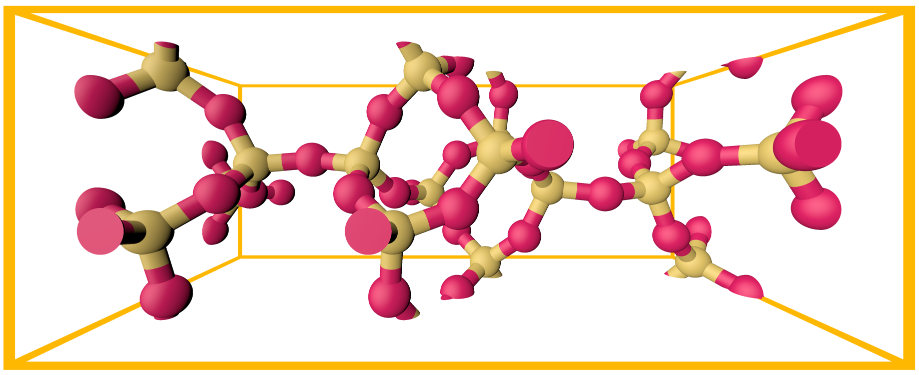


Figure 3.32: A periodic cell of the BRE zeolite structure, using all above-described techniques to implement periodic systems. Oxygen atoms are in red and silicon atoms in beige.

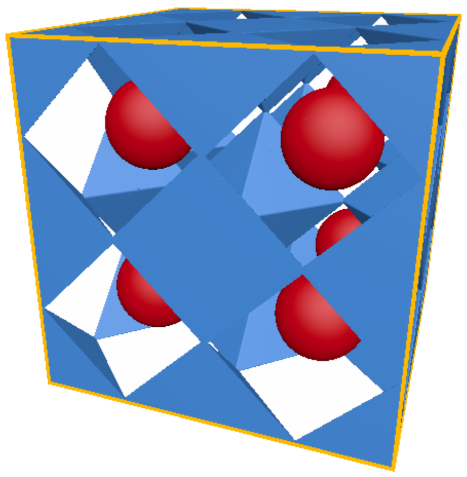
Summary

Up to this point, Narupa iMD and its predecessors have predominantly been employed in the study of small systems or protein-ligand complexes. As such, the visualisations techniques have been developed with an emphasis on protein visualisation. The technique described in this section aims to improve the experience and usability of interactive virtual reality when applied to periodic systems often encountered in the material science domain.

The periodic nature of these systems has often been a challenge for visualisation software, which adopts approaches such as duplicating geometry outside the unit cell, or wrapping entire molecules such that they are not broken across a boundary. Not only does this obscure where the well-defined limits of the simulation box are, it also leads to sudden teleportation of atoms and fragments from one side of the simulation box to the other as the system evolves with time. These sudden movements can be disorientating and confusing, especially when a user is immersed in their system in virtual reality.

When bonds are involved, there is also the issue of bonds that should cross the periodic boundaries, but instead stretch across the whole unit cell. Common approaches to this are to simply remove the bonds in question, which removes valuable chemical insight and can give a false impression of the coordination of chemical species.

This technique of periodic box clipping addresses the above issues. It leverages a familiar technique commonly used to illustrate unit cell occupancy, and is compatible with the ray casting technique employed in the rest of Narupa iMD.


(a) Perovskite structure of $SrTiO_3$.

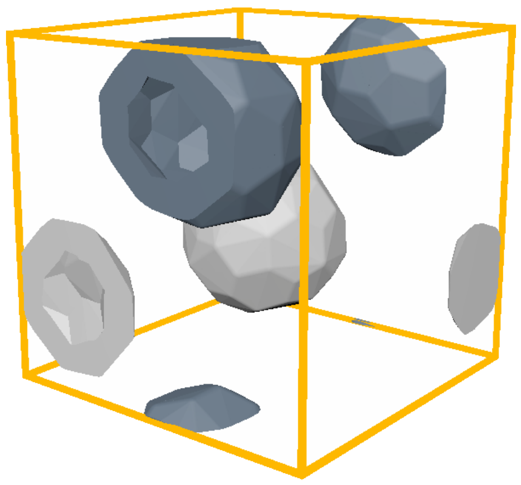

(b) Two fullerene molecules.

Figure 3.33: Two examples of systems visualised using periodic boundaries

3.4 Conclusions

Molecular visualisation is one of the most important uses of computers in chemistry. Through visualisation, we can produce representations of complex systems in a reproducible and well-defined manner.

I have described the graphics pipeline in detail, and illustrated why ray casting and geometry instancing are ideal techniques for molecular visualisation. By illustrating with the relevant examples of spheres and cylinders, I have shown how simple intersection formulae make them ideal primitives to build molecular geometries. To this, I have given a novel approach to ray casting an infinite chain of spheres, which has applications for rendering three-dimensional dotted and dashed lines that can be used for illustrating non-covalent bonds within a system. And finally, I have discussed shapes for which there is no analytic solution, where approaches such as ray marching must be utilised instead. I have illustrated this with an example of a DNA helix, showing how simply adjusting the shape used to represent something can make the three-dimensional layout of a molecule clearer.

I have also described a novel implementation for visualising systems using periodic boundary conditions. By applying the minimum image conventions to bonds, I prevent the issues that faces many visualisation platforms where bonds are drawn across the whole unit cell. As ray tracing is used for many of the primitives in Narupa iMD, I combine this with constructive solid geometry principles to then crop the geometry to be inside the simulation box. Given that ray casting is found in other visualisation packages, it is possible that in the future this technique will be adopted in other software programs. The periodic clipping technique also reduces sudden movements in the users' view, which can be distracting especially virtual reality. Reducing these visual artifacts is also an important goal of the next chapter, which will go into detail about ribbon diagrams and their visualisation.

Chapter 4

Continuity in Secondary Structure Rendering

In this chapter, I will describe my contributions to the rendering of protein secondary structure. I will first lay out the history and development of the ribbon visualisation of protein secondary structure and how it is currently implemented in modern molecular visualisation packages. Specific emphasis will be placed on the interpolation between points, describing the variety of methods employed by available software. Whilst the various approaches to molecular surfaces have been extensively covered in excellent reviews such as Kozlíková et al.[113], ribbon diagrams have been relatively underexplored by reviews of biomolecular visualisation. This extends to the software itself, with papers and documentation often neglecting to describe how exactly the ribbon diagrams are generated. I will therefore review the methods employed in a range of modern software, and how their different approaches can affect the shape of the diagram.

I will then outline the issue of visual continuity that occurs when using ribbon representations for dynamic protein data, and propose a new algorithm I developed that addresses these issues. This method, which I shall refer to as **double-normal interpolation**, defines an ellipsoid ribbon which does not suffer from sudden twists that occur in software such as PyMol, VMD and ChimeraX. I have previously published this method in the Frontiers of Computer Science journal.[24] This chapter discusses the same algorithm, however the text of this chapter is original and goes into greater depth.

4.1 Protein Structure

Proteins are large biomolecules that perform many functions within living organisms, such as providing structure, transport and catalysis. They consist of one or more polymer chains, each formed of a sequence of **amino acids** connected by peptide bonds. Each amino acid consists of a central carbon (denoted the α carbon or Cα), connected to an amide group, carboxylic acid group and a variable side chain. The amide of one amino acid and the carboxylic group of another can undergo a condensation reaction to form a **peptide bond**, as illustrated in Figure 4.1. A chain of amino acids connected by these peptide bonds is known as a polypeptide. Proteins are three-dimensional assemblies of one or more of these polypeptides.

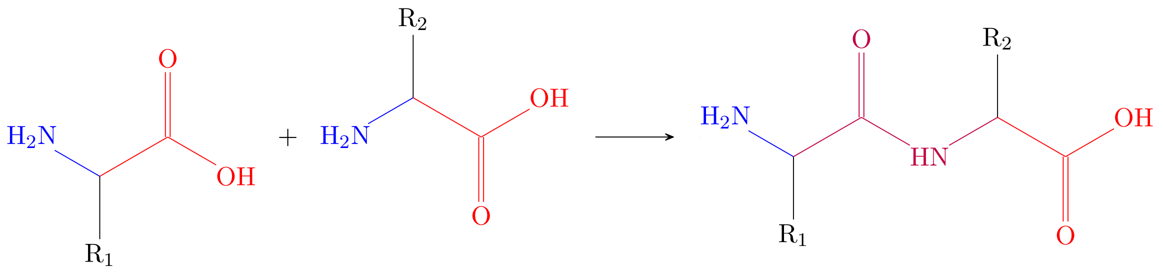
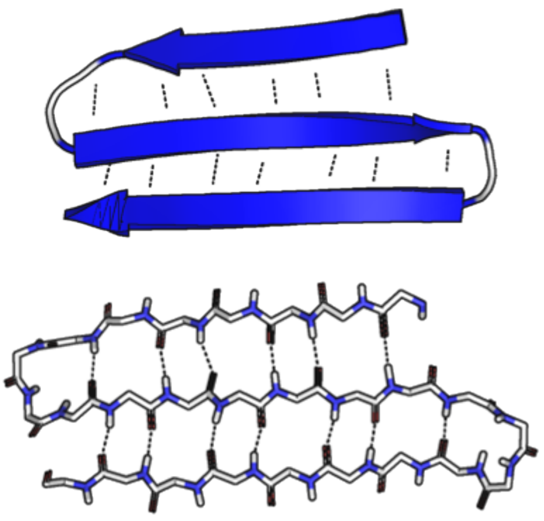


Figure 4.1: Condensation reaction between two amino acids to yield a peptide bond. The amine groups are coloured **blue**, the carboxylic groups red and the resultant peptide bond **purple**.

Each amino acid is defined by the contents of its sidegroup. Of the possible amino acids, 22 are considered ‘natural’ or *proteinogenic*, such as arginine, lysine or glycine. This select group of amino acids make up the polypeptides that form the vast majority of proteins. The exact sequence of amino acids that occur with a given chain describes the **primary structure** of the protein. Due to interactions between the various amino acids in the sequence, such as hydrogen bonding, electrostatic attraction and disulfide bridges, the polymer chain folds itself into a unique three-dimensional structure that determines the role of the protein. The large number of combinations of proteinogenic amino acids allows proteins to cover a multitude of different forms and fulfil many roles.

The overall three-dimensional structure that a protein folds into is known as the *tertiary structure* of the protein. However, throughout the numerous proteins whose structures have been identified, certain substructures or **motifs** have been found to occur repeatedly. These motifs were predicted a full decade before the first structures of proteins were determined by Pauling, Corey and Branson (for which Pauling later won the 1954 Nobel Prize in Chemistry).[159–161] By looking at the properties of the individual amino acids, they were able to predict the existence and stability of polypeptide helices, formed by hydrogen bonds between the carbonyls of the amino acids and the amines of amino acids further along the chain. This stabilisation causes the chain to adopt a regular helical shape. They proposed the existence of the **α-helix**,[161] where each amino acid is hydrogen-bonded to the residue which lies four residues further along the chain. Figure 4.2b illustrates the general appearance of an α-helix. Other helices are also possible, including the **π-helix** (with each residue bonded to one five further along) and the **3_{10}-helix** (with each residue bonded to one three further along), however, the α-helix is the most common. For the purpose of visualisation, we shall refer to all three collectively as helices.

The second motif predicted by Pauling and Corey[160] was the **β-sheet**. Here, different parts of the chain run parallel or anti-parallel to one another, with hydrogen bonds forming between the strands similar to rungs of a ladder. Figure 4.2a illustrates a β-sheet consisting of three anti-parallel strands, with the hydrogen bonds forming rungs between each strand. In reality, β-sheets tend to curve gently rather than lie completely flat. This leads to larger-scale motifs such as beta barrels, where many β-strands form the surface of a larger-scale cylinder.


(a) β-sheet.

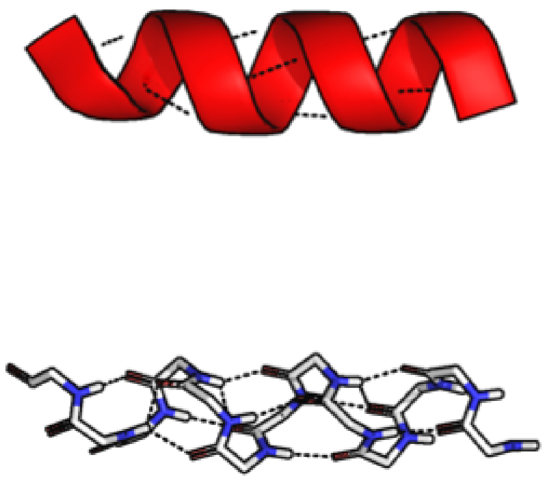

(b) α-helix.

Figure 4.2: Depiction of the two common secondary structure motifs, both as their protein backbone and as their common shorthand as arrows and coiled ribbons respectively. Image is © 2017 Thomas Shafee under the Creative Commons Attribution-Share Alike 4.0 International License: `http://creativecommons.org/licenses/by-sa/4.0`

Together, these motifs of helices and sheets are referred to as **secondary structure**. They occur in part because of inherent rigidity along the protein backbone enforced by the peptide bonds. Due to steric clashes between the side groups, the four constituent atoms of the peptide bonds are constrained to a plane (termed the peptide plane), with the alpha carbons forming the diagonal vertices of the connected sheets. There is hence only two angles per amino acid that are generally needed to describe the conformation of the protein backbone — the ϕ dihedral angle (C-N-Cα-C) and the ψ dihedral angle (N-Cα-C-N). Figure 4.3 illustrates how these peptide planes connect adjacent amino acids, spanning the peptide bonds between them.

The orientation of the peptide bonds about an alpha carbon can therefore be described in a two-dimensional Ramachandran plot.[162,163] On these plots, only certain areas are ener-

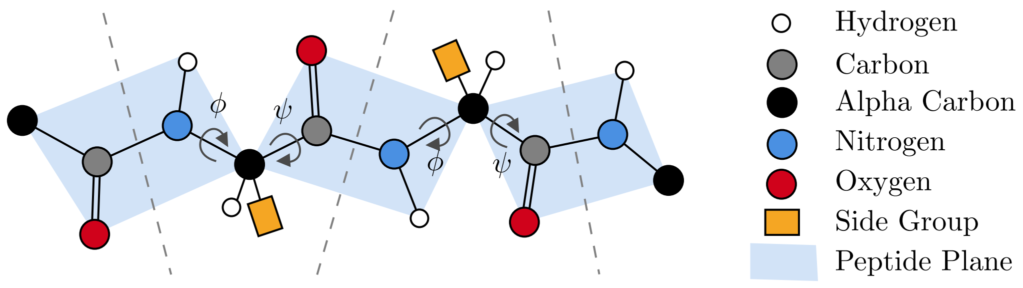


Figure 4.3: Illustration of three peptide planes. The six atoms of each peptide bond are constrained to be coplanar, leaving the conformation of the backbone to be dictated by the two dihedral angles describing how subsequent planes are rotated relative to each other. The alpha carbons form the joining vertices between adjacent peptide planes. The dashed lines divide the backbone into its individual amino acids, illustrating that each amino acid has a ϕ and ψ dihedral angle associated with it.

getically favourable due to steric constraints. Whilst these areas can be broadly partitioned into secondary structure motifs of β-sheets and α-helices, there is some overlap and ambiguity. We therefore require some better-defined approach to deciding if and where these motifs are present.

Determining Secondary Structure

The assignment of each residue to a certain motif is inherently arbitrary, as they are defined by criteria such as hydrogen bonding or geometric proximity in which there is no exact cutoff. Discovered protein structures were being stored in the Protein Data Bank,[164] with the secondary structure being determined by a crystallographer. To ensure consistent assignment, algorithms have been developed that assign secondary structures consistently based upon atomic coordinates.

Early approaches utilised the dihedral angles that appear on Ramachandran plots. However, this is at odds with the hydrogen-bond approach most often used in the manual assignment by crystallographers and used to define the motifs themselves..[165] Instead, approaches which approximate the direction and strength of the hydrogen bonds are commonly used. The first such complete algorithm was proposed by Levitt and Greer[165]. Their approach used only the atomic coordinates of the alpha carbons as input, as they were more available and easier to determine. To calculate the hydrogen bonds, they predicted the positions of the carbonyl and amine atoms based upon the positions of the alpha carbons.

The *de facto* standard method is the **DSSP** (Define Secondary Structure of Proteins) algorithm.[166] This uses the atomic positions of not just the backbone carbons, but the nitrogen of the amine and oxygen of the carbonyl. The algorithm starts by first determining the lowest energy hydrogen bond for each carbonyl and amine of the residues. The energy of a hydrogen bond between a carbonyl C=O and an amine N−H in DSSP is given by:

$$E = q_1 q_2 f\left(\frac{1}{r_{ON}} + \frac{1}{r_{CH}} - \frac{1}{r_{OH}} - \frac{1}{r_{CN}}\right)$$

where $q_1 = 0.42e$, $q_2 = 0.20e$, f is a dimensionless factor with value 332, and r is the distance between the pair of atoms measured in angstroms. This gives an energy E in $\mathrm{kJ\,mol^{-1}}$. A hydrogen bond is deemed to exist if this energy $E < -0.5\,\mathrm{kJ\,mol^{-1}}$.

Now that the hydrogen bonds have been assigned for each residue, small structures such as turns (where a residue is linked to another a couple of indices further up the chain) and bridges (where two strands are linked by two adjacent hydrogen bonds) are determined. Helices and sheets are then predicted based upon sequences of adjacent turns and bridges.

Other algorithms exist to determine the secondary structure, such as STRIDE[167] (STRuctural IDEntification), which has a more complex hydrogen bond definition but predicts longer structural motifs that may be split by DSSP due to minor deviations. Some other methods utilise alpha carbon distances and dihedral angles, such as DEFINE[168] and KAKSI[169]. Generally, the disagreement between the various algorithms for determining secondary structure lies in their classification of residues which lie towards the ends of the secondary structure motifs, whilst agreeing on residues clearly in the middle of helices and sheets.[169]

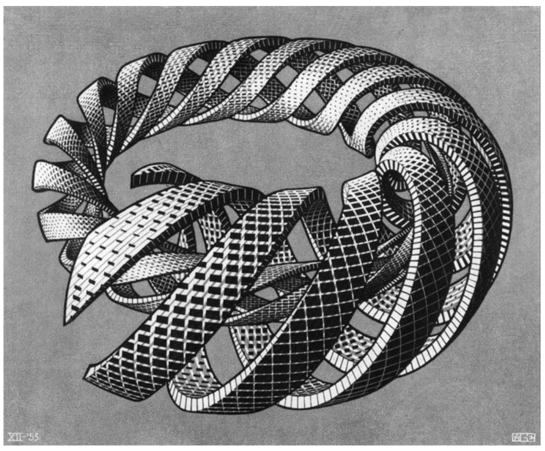
In Narupa iMD, I implemented the DSSP algorithm to calculate the secondary structure assignments for each residue. The advantage of DSSP is that it uses the atomic positions of atoms other than the alpha carbons, such as the amino hydrogen and carbonyl oxygen. In an atomistic simulation of a protein, these positions are available, whereas in a crystallographic study they may be difficult to determine. As the calculation of the assignments is expensive and involves an iteration along the entire length of each polypeptide chain, the assignments are recalculated at second-long intervals. The blending of subsequent assignments to avoid sudden changes is discussed later in this chapter.

4.2 History of the ribbon diagram

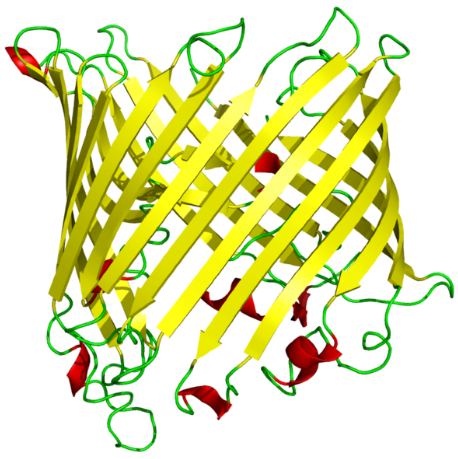
Generally, proteins consist of many thousands of atoms. Fully atomistic representations such as ball-and-stick can be visually overwhelming and difficult to interpret for this number of atoms. As discussed in Chapter 3, another family of representations are surface-based approaches. These describe the general shape of the tertiary structure but obscure the internal structure of the protein. Between these two extremes lie representations that elucidate the structure of the protein backbone — **ribbon diagrams** (also known as cartoon diagrams). This representation traces the path of the protein backbone, whilst using a ribbon surface to highlight the presence and orientation of secondary structure motifs. They are ubiquitous when depicting proteins, and hence are an important feature in any molecular visualisation software.

The origin of ribbon diagrams lies in earlier attempts to distil the complexity of a full ball-and-stick representation of a protein into something where the backbone could be clearly seen. Ribbon diagrams were developed by Jane Richardson and were inspired by drawings by M.C. Escher,[170] whose series of lithographs[171–174] involving ribbons illustrate that a series of parallel and aligned ribbons can imply the existence of a larger surface, whilst giving negative space through which depth can be perceived. These ribbons were used when drawing the two motifs common to proteins — β-sheets and helices.

Early ribbon diagrams were hand-drawn, being traced over printouts of the alpha carbon traces determined by x-ray crystallography.[112] Through these diagrams, Richardson developed



(a) M.C. Escher. *Spirals.* 1953



(b) Beta barrel structure.

Figure 4.4: Comparison between an artwork by M.C. Escher and the structure of a Beta barrel visualised using a ribbon structure. Both use the negative space created by the spaces between the aligned ribbons to allow a view to the other side of the structure, whilst still implying the full surface. *Spirals* is © M.C. Escher and used under fair use. The beta barrel image is © 2007 Opabinia Regalis under the Creative Commons Attribution-Share Alike 3.0 Unported license: `https://creativecommons.org/licenses/by-sa/3.0`.

many of the features that would determine how these diagrams appear to this day. She used arrowheads to indicate the direction of β-sheets, helping to distinguish between parallel and anti-parallel sheets. By only drawing the β-sheets and helices using these wide ribbon representations, whilst leaving the rest of the backbone as a thin tube representation, the secondary structure was emphasised over the tertiary structure. Figure 4.5 illustrates an example of her hand-drawn ribbon style. It illustrates features which we still attempt to capture today, such as regular helices and smooth curving β-sheets.

One of the first computer programs to generate these diagrams was developed by Lesk and Hardman[176], though helices were rendered as solid cylinders. It was Carson and Bugg[177–181] who developed a detailed method for generating ribbon diagrams. Their ribbons did not pass through the alpha carbons, but instead used guide points positioned midway between adjacent alpha carbons. The choice of using guide points between the alpha carbons is partly to address the natural 'wave' shape of a β-sheet.[182] Parallel to the work by Carson and Bugg, Priestle[183] developed an alternative method for smoothing out the alpha carbon positions to give a smooth curve. This technique was used in the MOLSCRIPT program,[107] though for helices a standard Hermite spline was used.

The flat faces of the ribbon are aligned to illustrate the directionality of the hydrogen bonds

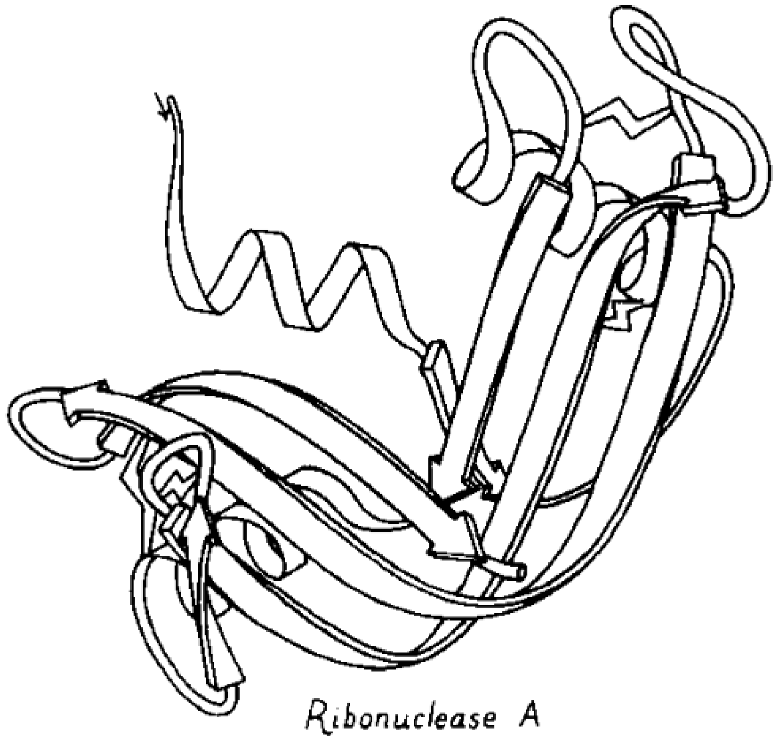


Figure 4.5: Illustration of Ribonuclease A by Jane Richardson, from *The Anatomy and Taxonomy of Protein Structure*.[175]

that link parallel or antiparallel β-sheets. A common issue encountered when rendering ribbons is that this alignment may flip up to 180 degrees between two adjacent residues.[107,177,184,185] This is often addressed by flipping the normal (to which the flat face of the ribbon will be aligned) if the angle between two adjacent normals is greater than 90 degrees.[107,177,185] This quirk and how it has been addressed is the raison d'être for my double-normal interpolation algorithm, and will be discussed at length later in this chapter.

Ribbon diagrams are present in practically all modern molecular visualisation software, second only to ball-and-stick in their ubiquity. With the increase in computational power, especially the power of modern GPUs and consumer graphics, ribbon diagrams of large biomolecules can be generated with ease. Generally, the mesh of the ribbon diagram is generated as a whole on the CPU. This is ideal for static structures, but can lead to performance issues when visualising dynamic trajectories. Zamborsky, Szabo, and Kozlikova[186] addresses this by creating small mesh sections to represent a section of a ribbon and deforming them dynamically on the GPU. This approach was also taken by Hermosilla et al.[185], who explicitly called out modern software such as PyMol or VMD as generating the 3D geometry of the entire ribbon on the CPU. These approaches are similar to that used for other large-scale rendering as discussed in Chapter 3, using the programmable nature of the GPU to avoid generating the entire mesh explicitly.

Ribbon diagrams are not unique to proteins. The same approaches used for polypeptides may be applied to other biologically important polymers such as polysaccharides[38,187] and nucleic acids.[188]

Unlike the original hand drawn ribbon diagrams, we must define exactly how to draw a curve that represents the protein backbone. This is usually achieved by breaking the curve down into small segments, each with a simple polynomial form. The next section will go into detail about how the path of the protein backbone is converted into a smooth curve.

4.3 Splines and Curves

The simplest way in which to display the backbone of a protein is to use a liquorice-style representation, with consecutive alpha carbons connected by straight segments. However, the sharp and angular nature of this representation is often considered at odds with the mental image of a smooth and flexible peptide chain. Hence, the underlying backbone structure of proteins is normally represented by a smooth curve. A curve may be specified **parametrically**, in which it is taken to be a vector-valued function $\vec{p}(t)$ of a scalar t. As t is varied smoothly between two values, the curve is traced out by the function $\vec{p}(t)$.

Commonly, a curve may be broken up into separate shorter **segments**, usually defined as polynomials in t. A curve created by the piecewise assembly of polynomial segments is called a **spline**. Each segment's parameter may be scaled such that it is defined over the range $[0, 1]$, and hence a general spline of N segments may be expressed as:

$$\vec{S}(x) = \begin{cases} \vec{s}_0(x) & 0 \le x < 1 \\ \vec{s}_1(x-1) & 1 \le x < 2 \\ \vdots & \\ \vec{s}_{N-2}(x-(N-2)) & N-2 \le x < N-1 \end{cases}$$

Figure 4.6 illustrates a spline with each segment defined over the range $[0, 1]$. As each segment itself is a polynomial and hence smoothly varying, how these curves connect at their endpoints defines the **continuity** of the curve as a whole. Generally, the minimum criteria we enforce is that the endpoints of each segment are connected, and hence the curve possesses C^0 continuity. If the tangents of each pair of subsequent segments are equal, there are no discontinuities in gradients and hence the curve possesses C^1 continuity. Finally, if the curvature is smooth between segments, the spline possesses C^2 continuity. Figure 4.7 illustrates C^0 and C^1 continuity for a spline consisting of four segments. Higher order continuity is often difficult to see visually.

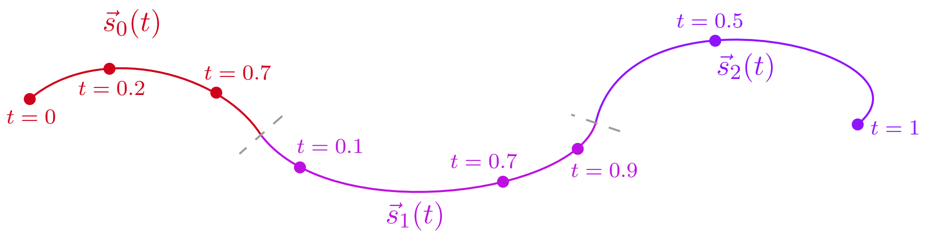


Figure 4.6: Division of a curve into spline segments $\vec{s}_0(t)$, $\vec{s}_1(t)$ and $\vec{s}_2(t)$. As t is increased from 0 to 1, Each segment $\vec{s}_i(t)$ traces out a smooth curve.

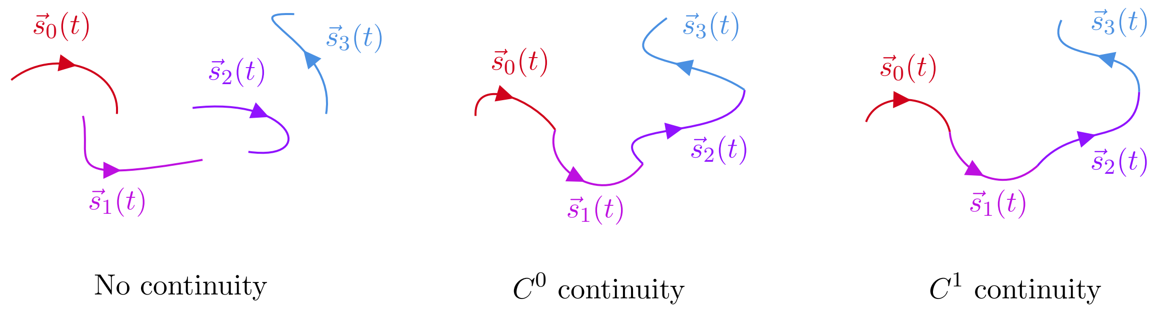


Figure 4.7: Continuity of splines consisting of piecewise polynomial segments $\vec{s}_i(t)$.

Splines find widespread application in interpolating or approximating data points. They are usually defined by a set of points, called **control points**. Two general categories exist for splines based upon a set of points — interpolating and approximating. Interpolating splines pass through each point (with each segment connecting two consecutive points), whilst approximating splines are influenced by, but do not necessarily pass through, the points. The choice of interpolating or approximating spline depends on the application and desired properties.

Generally, a low-order polynomial is chosen for each segment, as higher order polynomials require more operations to determine each point. The order of the polynomial also relates to how many parameters we have to define our curve by. For example, each linear segment requires two parameters, whilst a cubic segment requires four. In a simple case, connecting each pair of consecutive points with a straight line requires two parameters per segment. These parameters are the initial and final point, with the linear spline segment passing between them as shown in Figure 4.9.

Cubic Hermite Spline

If we wish to define a C^1 continuity spline, then each segment requires knowledge of not just its endpoints but also the tangents at its endpoints. Therefore, a common curve for interpolating points is a cubic spline. Appendix B illustrates that if each segment must be a cubic polynomial that connects $\vec{p}_i$ to $\vec{p}_{i+1}$, with associated tangents $\vec{m}_i$ and $\vec{m}_{i+1}$, it must take the form:

$$\vec{s}_i(t) = (2t^3 - 3t^2 + 1)\vec{p}_i + (t^3 - 2t^2 + t)\vec{m}_i + (-2t^3 + 3t^2)\vec{p}_{i+1} + (t^3 - t^2)\vec{m}_{i+1} \tag{4.1}$$

A spline of this nature is depicted in Figure 4.8. The four polynomial coefficients of this cubic are known as the **Hermite basis functions**, with the spline itself being a **cubic Hermite spline**. This method of smoothly connecting a set of points with associated tangents can be contrasted with simple linear interpolation as shown in Figure 4.9:

$$\vec{s}_i(t) = (1 - t)\vec{p}_i + t\vec{p}_{i+1} \tag{4.2}$$

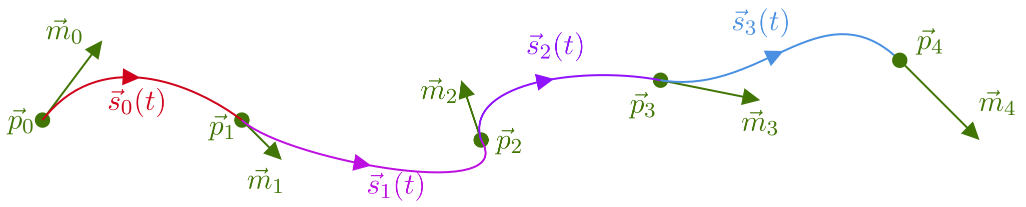


Figure 4.8: Cubic Hermite curve for a set of points $\{\vec{p}_i\}$ and their corresponding tangents $\{\vec{m}_i\}$

The cubic Hermite spline is the general form for any C^1 interpolating cubic spline. The set of tangents $\{\vec{m}_i\}$ are parameters — we have not yet defined how to calculate these. The common choice of tangents is to define them relative to the positions of nearby points. A **Cardinal spline** takes the gradient of the i-th point to be proportional to the vector between the preceding and subsequent point:

$$\vec{m}_i = (1 - c)(\vec{p}_{i+1} - \vec{p}_{i-1}) \tag{4.3}$$

The size of these tangents is determined by a parameter c. When $c = 1$, this yields tangents of zero length, and hence devolves to the linear interpolation spline as laid out in Equation 4.2 and Figure 4.9. When $c = 0.5$, this curve is referred to as a **Catmull-Rom spline**[189], with gradients:

$$\vec{m}_i = \frac{1}{2}(\vec{p}_{i+1} - \vec{p}_{i-1}) \tag{4.4}$$

More complex forms such as the Kochanek-Bartels[190] spline have parameters such as tension and bias. However, they still assume that the gradient at $\vec{m}_i$ is purely a function of $\vec{p}_{i-1}$, $\vec{p}_i$ and $\vec{p}_{i+1}$. Catmull-Rom splines are the archetypal example of an interpolating spline — the curve by definition passes through all the control points $\vec{p}_i$.

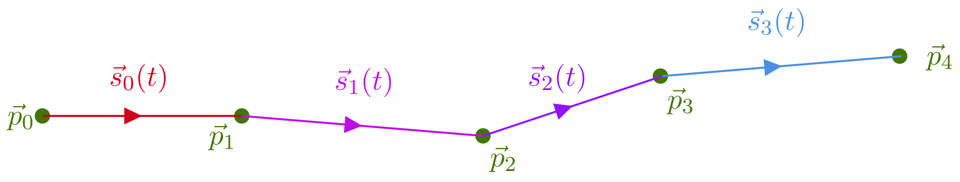


Figure 4.9: Simple linear interpolating spline between a set of points $\{\vec{p}_i\}$, as described in Equation 4.2

Natural Cubic Splines

Instead of determining the tangents by considering nearby points, they may be determined by applying further constraints on the curve. The general form for a cubic Hermite curve was obtained by enforcing C^0 and C^1 continuity. It may be additionally enforced that C^2 continuity is also required. Appendix B shows that if we additionally enforce C^2 continuity for a spline over N points, we obtain $N-2$ equations that constrain the values of the tangent vectors $\{\vec{m}_i\}$. Therefore, we require two additional conditions to uniquely define a cubic spline that has C^2 continuity. One solution is to enforce **natural boundary conditions**, which state that the curvature at each end of the full spline must equal $\vec{0}$. With these additional conditions, a unique C^2 spline for a set of points can be determined, known as a **natural cubic curve**.

In a natural cubic curve, the tangents and positions are related by a matrix equation which must be solved to calculate the tangents. Therefore, each tangent $\vec{m}_i$ is influenced by the positions of *all* the points $\{\vec{p}_i\}$. This is referred to as global support, where each tangent is a function of all points within the curve. In contrast, the Catmull-Rom spline has local support, as each tangent is a function of the preceding point, the current point and the subsequent point only. This is important if single data points are manipulated — a spline with global support must be completely recalculated, whilst a spline with local support need only recalculate segments near the altered point. Generally, for a ribbon diagram all the atoms are moving at once, and hence global support is not much of an issue.

The derivation of natural cubic splines may be found in Appendix B.

B-Splines

The above examples are interpolating splines — given a set of points, the splines pass through every one. The other broad category of splines is that of approximating splines. The points are here referred to as control points, as they influence the shape of the curve whilst not necessarily lying on it.

A classic example of a non-interpolating spline is a **B-spline**. The B of B-spline stands for basis, as they can be considered to form a basis for all other splines. The exact definition of B-splines can be found in Appendix B. In the cubic case, the uniform cubic B-splines are C^2 continuous, but do not pass through the control points. The uniform cubic B-spline may be written as:

$$\frac{1-3t+3t^2-t^3}{6}\vec{p}_{i-1}+\frac{4-6t^2+3t^3}{6}\vec{p}_i+\frac{1+3t+3t^2-3t^3}{6}\vec{p}_{i+1}+\frac{t^3}{6}\vec{p}_{i+2} \tag{4.5}$$

Figure 4.10 illustrates a B-spline for a set of points. Given a set of N control points $\{\vec{p}_i\}$, a uniform cubic B-spline does not pass through the points but instead consists of segments connecting a new set of N points $\{\vec{q}_i\}$. Therefore, we could invert the problem and instead calculate some new control points such that, if a B-spline is defined using these points, the resultant B-spline actually passes through our original control points. This approach results in a curve that is functionally identical to the natural cubic spline, and requires solving a similar matrix problem. This is expected, as for C^2 continuous splines that pass through a set of points, there are only two free parameters (defining how the endpoints behave). B-splines and natural cubic curves are therefore closely related, but B-splines have local support and are approximating splines, whilst natural cubic curves have global support and are interpolating splines.

B-splines also form the basis of non-uniform rational B-splines, also known as NURBS. These are B-splines where each point also has a corresponding weight, and find wide applications in defining smooth surfaces in 3D modelling.[191,192]

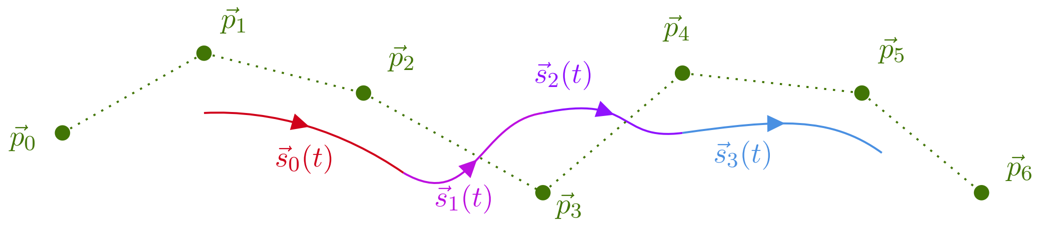


Figure 4.10: Example of a uniform cubic B-spline with control points $\{\vec{p}_i\}$. The spline is still made of separate cubic sections based on four consecutive points, but it does not pass through any of them.

Splines in Molecular Visualisation

As the origins of ribbon diagrams lie in hand-drawn curves with no exact mathematical definition, it is unsurprising that in current molecular visualisation several different spline formalisms are used. Table 4.1 lays out the variety of spline approaches used, with about half choosing interpolating splines and half using approximating splines.

The early work by Carson and Bugg[177–180] utilised B-splines passing through the midpoints of the peptide sheets (taken to be the midpoints between alpha carbons). B-splines were also used for DNA helices in UCSF Chimera.[193]

The technique of solving a B-spline to find control points to make it an interpolating spline can be found in software such as DRAWNA program for rendering nucleic acids.[194] and BioBrowser[195]. As mentioned above, this is functionally identical to using natural cubic splines, which are utilised in ChimeraX.[109]

Software such as SETOR[196] and VMD[64] provide a choice between an interpolating (Cardinal) and approximating (B-spline) spline when rendering ribbons.

The alternative method proposed by Priestle[183] involves multiple interpolation passes in which the alpha carbon positions are progressively smoothed out. This approach was later adapted by MolScript,[107] though generally this technique has not been widely adopted in comparison to cubic spline approaches.

An issue with approximating curves is raised when side chains are visualised in addition to the protein backbone. Here, the chains may appear to be disconnected if the curve does not pass through the alpha carbon. Two common ways of addressing this are to stretch the bond from the side chain to the curve, or to draw an additional ‘tether’ connecting the alpha carbon to the curve. This second approach is adopted by ChimeraX.[109]

PyMol uses a unique approach for calculating the spline used for its ribbon diagrams, with

Software	Spline	Control Points	Passes through Cα?
Narupa iMD	Catmull-Rom	Cα	Yes
VMD 1.9.4	Catmull-Rom	Cα	Yes
	B-Spline	Cα	No
Avogadro 1.95.0	B-Spline	Cα	No
Mol* 3.0.2	Catmull-Rom	Cα	Yes
NGL Viewer 2.0.0	Cardinal ($c = 0$)	Cα	Yes
ChimeraX 1.3	Natural Cubic Spline	Cα	Yes
PyMol	Custom Implementation	Cα	No
JSMol 14.31.23	Cardinal ($c = 1/8$)	Cα	Yes
UnityMol 1.1.3	B-Spline	Cα	No

Table 4.1: Summary of spline fits used in various molecular visualisation packages, highlighting the variety of methods used.

each segment given by:

$$\vec{s}_i(t) = (1-t)\vec{p}_i + t\vec{p}_{i+1} + \tau s(t)s(1-t)\|\vec{p}_{i+1} - \vec{p}_i\|(s(1-t)\vec{m}_i - s(t)\vec{m}_{i+1})$$

where τ is a user-controlled factor, $s(t)$ is a smoothing function:

$$s(t) = \begin{cases} \frac{1}{\sqrt{2}}\sqrt{t} & t < \frac{1}{2} \\ 1 - \frac{1}{\sqrt{2}}\sqrt{1-t} & t \geq \frac{1}{2} \end{cases}$$

and the tangents are given by:

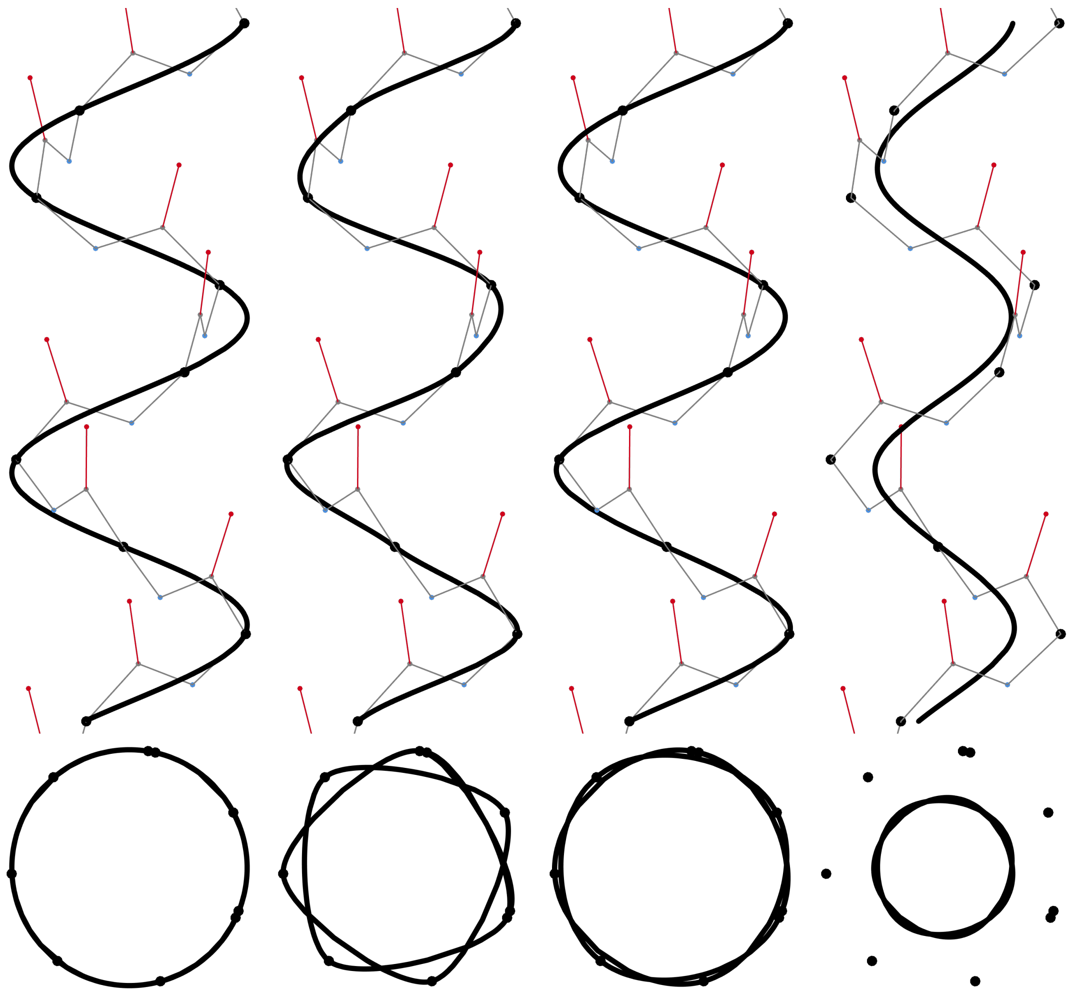
$$\vec{m}_i = \frac{1}{\left\| \frac{\vec{p}_{i+1}-\vec{p}_i}{\|\vec{p}_{i+1}-\vec{p}_i\|} + \frac{\vec{p}_i-\vec{p}_{i-1}}{\|\vec{p}_i-\vec{p}_{i-1}\|} \right\|} \left(\frac{\vec{p}_{i+1}-\vec{p}_i}{\|\vec{p}_{i+1}-\vec{p}_i\|} + \frac{\vec{p}_i-\vec{p}_{i-1}}{\|\vec{p}_i-\vec{p}_{i-1}\|} \right)$$

Due to the presence of square roots, the PyMol implementation is the only implementation examined that does not utilise cubic splines. The choices here have clearly been made to yield a visually pleasing ribbon, but it is difficult to compare to other algorithms and is poorly documented.

The dichotomy between interpolating and approximating splines is evident when the two main secondary structure motifs are considered — β-sheets and helices. When visualising helices, using approximating curves such as B-splines can lead to them appearing artificially thinner than they should.[177] However, the opposite occurs when considering β-sheets. Though considered to be approximately flat, when considered purely in terms of the positions of the alpha carbons, a β-sheet appears to zig-zag along its length. This is where using approximating curves such as B-splines or using the midpoints between alpha carbons (which would approximate the centre of the peptide bonds) would lead to a more aesthetically pleasing result.

Figure 4.11 compares the three common approaches of ribbon fitting for a standard alpha helix. The Catmull-Rom approach fails to capture the symmetry of the helix, giving a mis-shapen appearance with sharper corners. It is important to be able to identify areas where a helix is kinked or deviates from the idealised structure,[188] which the Catmull-Rom spline complicates by deviating from an exact helix. The natural cubic approach fairs much better, forming a near perfect circle when viewed from above. This is in part due to its enforcement of C^2 continuity between its cubic segments. A perfect helix has a constant non-zero curvature and torsion, and hence a cubic curve with higher continuity requirements is a better fit. Finally, the B-spline approach also captures the correct shape of the helix, however as it is not an interpolating spline, the overall radius of the helix is underestimated. This issue has been observed since Carson and Bugg[177], who suggested a 1.5 Å translation to avoid slender helices.

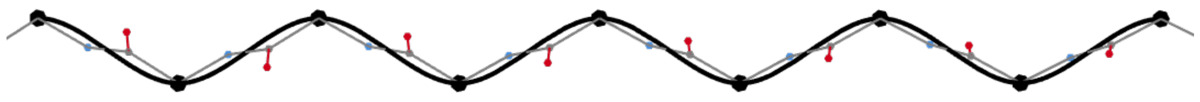
Whilst in helices it is desirable to capture the shape and size of the helix, defined by the positions of the α-carbons, in a β-sheet the opposite is the case. Naturally, the α-carbons of a β-sheet zig-zag, when viewed in the direction of their hydrogen bonds. When using an interpolating spline, this yields ribbons which adopt a wavy appearance. For β-sheets, it is

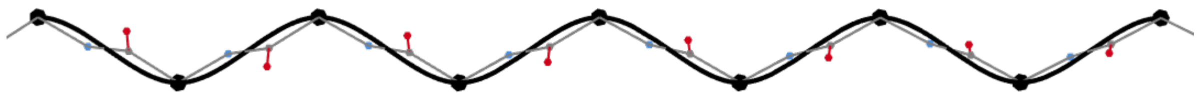


(a) Pure Helix (b) Catmull-Rom (c) Natural Cubic (d) B-Spline

Figure 4.11: Comparison of the three common curve approaches fitted to an ideal α-helix, as seen from the side and from above. 4.11a illustrates a pure helical curve, which appears sinusoidal from the side and as a circle from above. The stick diagrams indicate the backbone atoms of the peptide chain, namely the alpha carbon, amine nitrogen and carbonyl carbon and oxygen. The α-helix was generated using the PeptideBuilder 1.1.0 Python package,[197] using dihedral angles of $\phi = -60°$ and $\psi = -45°$.

the overall direction and orientation that are important factors. Therefore, most software such as ChimeraX and JSMol will apply some smoothing modification to the positions used for β-sheets, whilst using the raw alpha carbon positions for helices and other structures.


(a) Catmull-Rom


(b) Natural Cubic

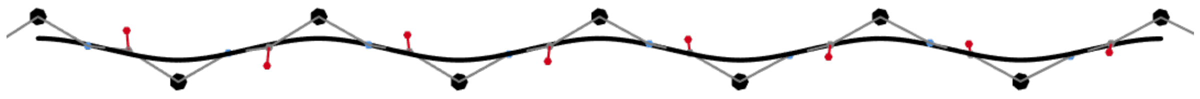

(c) B-Spline

Figure 4.12: Side-view of different splines fitted to a β-sheet. The β-sheet was generated using the PeptideBuilder 1.1.0 Python package,[197] using dihedral angles of $\phi = -135°$ and $\psi = 135°$.

4.4 Extruding the curve

Until this point, we have defined a **space curve** — a one-dimensional parameterised path through space. We can extrude this path to give a constant-thickness cylinder, creating a tube representation that illustrated the path of the protein backbone. Figure 4.13 illustrates this representation in Narupa iMD.

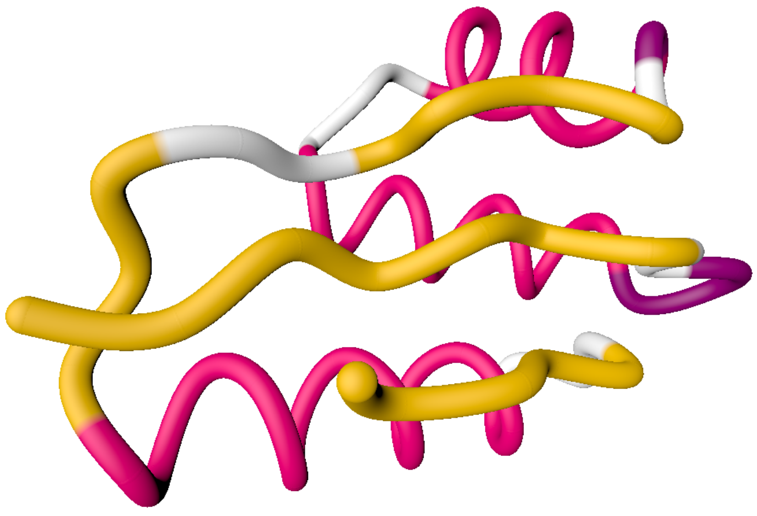


Figure 4.13: Tube representation of a small protein (Code 1CTF) along a natural cubic spline, and coloured according to the secondary structure motifs.

However, a core part of the diagrams developed by Richardson is the ribbon part used to illustrate the secondary structure motifs, both to show their orientation and differentiate them from more general turns and curves. Here, we use ribbon in the sense of a two-dimensional sheet which follows the curve. In the field of differential geometry, a **ribbon** is defined as the combination of a smooth curve with a unit normal $\vec{n}$ assigned at every point along the curve. Therefore, defining a ribbon requires assigning some normal vector $\vec{n}$ to every point along our curve.

Together with the tangent vector of the curve (which is simply given by the derivative of the curve's parameterised equation $\vec{p}(t)$), the normal and tangent form a pair of orthogonal vectors. We may also define a third axis at each point, called the **binormal** $\vec{b}$, which lies perpendicular to both tangent and normal. Hence, at every point along the curve we now have an orthogonal set of basis vectors (the tangent, normal and binormal), which form a set of coordinate axes. This assignment of a coordinate frame to each point along a curve is referred to as **curve framing**, an example of which is given in Figure 4.14.

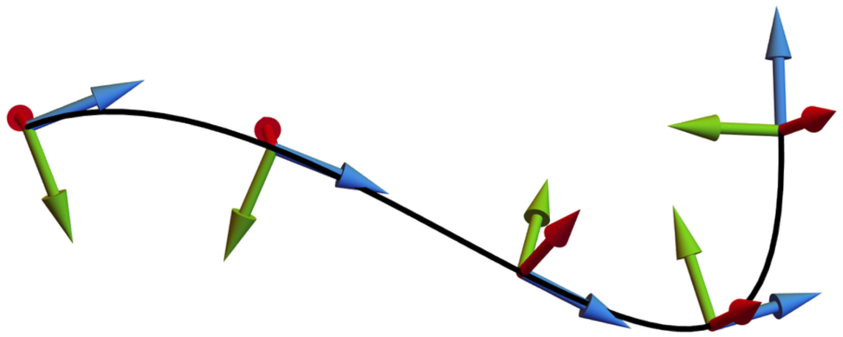


Figure 4.14: Illustration of a Frenet-Serret frame along a cubic Hermite curve. At each point along the curve (black), there is an orthonormal basis set given by the tangent (blue), normal (green) and binormal (red).

Defining Orientation

Whilst there are techniques such as Frenet-Serret which define a normal at every point along a curve purely based on its parameterised form, this neglects the meaning of the orientation of the ribbon — they highlight the orientation of the peptide planes.

To do this, we adopt a method similar to how tangents work for a Hermite spline. At each atom, we calculate some normal $\vec{n}_i$ that describes the direction the ribbon will face. Carson and Bugg[177] took the component of the Cα-O bond which is perpendicular to the Cα-Cα bond as their normal, as it lies on the peptide plane. Krone, Bidmon, and Ertl[198] suggest normals of the form:

$$\vec{n}_i = \frac{\vec{O}_i - \vec{C}_i^\alpha}{\left\|\vec{O}_i - \vec{C}_i^\alpha\right\|} \tag{4.6}$$

Due to the peptide plane's structure, the C=O bond is approximately antiparallel to the N−H bond. Therefore, using the carbonyl bond is a good approximation of the direction of the hydrogen bond and correctly aligns the ribbon with the direction of helices and sheets.[198]

Given a set of normals $\{\vec{n}_i\}$ defined at the control points, we need a method for propagating them along each spline segment. This must be a technique which reproduces the correct normals at the endpoints, but smoothly interpolates between them along the spline segment. One method for calculating normals at all points along the spline is to define a second spline. This second spline will be defined in the same manner as the main spline, but each control point will be shifted by some distance Δ along its normal. By evaluating both the main spline $\vec{p}(t)$ and this second spline $\vec{q}(t)$ for a given value t, a normal can be calculated.

This approach can be used to generate a sequence of closely spaced splines using various values of Δ (both positive and negative). This visualisation is sometimes referred to as the strands representation, and was originally described by Carson and Bugg[177]. Figure 4.15 illustrates an example produced by Carson. It was an effective technique to illustrate ribbons where hardware was better suited for visualising lines rather than three-dimensional meshes, but is rarely used in modern software.

ChimeraX assigns normals differently to helices and β-sheets. Helical segments used a geometry-based normal, based on the preceding and subsequent points. Sheets instead align the normal with the intersection line between subsequent peptide planes. To blend between the normals, parallel transport is used. This is a technique which propagates a vector along a curve, minimising the amount of twisting. However, at the end of the segment the smoothly propagated normal may not align with the desired value of $\vec{n}_{i+1}$. To address this, the normals along the segment are twisted evenly such that the final normal is aligned with $\vec{n}_{i+1}$. Generally, this method is less performant than other methods, as the parallel transport algorithm is not well suited for GPUs.[198]

VMD utilises a similar method to Carson and Bugg, but takes the normal $\vec{n}_i$ to be the average of the peptide normals of the preceding and subsequent peptide chains. It uses simple linear interpolation to blend between the initial and final normals between the start and end.

Regardless of the method used, we now have a normal $\vec{n}(t)$ that varies smoothly along each segment and across the endpoints of each segment. If all that's desired is a flat ribbon, then this is sufficient to draw the two-dimensional surface which follows the curve and is aligned with the normals. However, a three-dimensional ribbon is often preferable, and to do this we now need to extrude a cross-section along the curve.

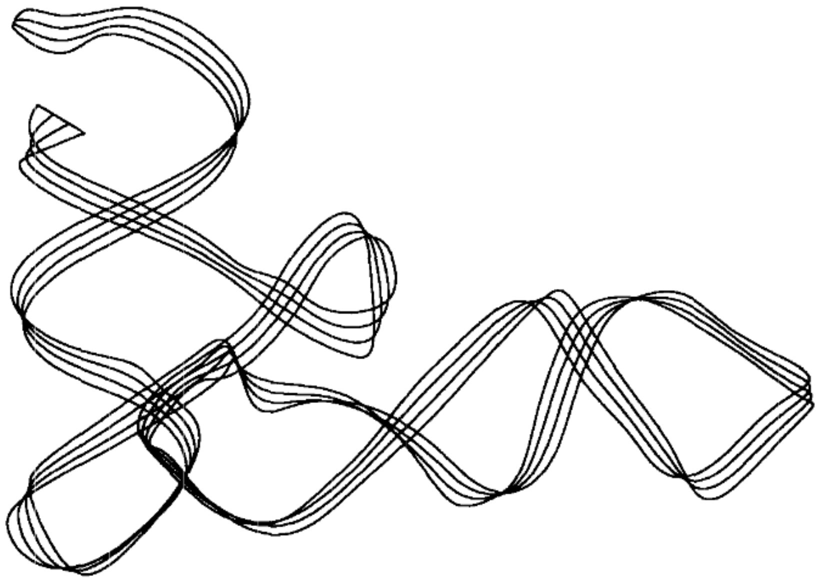
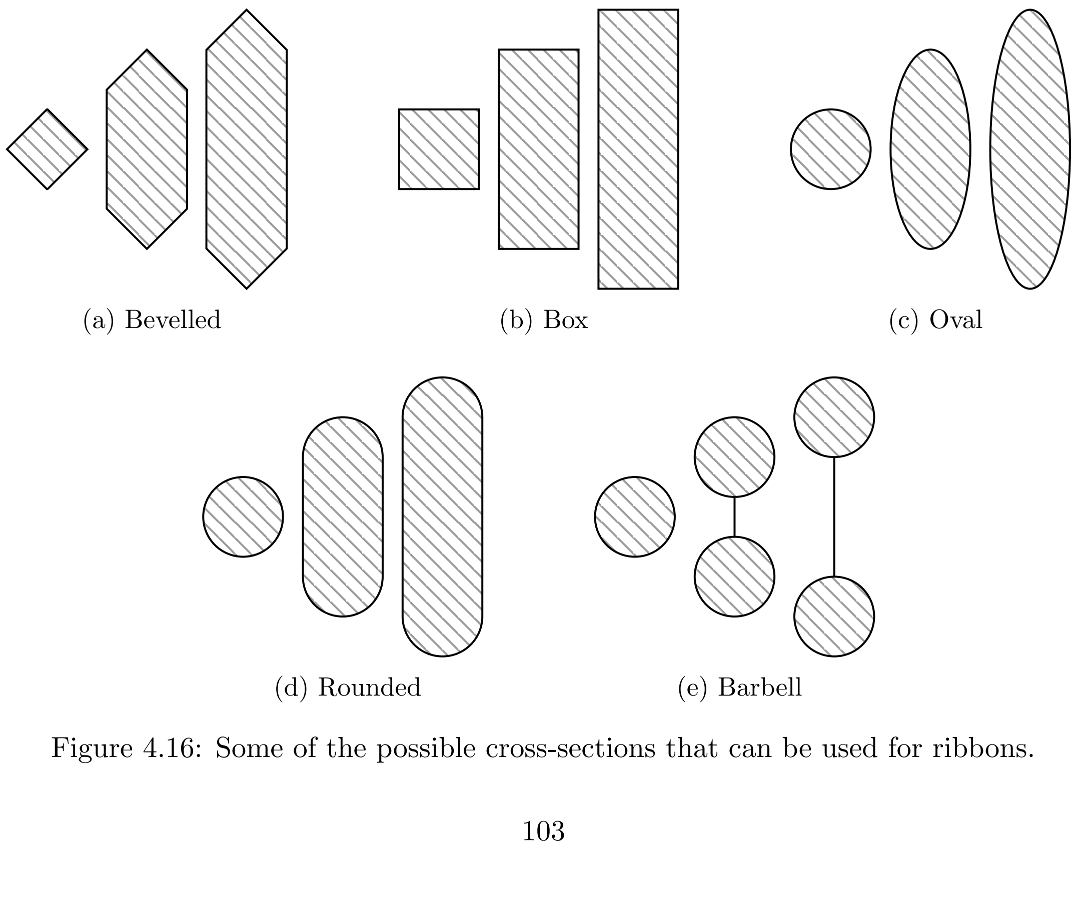


Figure 4.15: Example of a strands representation of FMET t-RNA, reproduced from Carson[178].

Cross-sections

The technique of moving a 2D cross-section along a three-dimensional curve is a common technique in modelling, and is referred to as extruding or sweeping.[199] The normal and binormal at every point along the curve define a two-dimensional plane perpendicular to the curve, and onto this surface we can align a shape to be our cross-section. There are a variety of cross-sections we can choose, with common choices being either ellipse or oval representations, or a boxy representation. The box cross-section is the one used by Richardson in her original diagrams, presumably as it is the easiest to draw by hand in the absence of shading. Another common cross-section is the barbell or dumbbell shape, consisting of two circular cross-sections joined by a flat ribbon. This representation highlights the edges of the ribbon, and can be easier to implement than an oval cross-section. Programs such as VMD, ChimeraX and PyMol all provide these three common cross-sections. Narupa iMD supplements this to two additional cross-sections — a bevelled cross-section and a rounded cross-section. Figure 4.16 illustrated how this cross-sections appear for various ribbon widths.


(a) Bevelled (b) Box (c) Oval

(d) Rounded (e) Barbell

Figure 4.16: Some of the possible cross-sections that can be used for ribbons.

4.5 Ribbon Flipping

Consider holding a belt between your hands, and slowly twisting one end of the belt around. The belt will slowly become more and more twisted, as illustrated in Figure 4.17a. As a ribbon diagram is calculated at a specific point in time, and does not account for previous timesteps or diagrams, we do not end up with an overly twisted ribbon as for the example of the belt. Instead, the direction of the twist will either be clockwise or anticlockwise, depending on which would involve twisting through a smaller angle as illustrated in Figure 4.17b. When the initial and final normals are exactly 180° apart, then there is an ambiguity.

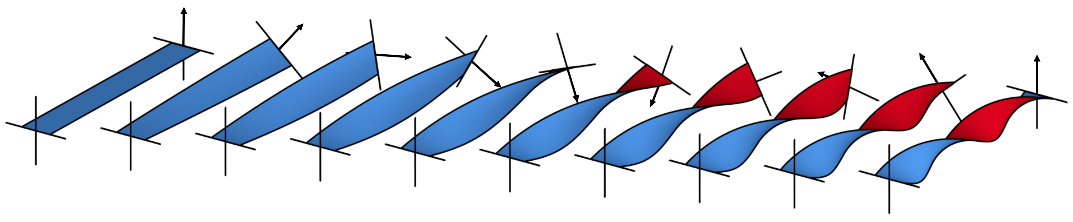
The salient feature of the normals in ribbon diagrams is that they indicate the alignment of the ribbon. Hence, flipping a normal by inverting it gives the same alignment at the endpoints, but could result in a less twisted appearance. This was first employed by Carson and Bugg[177], who noted that β-sheets commonly had normals which were antiparallel as you pass down the β-sheet. This leads to an overly twisted β-sheet, where the desired appearance is flat. They proposed flipping the final normal when $\vec{n}_i \cdot \vec{n}_{i+1} < 0$ to give a less twisted appearance as illustrated in Figure 4.17c.

Figure 4.17c shows that the maximum amount of twist possible between endpoints is now 90°. This yields static diagrams which have minimal twisting, and hence illustrate the secondary structure motifs most clearly. However, when applied to dynamic visualisation, where each ribbon diagram is a continuation of the previous one, the phenomenon of **ribbon flipping** occurs. Figure 4.18 illustrates this for consecutive frames of a protein trajectory as seen in major visualisation software. These are often seen as small flickering parts of the diagrams when viewed as a trajectory, often at the edges of secondary structure motifs.

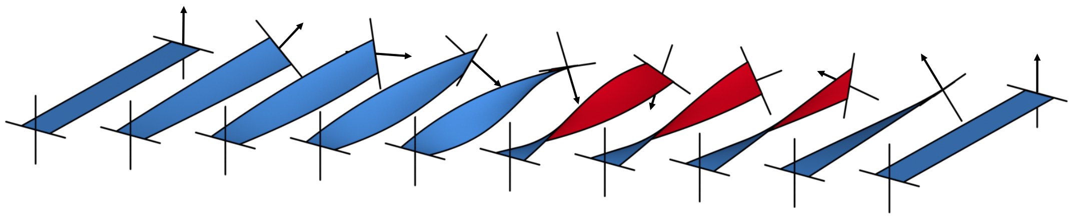
Visual continuity

The issue of ribbon flipping in Narupa iMD is exacerbated by two properties — the use of immersive virtual reality, and the relative speed of the simulation. The use of immersive environments has been previously observed to help identify graphical issues that were less obvious on a two-dimensional screen. For example, Akkiraju et al.[200] only noticed the severity and frequency of self-intersections in their visualisation of a protein molecular surface when immersed in the CAVE environment. The immersive element of virtual reality means that glitches and changes which appear nonphysical (referred to here as *visual artifacts*) have more of an effect on the user.

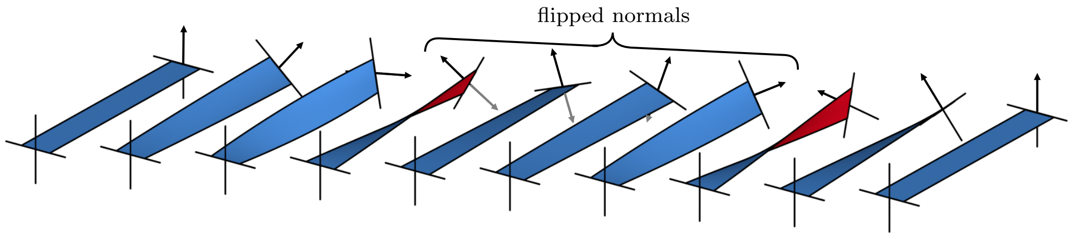
An example of this is discussed in Chapter 3, when discussing the visualisation of periodic systems. There, the sudden movement of atoms from one side of the unit cell to the other does not agree with our mental picture of how these objects should behave. As molecular visualisations are the natural progression of physical three-dimensional models, we expect them to behave in a similar physical way. Unlike Monte Carlo simulations, in molecular dynamics



(a) Rotating a ribbon by always rotating clockwise from the start normal to the end normal. This leads to a ribbon that becomes progressively more twisted, with the twist ranging from 0 degrees to 360 degrees.



(b) Rotating a ribbon either clockwise or anticlockwise, depending on which is a smaller angle. This results in a less twisted appearance, but there is a flip at 180 degrees. The twist ranges from −180 degrees to 180 degrees.



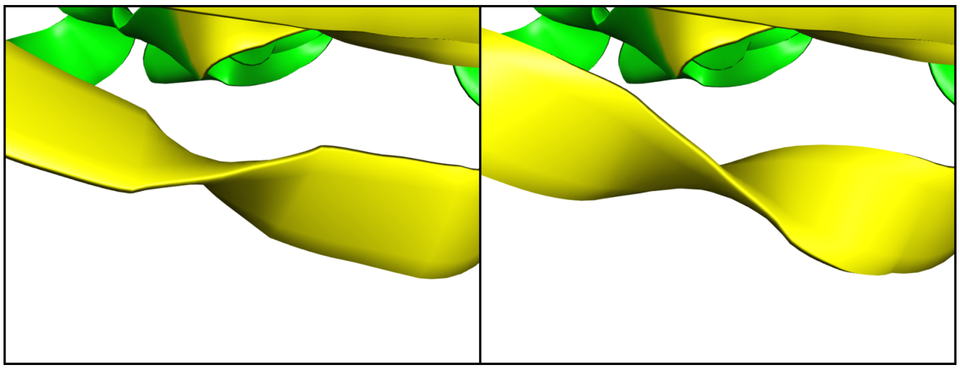
(c) As above, but flipping the final normal if it is not in the same direction as the initial normal. This limits the twist to −90 degrees to 90 degrees, but contains flips at 90 degrees and −90 degrees.

Figure 4.17: Illustration of three ways to interpolate a ribbon, given an initial and final orientation.

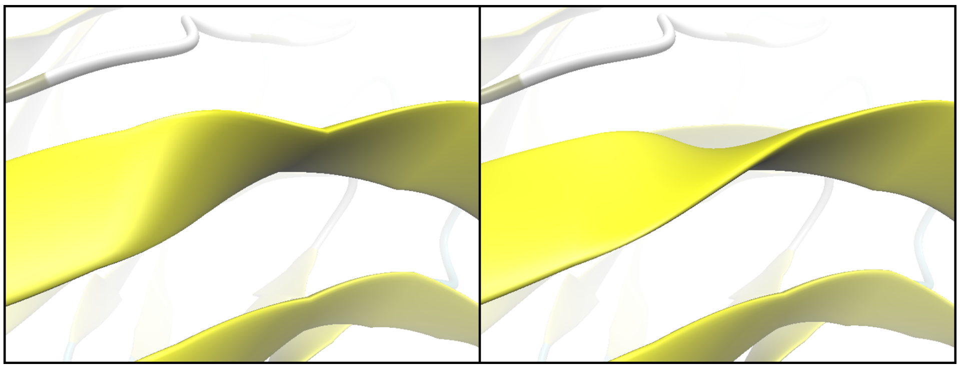
simulations the system varies in a smooth time-varying fashion, further contributing to the view that the atomic system being simulated can be compared to a physical model in real life.

In the world around us, physical objects tend to move slowly, at a timescale in which we can observe their motion. Natural phenomena which are sudden or abrupt, such as lightning strikes or explosions, are associated with unpleasant or dangerous experiences. To a lesser extent, this concept should also apply when immersed in a virtual world. This extends to concepts such as the user interface, where smoothly fading menus in and out is more desirable than sudden transitions.

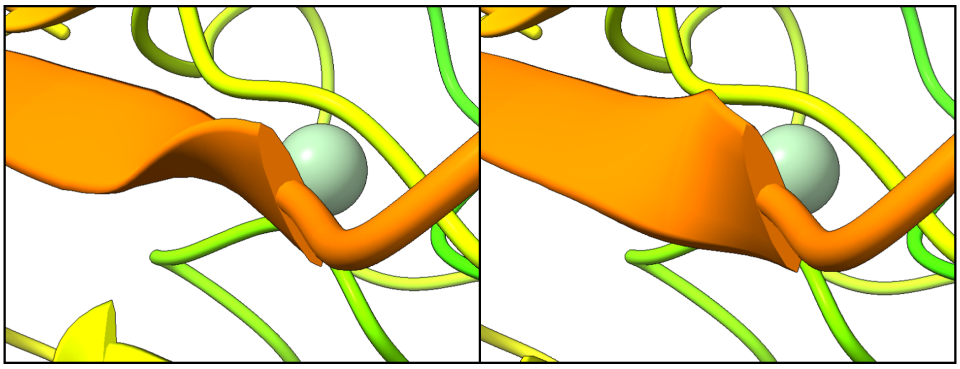
The other reason I postulate that ribbon flipping has not been considered a major issue before is due to the difference in time resolution between a pre-recorded trajectory and a live simulation. When using interactive molecular dynamics, the simulation is normally throttled at a certain rate to ensure the user has time to react to changes as they happen. Therefore,



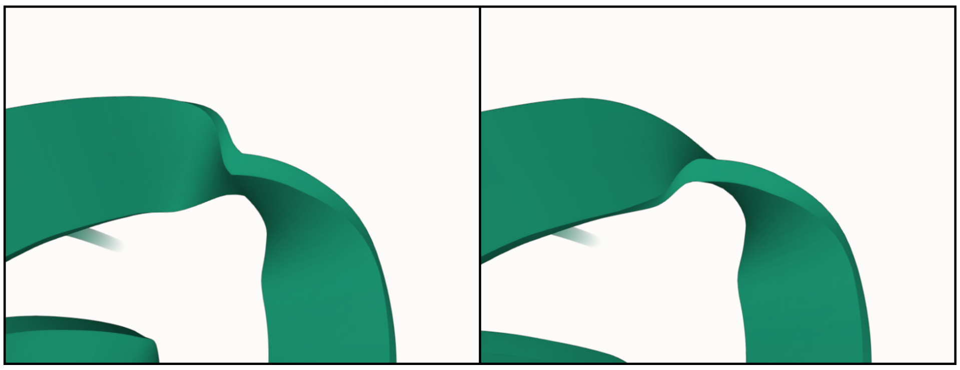
(a) PyMol



(b) VMD 1.9.4



(c) ChimeraX 1.3



(d) Mol* 3.0.2

Figure 4.18: Ribbon flipping as it occurs in consecutive frames of a protein trajectory in major visualisation software.

the relative simulation time per real-time second is smaller, and hence in real space terms, the atoms move more slowly than for watching back a trajectory. This gives the appearance of smoother, more continuous motion of the atoms.

Compare this to viewing a pre-recorded trajectory, where normally the positions are not recorded at every frame for performance and storage reasons. Here, atoms may move larger distances between each frame, as the interest here lies more within the global motion of the system rather than the small individual motions of each atom. Due to this large time separation, it appears less as the continuous motion of a system and more as a series of snapshots. Any visual discontinuities of the ribbon are therefore conflated with the large displacement of the ribbon between frames. We can therefore define a good measure of what would be considered a visual artefact in interactive simulations — a visual artifact is a sudden movement or displacement that occurs even in the limit of smaller timesteps between frames.

This is an issue which I have addressed in Narupa iMD by developing the **double-normal interpolation** technique. This technique does not suffer from these sudden flips, and hence provides visual continuity in secondary structure visualisation.

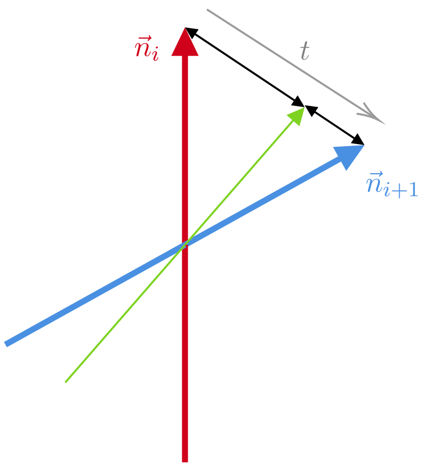
Double-normal method

Fundamentally, the discontinuity we are considering is caused by a change between a clockwise and anticlockwise twist. This twist's direction is directly related to how we interpolate the ribbon's normal between its initial and final value, as depending on their relative orientations either a clockwise or anticlockwise twist will be preferred. To prevent this artifact, we need an approach where two properties hold — smoothly changing either normal leads to a smooth change in the cross-section, and inverting either the initial or final normals does not alter the shape of the ribbon.

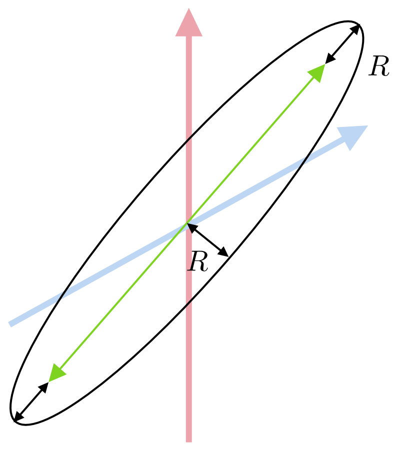
For this explanation, we will ignore the curving of the actual space curve, and instead consider a simple straight ribbon. The approach will be equally applicable to space curves. With this in mind, the problem now becomes a two-dimensional problem of rotating a cross-section from an initial rotation to a final rotation.

Our two-dimensional problem is defined by two normals, $\vec{n}_i$ and $\vec{n}_{i+1}$. These represent the initial and final orientations of this ribbon section. As we vary t from 0 to 1, the standard approach would be to interpolate between the two normals. Figure 4.19a shows an example of performing this using simple linear interpolation, followed by renormalising the normal. This normal can then be used to create a cross-section. For reasons that will become evident when describing the double-normal method, we will consider the case of an ellipsoid cross section. Here, some radius R is added to the normal, as well as used to widen the ribbon by the same amount as shown in Figure 4.19b.

It is clear to see that inverting $\vec{n}_{i+1}$ would cause the interpolation to switch from a clockwise rotation to an anticlockwise rotation. The double-normal method posits that instead of one



(a) Linear interpolation between $\vec{n}_i$ and $\vec{n}_{i+1}$.



(b) Ellipse created from radius R and interpolated normal.

Figure 4.19: Steps involved in simple linear interpolation of normals to create an ellipsoid cross-section.

normal, we instead define two separate normals:

1. An interpolation from $\vec{n}_i$ to $\vec{n}_{i+1}$ (or equivalently, $-\vec{n}_i$ to $-\vec{n}_{i+1}$).
2. An interpolation from $\vec{n}_i$ to $-\vec{n}_{i+1}$ (or equivalently, $-\vec{n}_i$ to $\vec{n}_{i+1}$).

Therefore, for any situation, one of these normals will be the *clockwise* normal and one will be the *anticlockwise* normal. For simplicity, I will define these two normals as $\vec{\eta}_i^+(t)$ and $\vec{\eta}_i^-(t)$, which will be given by linear interpolation:

$$\vec{\eta}_i^+(t) = (1-t)\vec{n}_i + t\vec{n}_{i+1}$$
$$\vec{\eta}_i^-(t) = (1-t)\vec{n}_i - t\vec{n}_{i+1}$$

Figure 4.20 illustrates how these two normals interpolate from $\vec{n}_i$ to $\vec{n}_{i+1}$. These two normals and their negatives define a parallelogram which will define our cross section, with corners $\vec{\eta}_i^+$, $-\vec{\eta}_i^+$, $\vec{\eta}_i^-$ and $-\vec{\eta}_i^-$. For our two normals, we instead choose the midpoints of the sides of this parallelogram, as these will be the directions in which we create our cross section. These normals are given by:

$$\vec{d}_i = \frac{1}{2}(\vec{\eta}_i^+ + \vec{\eta}_i^-) = (1-t)\vec{n}_i$$
$$\vec{d}_{i+1} = \frac{1}{2}(\vec{\eta}_i^+ - \vec{\eta}_i^-) = t\vec{n}_{i+1}$$

As shown above, these two normals can be interpreted in two different ways. The first is as the midpoints of the sides of the parallelogram as derived above. The other is that this is

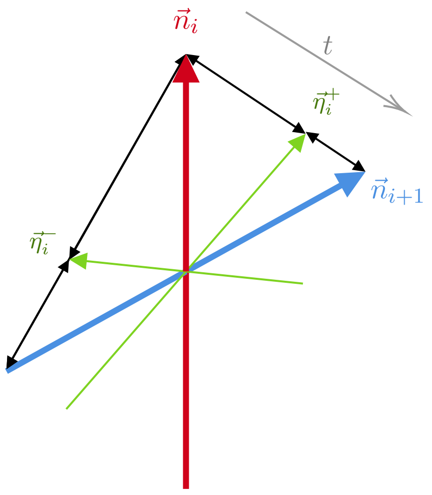


Figure 4.20: Linear interpolation of the two normals in the double-normal method.

simply a partition of a linear interpolation (lerp) into its two vector components (one along $\vec{n}_i$, one along $\vec{n}_{i+1}$):

$$\text{lerp}(\vec{n}_i, \vec{n}_{i+1}; t) = \underbrace{(1-t)\vec{n}_i}_{\vec{d}_i} + \underbrace{t\vec{n}_{i+1}}_{\vec{d}_{i+1}} \tag{4.7}$$

Therefore, $\vec{d}_i$ represents the diminishing contribution of $\vec{n}_i$, and $\vec{d}_{i+1}$ represents the increasing contribution of $\vec{n}_{i+1}$.

An ellipse can be defined as the action of a 2×2 matrix on the unit circle. Therefore, by calculating this matrix for each point along the ribbon, a circular cross-section can be deformed to give the corresponding elliptical cross-section. The 2×2 matrix $\mathbf{M}$ defines the axes of the ellipse, with the eigenvalues being the length of the major and minor axes, and their corresponding (orthogonal) eigenvalues their directions.

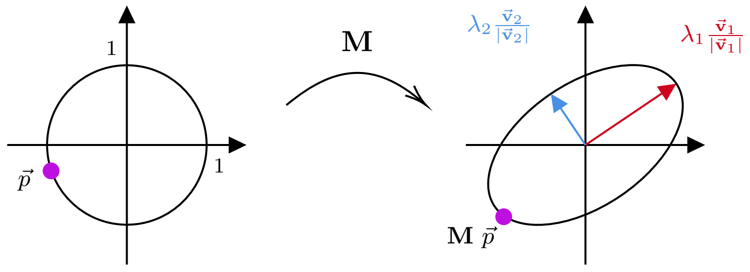

1
1
$\vec{p}$
$\mathbf{M}$
$\lambda_2 \frac{\vec{\mathbf{v}}_2}{|\vec{\mathbf{v}}_2|}$
$\lambda_1 \frac{\vec{\mathbf{v}}_1}{|\vec{\mathbf{v}}_1|}$
$\mathbf{M}\,\vec{p}$

Figure 4.21: Transformation of a circle with radius 1 into an ellipse using a matrix $\mathbf{M}$. The major and minor axes have lengths and directions dictated by the eigenvalues λ_i and corresponding eigenvectors $\vec{v}_i$ of the matrix $\mathbf{M}$.

Going back to the case for a single normal, the matrix M which converts a circular cross-section into an ellipse should have a major axis of length $W + R$ and a minor axis of length R, with the major axis being aligned along the normal $\vec{n}$. Here, W is the width of the ribbon, and R the radius. This can be done using the matrix:

$$\mathbf{M} = \frac{W}{|\vec{n}|}\begin{pmatrix} \frac{R|\vec{n}|}{W} + n_x^2 & n_x n_y \\ n_x n_y & \frac{R|\vec{n}|}{W} + n_y^2 \end{pmatrix} = R\mathbf{I} + W\frac{\vec{n}\otimes\vec{n}}{|\vec{n}|} \tag{4.8}$$

where $\otimes$ indicates the outer product of two vectors, and $\mathbf{I}$ is the 2×2 identity matrix. This matrix transforms a point $\vec{p}$ on the unit circle to the point:

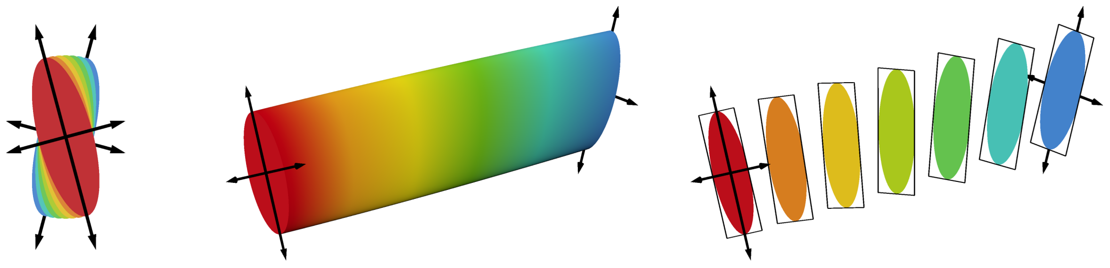
$$\mathbf{M}\vec{p} \mapsto R\vec{p} + W\frac{\vec{n}\cdot\vec{p}}{|\vec{n}|}\vec{n} \tag{4.9}$$

Our matrix $\mathbf{M}$ can be split into two parts — a scaling of the cross-section by a radius R, and a stretching of the cross-section in the direction of the normal $\vec{n}$, weighted by how much the point and the normal are aligned. The resulting ellipse has the desired major and minor axes lengths, and is aligned with $\vec{n}$.

In my double-normal method, we simply extend this equation by applying both normals $\vec{d_i}$ and $\vec{d}_{i+1}$ into the equation as two individual stretches:

$$\mathbf{M} = R\mathbf{I} + W\left\{\frac{\vec{d_i}\otimes\vec{d_i}}{|\vec{d_i}|} + \frac{\vec{d}_{i+1}\otimes\vec{d}_{i+1}}{|\vec{d}_{i+1}|}\right\} \tag{4.10}$$

We've shown that these normals are simply scalings of the initial and final normals, so the final matrix for my double-normal interpolation matrix can be written in terms of the initial


(a) Clockwise rotation

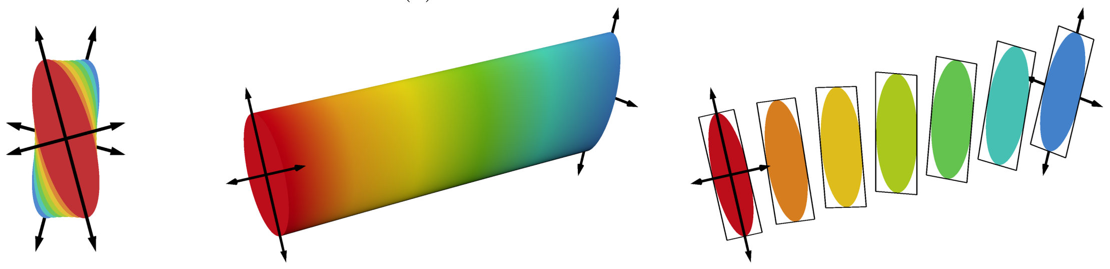

(b) Double-normal interpolation

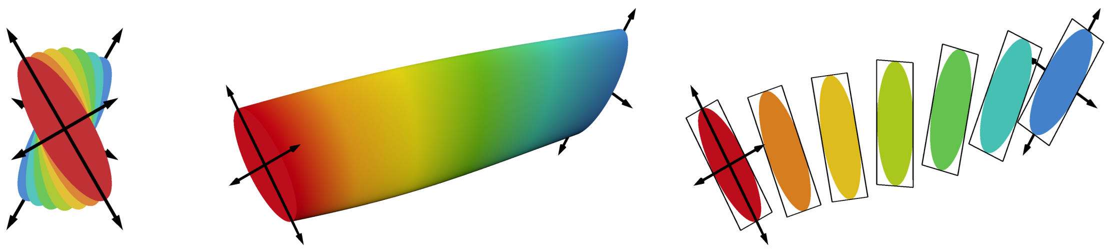
Figure 4.22: Comparison of my double-normal interpolation technique to standard ribbon rotation for a 30 degree ribbon rotation. At these small angles, my technique is virtually indistinguishable from a standard ribbon rotation.

and final normals as:

$$\mathbf{M} = R\mathbf{I} + W\left\{(1-t)\frac{\vec{n}_i \otimes \vec{n}_i}{|\vec{n}_i|} + t\frac{\vec{n}_{i+1} \otimes \vec{n}_{i+1}}{|\vec{n}_{i+1}|}\right\} \tag{4.11}$$

This highlights another interpretation of the difference between a simple single-normal interpolation and my double-normal interpolation. In a single-normal interpolation, the normal is interpolated from the initial normal to the final normal, and then a single stretch is performed along this intermediate direction. The double-normal method can be interpeted as an interpolation between two separate stretches — a stretch along the initial normal, and a stretch along the final normal. The factors of t ensure that at the start and end points, my method exactly matches that of the single-normal method. At intermediate points, it results in an ellipse which is aligned somewhere between $\vec{n}_i$ and $\vec{n}_{i+1}$. However, depending on the angle between $\vec{n}_i$ and $\vec{n}_{i+1}$, the ribbon is also inflated to become less elliptical and more circular. This is visible in Figure 4.23. At small angles, the double-normal method is approximately equal to the single-normal method, as shown in Figure 4.22.

Importantly, when the initial and final normals are at 90° to each other, the ellipse shrinks and forms a circular cross section midway along the ribbon. This entire transition involves shrinking and stretching the cross section along the ribbon, with no rotation. This is why my double-normal method avoids the flipping ambiguity - at the point the flip would occur, my method no longer involves any rotation.


(a) Clockwise rotation

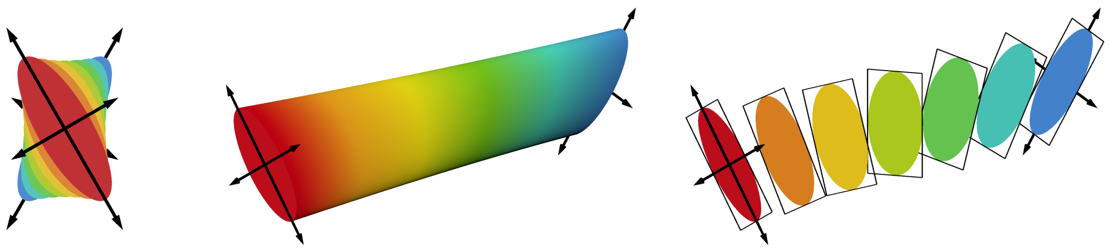
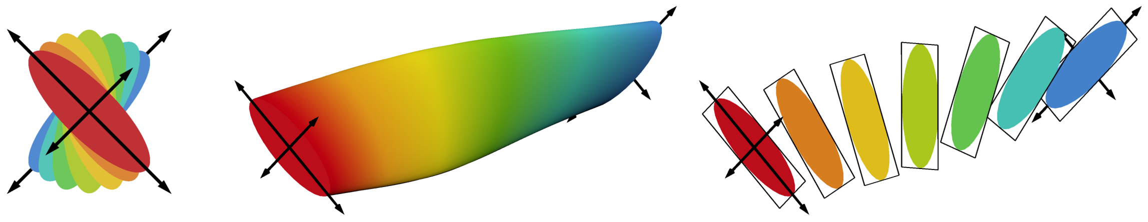
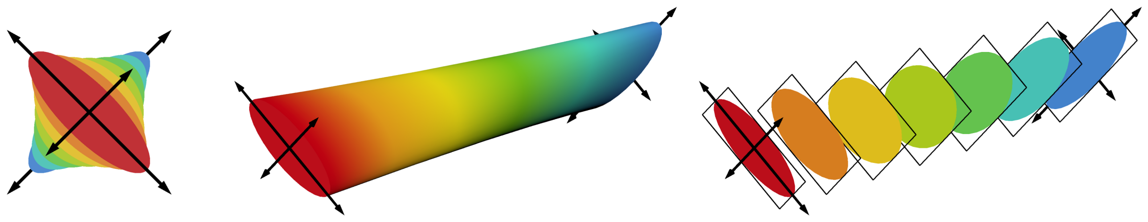

(b) Double-normal interpolation

Figure 4.23: Comparison of my double-normal interpolation technique to standard ribbon rotation for a 60 degree ribbon rotation. As the angle approaches 90 degrees, my technique begins to taper inwards at the centre.

One argument against this method is that the cross-section is no longer constant, and hence information about the alignment is lost. However, I believe the circular cross-section that arises actually has an important physical meaning. Consider Figure 4.24, where the ribbon could equally be seen as twisting clockwise or anticlockwise. Therefore, at the midpoint, there are arguably two possible interpretations of the orientation, which are at 90° to one another. The circular cross-section of the double-normal method clearly illustrates this as being a degenerate case, and by becoming circular is indicating that there is no longer a well-define direction of the ribbon. The double-normal method can be seen as not only a geometric trick, but as a way of encoding the uncertainty of the ribbon's orientation between well-defined endpoints as an appropriate morphing of an elliptical cross-section to a circular cross-section.

It can also be seen visually that the method gives similar images to that of a single-normal approach. Whilst Figure 4.24 may make it seem like a radical change, in practice I believe


(a) Clockwise rotation


(b) Double-normal interpolation

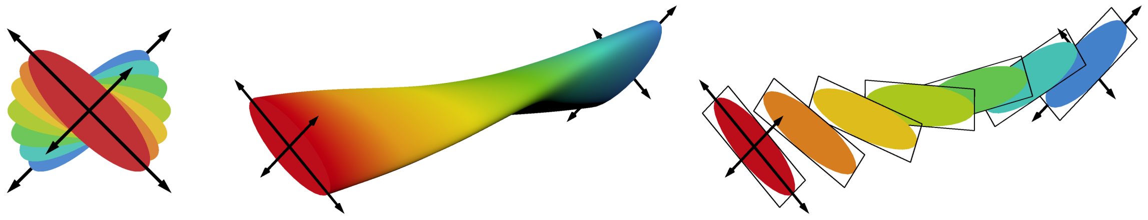
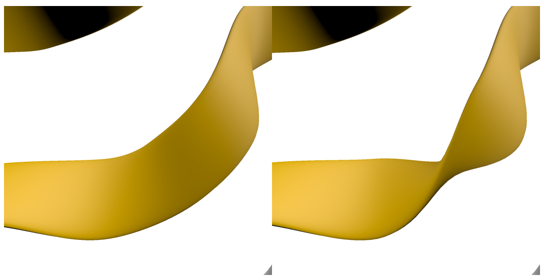

(c) Anticlockwise rotation

Figure 4.24: Comparison of my double-normal interpolation technique in comparison to standard ribbon rotation for the degenerate case where the angle between the initial and final normals is 90 degrees. My method avoids the ambiguity of which direction to rotate by instead using a squeeze-stretch approach.

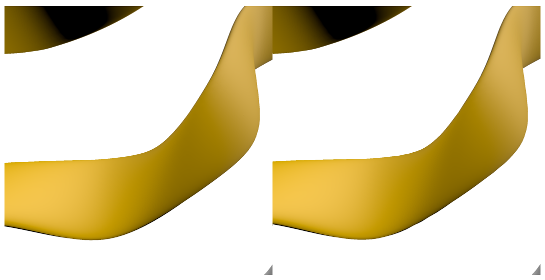
our eyes trick us into accepting that the ribbon is twisting normally. Therefore unless the user is looking very closely at the exact part of the ribbon, the double-normal technique does not massively alter the appearance of a ribbon diagram in comparison to methods in which ribbon-flipping is an issue. Figure 4.26 illustrates that at a distance the diagrams are virtually indistinguishable.

As Figure 4.25 illustrates, using this method Narupa iMD can render dynamic ribbon diagrams where the geometry of the ribbon does not undergo any sudden changes. Due to the discrete categorisation of secondary structure (as calculated via DSSP), parts of the ribbon may suddenly change colour and size in a dynamic simulation. We must therefore use a method which prevents sudden changes when residues transition between secondary structure motifs.

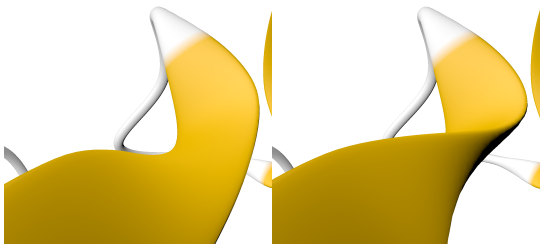
Videos illustrating the double-normal method may be found in the supporting information of the 2021 paper of Jamieson-Binnie and Glowacki[24].



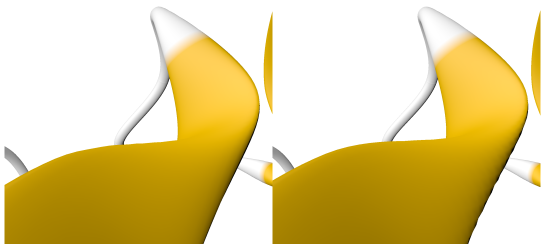
(a) Example of a β-sheet flip using existing visualisations.



(b) Example of β-sheet flip using my double-normal method.

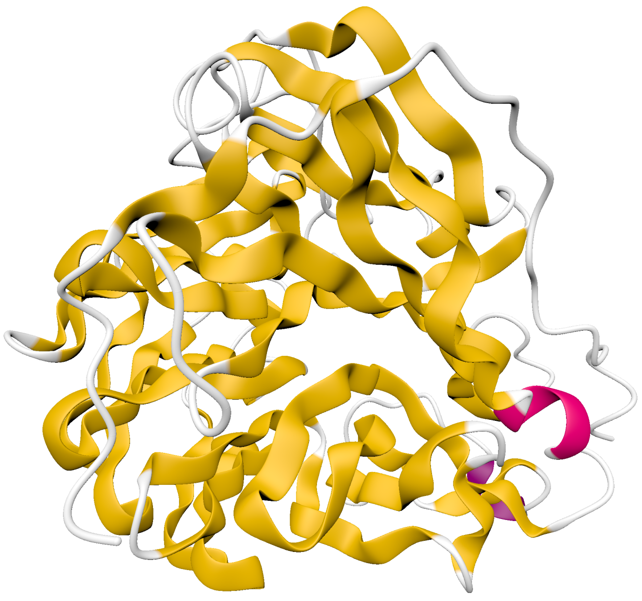


(c) Another example of aβ-sheet flip using existing visualisations.

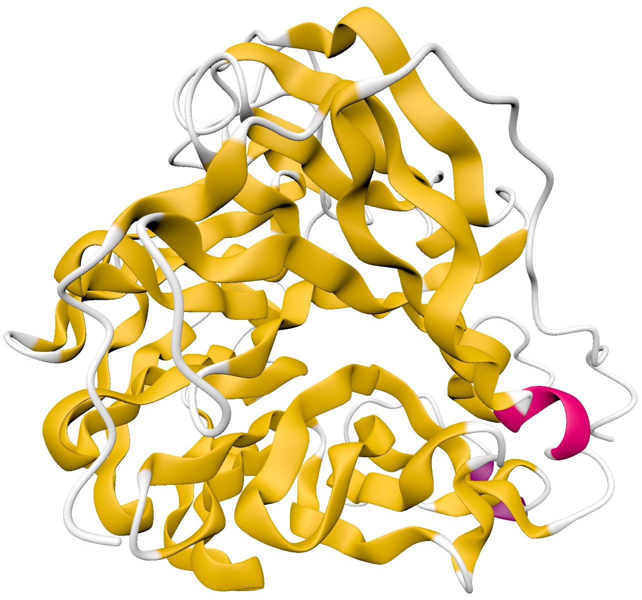


(d) The same example using my double-normal method.

Figure 4.25: Examples of flipping artifacts between adjacent timesteps that now are continuous thanks to my double-normal method.



(a) Neuraminidase visualised using linear interpolation of a single normal.



(b) Neuraminidase using my double-normal interpolation method.

Figure 4.26: Comparison of neuraminidase rendered using a standard single-normal interpolation and my double-normal interpolation method, illustrating their similarities at a distance.

Smooth Blending

The assignment of DSSP is used to determine the secondary structure of each residue, and hence the colour, width and shape of the ribbon at that point. For example, β-sheets are conventionally depicted in yellow and as long smooth ribbons; α-helices may be depicted in pink, but must not be smoothed too much to obscure the shape of the helix; and general turns and bends are often depicted as white tubes rather than ribbons. Whilst the above algorithm can blend between these different states smoothly, sudden changes occur when the movement of certain atoms causes a residue that was previously assigned to one secondary structure to be assigned to another.

This manifests itself as sudden changes in colour or size when DSSP is run. I address this by introducing a time delay on changes in these variables. When a new colour is assigned, the colour that is displayed is not instantly changed — instead, the colour is smoothly varied from the previous state to the new state. Figure 4.27 illustrates this technique, showing how there are no discontinuities in colour or size over time.

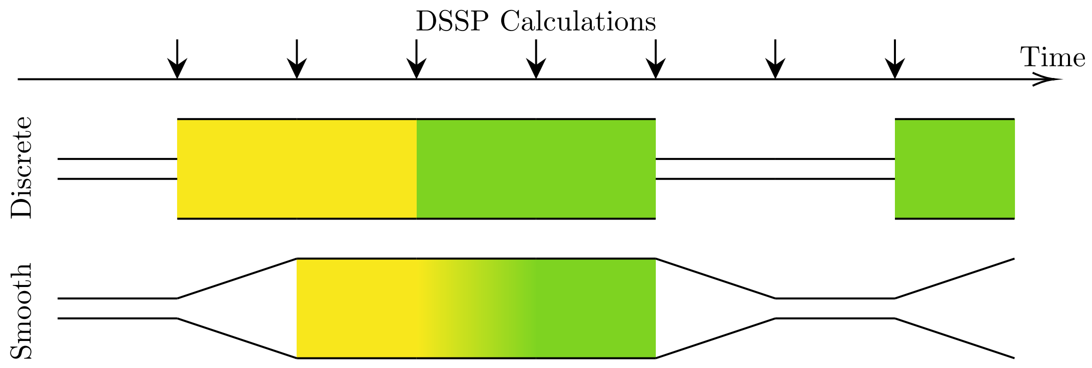


Figure 4.27: Comparison of discrete changes, where the DSSP calculation changes the colour and size directly; and smoothed, where the calculated changes are delayed to give a smooth transition.

4.6 Conclusions

Whilst many excellent reviews exist that go into detail on molecular surfaces,[113,201,202] the current usage of ribbon diagrams is comparatively less explored. I have therefore presented an overview of the current approaches found in various molecular software packages. As Table 4.1 illustrates, different software has adopted different types of splines to depict their ribbon diagrams. They are broadly split into three categories: Catmull-Rom or Cardinal spline approaches, which go through the α-carbons but can fail to capture the curved shape of helices; B-Splines, which are smoother but do not pass through the α-carbons; and natural cubic splines, which interpolate through the α-carbons and capture curvature well, but are more complicated to calculate. By analysing how these three cases handle ideal helices and β-sheets, we can draw some broad conclusions about their effectiveness. For helices, Catmull-Rom splines fail to accurately describe the helical geometry, whilst B-splines artificially reduce the radii of the helix. In this case, a natural cubic spline performs best. For β-sheets, the oscillating nature of the α-carbons can give a wavy appearance that is considered undesirable. In this case, a B-spline is often the better choice. Therefore, it is generally useful to combine different spline types, to successfully capture both the helix and β-sheet shapes.

Ribbon diagrams, whilst ideal for static structures, have certain features that make them ill-suited for dynamic visualisation — the discrete assignment of secondary structure elements and discrete flips between clockwise and anticlockwise twists. I address this first issue using a simple blending over time, to avoid sudden changes in colour or scale. For the ribbon, I have proposed a new double-normal algorithm that can generate an elliptical cross-section that never experiences a flip. The smooth nature of the shape means visually it is similar to an unmodified ribbon diagram. The nature of the blending also has a deeper meaning, as it adopts a circular cross section where the initial and final orientation are perpendicular. This can be taken as an indication of the uncertainty of the ribbons orientation at this point.

As Figure 4.11 illustrates, certain cubic ribbons can approximate helices rather well. However, even when the data points lie on a perfect helix, this cubic interpolation will not match exactly. Recent work by Yuksel[203] addressed a similar problem in two dimensions, fitting a spline through a set of points on a perfect circle. They showed that a technique based on blending circular segments could replicate circular arcs if the data points themselves lay on an arc. A similar approach could in principle be applied here, with helices fitted to consecutive sets of points, and a blending function used to smoothly combine them.

Chapter 5

Interactive Molecular Dynamics

In this final chapter, I will discuss the main application of the Narupa framework — interactive molecular dynamics. This involves the application of user-driven forces to parts of a simulation, which can be used to explore chemical pathways or as an educational tool.

I start this chapter with a more detailed overview of molecular dynamics, including force-fields and integrators. I will contextualise interactive molecular dynamics compared to other rare event sampling methods, such as metadynamics, umbrella sampling and steered molecular dynamics. This will segue into how we implement IMD into the molecular dynamics packages we employ, and examine how their different implementations present different challenges.

Interactive molecular dynamics requires defining a potential energy function to perturb the system. I overview the current methods of Gaussian and harmonic wells, and discuss their advantages and disadvantages

Finally, I will discuss preliminary work I have done on implementing a rotational interactive force. This addresses issues currently encountered when attempting to apply fine-grained rotations to small molecules without deformation, allowing the user to more precisely manoeuvre a ligand into a binding pocket. It also opens up Narupa to be applied to simulations with finite-sized or asymmetric particles, where interactive molecular dynamics cannot currently cause rotations.

5.1 Introduction

Molecular dynamics is the propagation of a molecular system through time, commonly through the application of classical dynamics. Most systems are approximated as a set of point-mass particles, with masses $\{m_i\}$, positions $\{\vec{q}_i\}$ and momenta $\{\vec{p}_i\}$. The dynamics of the system is governed by Newton's law, which may be written as the second order differential equation in terms of the atomic coordinates:

$$m_i \frac{\mathrm{d}^2 \vec{q}_i}{\mathrm{d}t^2} = \vec{F}_i(\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_N) \tag{5.1}$$

Here, $\vec{F}_i$ is the force experienced by the i-th particle. This force is often taken to be the gradient of some global potential energy function U, which itself is a function of the atomic coordinates:

$$\vec{F}_i(\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_N) = -\boldsymbol{\nabla}_{\vec{q}_i} U(\vec{q}_1, \vec{q}_2, \cdots, \vec{q}_N) \tag{5.2}$$

Except for certain small systems, equation 5.1 has to be addressed numerically. For a system of N atoms, this requires solving a system of N coupled differential equations. This is commonly achieved as an initial value problem — given a set of initial conditions, the differential equations are used to propagate the system forward in time in discrete steps. When tackling molecular dynamics in this manner, the problem may be broken down into three tasks:

1. The initial conditions of the system, including the positions and velocities for each atom.

2. A **forcefield**, which determines the potential energy function and hence force to be applied to each atom.

3. An **integrator**, which propagates the system forward in time under the influence of the forcefield.

Initial conditions

As Equation 5.1 is a second order differential equation, the initial positions $\{\vec{q}_i\}$ and velocities $\{\vec{v}_i\}$ of the atoms must be specified. The initial positions of the system can be obtained from experimental data such as crystallography. The velocities are normally sampled from a **Maxwell-Boltzmann distribution**, derived from the kinetic theory of gases. As each velocity is random and independent, a given choice may result in the entire system having an initial momentum. This can cause the simulation to drift in a general direction in the absence of a thermostat. This can be countered by either modifying the initial velocities, or performing an extra step during the integration to cancel out any centre of mass velocity.

Forcefields

The choice of molecular dynamics forcefields are broad, and can span from simple classical potentials to machine-learned potentials. There is a fine balance between increased accuracy and validity of more complex forcefields, and the computational cost that this imposes. Generally, a forcefield can be specified in terms of a potential energy function $U(\{\vec{q_i}\})$, with the forces given as the gradients with respect to each atom's position. The calculation of the energies and gradients is the most computationally expensive part of a molecular dynamics simulation.

The potential energy $U(\{\vec{q_i}\})$ is usually broken down into various terms, with various levels of experimental and theoretical justifications for their existence. These terms in turn are often dependent on relative atomic vectors $\vec{r_{ij}} = \vec{q_j} - \vec{q_i}$. If a potential energy term may be written as a function of these relative distances (for example, distances, angles and dihedrals), then it is invariant under translation and rotation. This invariance in the potential leads to the conservation of linear and angular momentum in our system.

Forcefield are generally broken down into various two-, three- and four-body terms. **Pair potentials** are terms which apply between all pairs of atoms in the system, regardless of any bonds defined for the system. One example of a pair potential is a Coulombic potential between charged particles. This electrostatic attraction is modelled according to Coulomb's law:

$$\vec{U}(\{\vec{q_i}\}) = \sum_{i>j} \frac{q_i q_j}{\epsilon r_{ij}} \tag{5.3}$$

where q_i is the charge of the i-th atom and ϵ is an appropriate dielectric constant.

Another pair interaction that can be modelled are van der Waals interactions. In reality, these occur due to electrostatic interactions due to instantaneous dipoles forming on each atom. As the shape and distribution of the electrons around each atom are not modelled in most molecular simulations, simple models are used to represent this low-level attraction. One of the most common and studied potentials for these forces is the **Lennard-Jones** or 12-6 potential:

$$\vec{U}(\{\vec{q_i}\}) = \sum_{i>j} 4\epsilon \left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6} \right] \tag{5.4}$$

where ϵ and σ are appropriate constants (that may depend on i and j) which determine the depth and location of the potential well. The pairwise van der Waals force fulfils an important role within a molecular forcefield, as its strong repulsive force at short distances prevents atoms from overlapping or colliding.

Pairwise potentials by themselves are only suited for simulating atoms or fluids consisting of single atoms or ionic bonds. In reality, certain atoms may form covalent bonds by the sharing of electrons. This yields a strong attractive force that hold certain atoms together. Generally, molecular dynamics is not concerned with the creation or destruction of covalent bonds, as this requires a large amount of energy. However, if the covalent bonds are instead modelled as rigid connections, this will not account for vibrational motion that can occur.

The compromise is to model bonds between covalently bonded atoms using a Taylor series. Commonly, this is taken to quadratic order and yields the harmonic potential:

$$U(\{\vec{q}_i\}) = \sum_i \sum_{j\in B(i)} \frac{k_{ij}}{2} \|\vec{r}_{ij} - d_ij\|^2 \tag{5.5}$$

The harmonic potential is quadratic about the idealised bond length d_{ij}, allowing fluctuations but preventing dissociation. Higher order polynomials such as quartics and sextics are also employed.

The interaction of covalent bonds with each other also lead to higher-order terms that affect three- and four-bodies. Between adjacent bonds, harmonic potentials are often used to enforce an ideal angle between three atoms. Likewise, dihedral potentials (often chosen to be sinusoidal to account for the rotational symmetry) affect the geometries of chains of four atoms. When greater accuracy is desired, these potentials may also be extended with higher order polynomial terms.

Some of the earlier forcefields were those proposed by Allinger for use with hydrocarbons, namely the MM1, MM2[204], MM3[205–208] and MM4[209–211] forcefields. These forcefields used higher order polynomials for their bonding interactions, such as a 6th-order polynomial for the bonding term. These forcefields also contain coupling terms between adjacent bonds and angles, which leads them to be often classified as Class II forcefields. This makes them better suited for calculating vibrational properties, which are influenced by coupling between vibrational modes.

In comparison to forcefields optimised for specific categories of molecule, others may aim to be a *universal* force field. Examples of this are the Merck Molecular Force Field (MMFF)[82], the Universal Force Field (UFF)[81] or the Generalized Amber Force Field (GAFF)[212]. While lacking in accuracy,[213] they are useful for geometry optimisation for a wide range of molecules. It is this feature that results in them often being implemented in general chemistry libraries for cheminformatics, such as RDKit[80] and Open Babel[60].

Integrators

An integrator is the method with which we propagate the positions and velocities forward in time, by making approximations that allow the calculation of $\{\vec{q}_i(t + \delta t)\}$ and $\{\vec{v}_i(t + \delta t)\}$ given $\{\vec{q}_i(t)\}$ and $\{\vec{v}_i()\}$. Generally, we track both positions and velocities during a simulation, which yields $2N$ first-order differential equations:

$$\frac{\mathrm{d}\vec{q}_i}{\mathrm{d}t} = \vec{v}_i \tag{5.6}$$

$$\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = \frac{1}{m_i}\vec{F}_i \tag{5.7}$$

The simplest algorithm for propagating these equations is the **Euler method**. This uses a first-order Taylor expansion for both velocities and positions:

$$\vec{q}_i(t+\delta t) = \vec{q}_i(t) + \vec{q}_i(t)\ \delta t \tag{5.8}$$

$$\vec{v}_i(t+\delta t) = \vec{v}_i(t) + \frac{1}{m}\vec{F}_i(t)\ \delta t \tag{5.9}$$

Whilst simple, the Euler method performs poorly when conserving energy over long periods of time. One method to address this is to use the **semi-implicit Euler method**:

$$\vec{v}_i(t+\delta t) = \vec{v}_i(t) + \frac{1}{m}\vec{F}_i(t)\ \delta t \tag{5.10}$$

$$\vec{q}_i(t+\delta t) = \vec{q}_i(t) + \vec{v}_i(t+\delta t)\ \delta t \tag{5.11}$$

In this method, the velocities are propagated first, and the velocity at $t + \delta t$ is used to propagate the positions. This simple change results in more stable dynamics, as this integrator is **symplectic**. Symplectic integrators work well for Hamiltonian systems (of which molecular dynamics is an example), where volume in phase space should be conserved.

Another common symplectic integrator is the **velocity Verlet** integration scheme. The equations of the velocity Verlet scheme are:

$$\vec{q}_i(t+\delta t) = \vec{q}_i(t) + \vec{v}_i(t)\ \delta t + \frac{\vec{F}_i(t)}{2m_i}\ (\delta t)^2 \tag{5.12}$$

$$\vec{v}_i(t+\delta t) = \vec{v}_i(t) + \frac{\vec{F}_i(t) + \vec{F}_i(t+\delta t)}{2m_i}\ \delta t \tag{5.13}$$

Equation 5.12 can be understood to be the Taylor series for $\{\vec{q}_i(t)\}$ to second order, and Equation 5.13 is the midpoint method for $\{\vec{v}_i(t)\}$ (with the assumption that $\vec{F}_i$ does not depend on the velocities). These symplectic algorithms allow for NVE ensemble simulations, where the energy, volume and number of particles is approximately constant.

For most simulations however we wish to run NVT ensembles, where now temperature instead of energy is conserved. Modifications to our integrator to allow for this are referred to as thermostats, and include the Berendsen, Andersen and Nose-Hoover. The Andersen thermostat randomly selects a subset of particles at each timestep, and assigns to these a new velocity sampled from a Maxwell-Boltzmann distribution. Meanwhile, the Berendsen thermostat scales the velocities in order to modify the temperature.

Another common way of running NVT ensembles is to modify Newton's law in a method known as **Langevin dynamics**. In Langevin dynamics, the core equation that is used for propagation of the system is:

$$m_i \frac{\mathrm{d}^2\vec{q}_i(t)}{\mathrm{d}t^2} = \vec{F}_i(t) - m_i\gamma\frac{\mathrm{d}\vec{q}_i(t)}{\mathrm{d}t} + \sqrt{2m_i\gamma k_B T}\vec{R}(t) \tag{5.14}$$

Langevin systems are therefore defined by two additional parameters, namely the damping constant γ and the desired temperature T. The two additional terms that appear to the right of equation 5.14 are a drag force which opposes and is proportional to the velocity of the atom, and a random force which moves the atom in random directions. To this end, $\vec{R}(t)$ is a random direction which is uncorrelated in time. These forces are meant to treat the atoms as if they were moving through some gas, where collisions with other molecules would slow them down and also impart random kicks. The size of these kicks is chosen such that over time, the temperature of the system will tend towards the desired temperature T.

5.2 Interactive Molecular Dynamics

Since its inception at the turn of the millennium,[66] **interactive molecular dynamics** (IMD) (sometimes referred to as interactive molecular simulation or IMS[214]) has been used in a range of applications, from drug docking[23,25,215] and molecular transport[216] to molecular modeling[217] and exploring reaction networks[27]. The traditional workflow of molecular dynamics involves visualising a simulation post-execution. Therefore, if an issue has occurred the entire simulation may have to be run again, costing both the user's time and compute cycles.[214] Interactive simulations can limit this by giving a user real-time direct feedback on the simulation.

One of the earliest examples of interactive simulations is the Sculpt program[218], used for continuous energy minimisation of a protein. Using a two-dimensional interface, small parts of a protein could be adjusted whilst the system was minimised. This prevented issues that may occur when making modifications using a simpler molecular builder.[218]

Building upon the concepts of Sculpt, the field of **steered molecular dynamics**[219] introduced the concept of time-varying predefined restraints to drive a molecular system to undergo a particular transition. By utilising the full breadth of molecular simulation, rather than just energy minimisation, SMD allows the exploration of time-based properties such as temperature and entropy.[219]

For example, when inspecting a ligand-receptor system Grubmüller, Heymann, and Tavan[220] observed for low restraint velocities, the rupture force could be extrapolated backwards to yield a result which agreed with atomic force microscopy experiments.[219]

Another approach similar to SMD is **self-guided molecular dynamics** (SGMD).[221] Instead of some user-defined acceleration, SGMD accelerates the existing motion of the system. The force experienced by each particle is time-averaged over a short period prior to the current timestep, which gives an indication of their overall motion. This time-averaged force is then scaled by a factor λ and reincorporated into the equations of motion. Therefore, SGMD measures overall motions occurring in the system and accelerates them. This technique has been applied to systems such as alanine dipeptide,[221] a β-hairpin,[222] as well as being applied to helix folding,[223] the isothermal-isobaric ensemble,[224] and Langevin dynamics.[225]

SMD involves defining ahead of time the biasing force and how it will vary, often by moving a harmonic constraint at a constant velocity along a pre-defined path. Interactive molecular dynamics aims to introduce human intuition by allowing a user to alter the forces on the fly while observing the system. The advantage of IMD over SMD are that it also allows exploration of the dynamics of a system.[70]

In order to accurately apply interactive forces in three dimensions, appropriate control over the interaction site is required. Two pieces of technology commonly used to perform IMD are haptic feedback devices[226] and virtual reality.[23] Using a haptic feedback device, a pen-like controller is suspended physically in space and can be moved by a user, manipulating a corresponding point in 3D space within the simulation. It hence grants the user three translational degrees of freedom. Virtual reality on the other hand often features wireless controllers held by the user, which can be orientated in space for a total of six degrees of freedom. Up to now, the forces applied using existing IMD-VR implementations use only the position of the controller, analogous to the haptic devices that preceded it. The additional control that a user may exert by rotating their hand is therefore absent.

An example of software capable of performing interactive molecular dynamics in virtual reality (IMD-VR) is the Narupa framework. Like common SMD forces, one or more particles can be pulled towards a point in space with a harmonic potential controlled by the user. In lieu of a harmonic well, a Gaussian well can be employed to avoid excessive forces at larger distances.[23] To apply a force to a group of atoms, the force to be applied is calculated as if a single point mass were located at the group's center of mass, which is then distributed between each atom weighted by their mass.[23] When the interaction is ended, there is also the option to reset the velocities of the affected particles to velocities sampled from the Maxwell-Boltzmann distribution, in order to cancel any net momentum they have built up.[23]

Rare Event Sampling

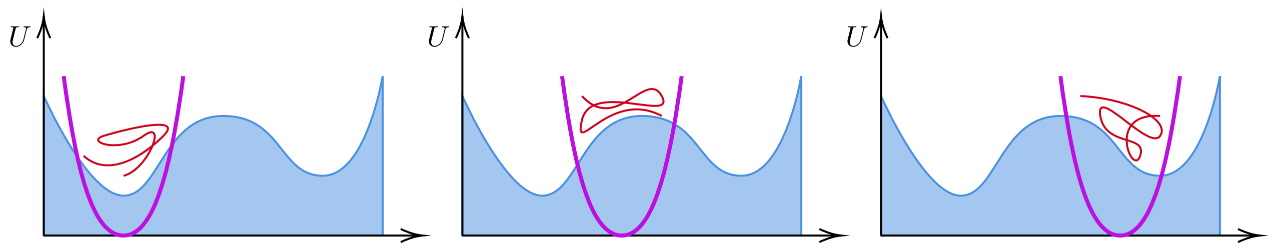
Generally, we calculate properties of our system based on the concept of **ergocity** — that given enough time our system will visit all parts of its potential energy landscape. However, the timescale over which chemically relevant processes may occur often limit what is explorable using unbiased molecular dynamics. Therefore, a wealth of methods collectively referred to as **biased molecular dynamics** have evolved to accelerate the simulation of rare events, whilst not unduly modifying the result. Optimising the time it takes to complete these calculations not only saves researchers' time, but reducing the computational load required to achieve the same result reduces the energy needed by the computing resources used.

These methods generally require the definition of a **reaction coordinate**. The reaction coordinate is a function of the current state of the system, and projects this into a low-dimensional space (commonly one-dimensional). This value characterises the particular reaction we wish to occur, with progression along the coordinate representing the path we wish to observe.

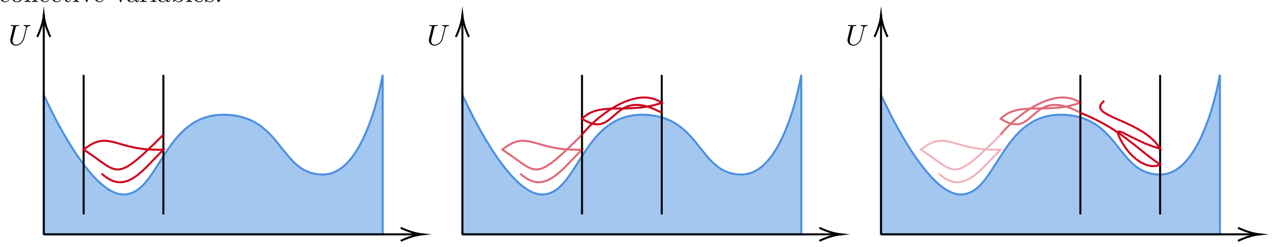
Umbrella sampling[227] is a method where multiple simulations are run, each fixed about a particular reaction coordinate. A potential well in reaction coordinate space limits the simulation to a particular region of the potential energy surface.

Metadynamics[228,229] is a method that is aims to prevent the repeated sampling of the same regions. It achieves this by accumulating small biasing potentials (usually Gaussian in shape) at the current point along the reaction coordinate of a simulation. Over time, this has the effect of filling in the wells in the potential energy surface the system may currently occupy, and encourages it to explore other regions.

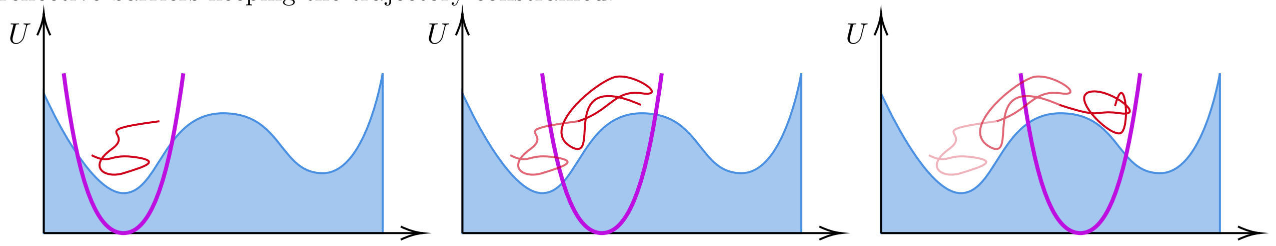
Another method is **boxed molecular dynamics**[230] (BXD). In BXD, the reaction coordinate is sliced into boxes, defining a minimum and maximum value the reaction coordinate may



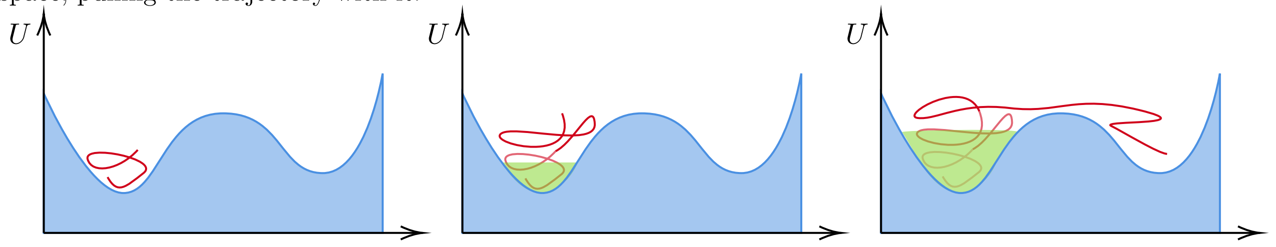
(a) Umbrella Sampling. Separate simulations are run with biasing potentials positioned at different collective variables.



(b) Boxed Molecular Dynamics. The simulation is constrained to successive regions of CV space, with reflective barriers keeping the trajectory constrained.



(c) Steered Molecular Dynamics. A constraining potential is moved at constant velocity across CV space, pulling the trajectory with it.



(d) Metadynamics. As the trajectories spents time in a region, Gaussians are deposited to fill in potential wells and allow the trajectory to escape.

Figure 5.1: Comparison of various rare event sampling techniques.

take. The simulation proceeds freely, but every time the system's dynamics would cause it to cross a boundary, all velocities within the system are reflected. Therefore, over a period of time, the relative rates at which the simulation attempts to cross the boundary either side can be measured. By then allowing the simulation to evolve to the next partition and repeating, the overall reaction coordinate is discretised, with rates describing how the system would like to evolve between these bins. By renormalising the results, BXD allows the computation of free energy profiles. BXD has been applied to a variety of systems,[231] as well as modified to be carried out in energy space.[232,233]

These methods are all based on a reaction coordinate, which must be predefined in some manner. There are many ways that this process can be automated, such as applying principal component analysis (PCA)[234,235] or time-structure independent component analysis (tICA)[236–238] to a trajectory. However, these approaches still require an original trajectory to be made.

For example, Sultan and Pande[239] used tICA to generate collective variables to drive metadynamics. Interactive molecular dynamics, whilst similar to steered molecular dynamics in how it can pull the simulation into new regions, finds better use as a tool to explore a system and define a reaction path before employing other techniques. This effectively uses IMD to establish an appropriate reaction coordinate. PCA has been successfully used by Deeks et al.[25] to study the binding pathways produced in a user study utilising interactive molecular dynamics. In a subsequent study Deeks et al.[240] showed that calculating the free energy along paths created in virtual reality can distinguish between favourable and unfavourable pathways.

It is in this niche that interactive molecular dynamics can find its home. Due to the large amount of simulation time and replications required, it is unsuitable for directly probing rare events in a similar manner to steered molecular dynamics. But instead, it can be used to explore systems, to build intuition for how they react to perturbations and to create trajectories that can go on to form the basis for other sampling techniques.

Molecular Dynamics in Narupa iMD

Narupa iMD is an adaptable framework that can support a wide range of MD software through a Python interface. The exact implementation of IMD therefore depends on the methods and techniques available in the relevant packages.

OpenMM

OpenMM is a popular GPU-accelerated library for running molecular simulations, especially protein systems.

A forcefield in OpenMM is split into discrete terms, each which specifies the force, energy and which atoms the force acts upon. Each term is implemented in a separate class, such as `HarmonicBondForce` or `PeriodicTorsionForce`. OpenMM supports extending these possible forces in two different ways. Firstly, a plugin may be written in C++, which allows complex forces to be programmed directly for the GPU. However, it also provides a family of so called *custom* forces. These are forces which accept as an argument an arbitrary function defining the potential energy.. This expression can consist of simple mathematical operations (such as exponentials, powers and trigonometric functions) and refer to various properties such as atomic position, bond lengths and dihedral angles. OpenMM can then calculate the forces by differentiating these expressions.

If we have already computed a force $\vec{F}_i^I$ to apply to each atom, we can define a custom OpenMM force with energy given by:

$$U(\{\vec{q}_i\}) = -\sum_i \vec{F}_i^I \cdot \vec{q}_i \tag{5.15}$$

This potential is chosen such that the force on the i-th atom is $\vec{F}_i^I$. OpenMM custom forces allow the specification of per-atom parameters, so the interactive force can be controlled by supplying the $3N$ parameters required to define $\left\{\vec{F}_i^I\right\}$.

This approach is not without its issues. In Narupa, OpenMM is invoked from Python one integration step at a time. The forcefield parameters of the custom force however cannot be altered during the step. Generally, in an integration scheme such as velocity Verlet the forces $\left\{\vec{F}_i(t+\delta t)\right\}$ must be calculated midway during the step.

$$\begin{aligned}
\vec{q}_i(t+\delta t) &= \vec{q}_i(t) + \vec{v}_i(t)\delta t + \frac{1}{2m_i}\left[\underbrace{\vec{F}_i(t) + \vec{F}_i^{\mathrm{I}}(t)}\right]\delta t^2 \\
\vec{v}_i(t+\delta t) &= \vec{v}_i(t) + \frac{1}{2m_i}\left[\underbrace{\vec{F}_i(t) + \vec{F}_i^{\mathrm{I}}(t)} + \underbrace{\vec{F}_i(t+\delta t) + \vec{F}_i^{\mathrm{I}}(t+\delta t)}\right]\delta t
\end{aligned}$$

The calculation of $\left\{\vec{F}_i(t+\delta t)\right\}$ proceeds between these two steps, as it relies on the value of $\{\vec{q}_i(t+\delta t)\}$ and is required to calculate $\{\vec{v}_i(t+\delta t)\}$.

Most forces in OpenMM are calculated when requested by OpenMM itself, and hence can be incorporated into the dynamics correctly. When adopting the approach of equation 5.15, the calculation of $\left\{\vec{F}_i^I\right\}$ is performed outside of OpenMM, and hence cannot occur during the integration step. Instead, the interactive part of the force is not recalculated between the position step and the velocity step:

$$\begin{aligned}
\vec{q}_i(t+\delta t) &= \vec{q}_i(t) + \vec{v}_i(t)\delta t + \frac{1}{2m_i}\left[\vec{F}_i(t) + \vec{F}_i^{\mathrm{I}}(t)\right]\delta t^2 \\
\vec{v}_i(t+\delta t) &= \vec{v}_i(t) + \frac{1}{2m_i}\left[\vec{F}_i(t) + \vec{F}_i(t+\delta t) + 2\vec{F}_i^{\mathrm{I}}(t)\right]\delta t
\end{aligned}$$

Generally, with the timesteps in question this is not a major issue. As interactions are large artificial forces anyway, it is not important if there is a slight deviation due to their incorporation into the integration scheme. Addressing this issue could only be solved by implementing a custom force in OpenMM directly in C++, by writing a plugin. However, as separate versions have to be written for the various platforms OpenMM supports, such as CUDA and OpenCL, it is simpler to accept the slight deviations and only update the interactive forces between the timesteps.

ASE

The Atomic Simulation Environment (ASE)[76] is written entirely in Python, and hence well suited for prototyping and modification. Instead of the separate force terms that OpenMM employs, ASE attaches a `Calculator` object to the system of interest. This object's role is to provide the forces and potential energy when requested. Importantly, it is informed when the positions have changed and hence knows when a force needs to be recalculated. This calculator-based approach allows ASE to be easily integrated with a wide range of other packages such as OpenMM and LAMMPS.

More importantly, it allows interactive forces to be propagated in the same manner as the non-interactive forces. As both ASE and the interactive forces are calculated through Python, the interactive forces can be recalculated on demand at the right point during the integration step.

LAMMPS

When LAMMPS[67] is compiled with the `PYTHON` package, it enables LAMMPS to execute arbitrary python code. LAMMPS can execute code either through its own interpreter, or using the driving Python interpreter. This arbitrary code may be executed at two points — at the end of each timestep or after each force computation. This second approach (called the `post_force` callback) allows the incorporation of interactive forces at the correct point.

The implementation of LAMMPS integration in Narupa iMD utilises this function. In that case, LAMMPS is run directly, with a Narupa server created within the Python interpreter used by LAMMPS. However, this requires direct modification of the LAMMPS input scripts, and is ill-suited for modification. In `narupatools`, LAMMPS is instead called from Python. This also uses the `post_force` callback, but this invokes the calculation in the calling Python library. The extensible nature of LAMMPS means that it supports a wider variety of systems than OpenMM. Of note are simulations that involve particles which may have orientations or non-uniform sizes. An example of a model implemented in LAMMPS is the ox-DNA coarse-grained model for DNA. Visualisation of these systems requires the extraction of data such as size and orientation from LAMMPS, to be transmitted to the frontend. `narupatools` uses the feature-rich LAMMPS API to extract this data and add it to the frames sent to the user.

Interactive Potentials

The incorporation of user interactions into an existing molecular dynamics workflow involves adding additional forces to the system. These interactive forces should be easy to control by the user, and be applicable to a wide range of simulations.

In current implementations of IMD, interactions are point-based. Here, the user applies an interaction at a specific point $\vec{c}$ in space, and the target particles are pulled towards this point. This point could be controlled by the position of a haptic pen, or by the location of a virtual reality controller. This approach is similar to SMD, where $\vec{c}$ is a position which moves along a predesignated path.

In SMD, the attractive potential used is usually chosen to be a harmonic potential, of the form:

$$U(\vec{q}_i) = \frac{1}{2}k\|\vec{q}_i - \vec{c}\|^2 \tag{5.16}$$

Here, the constant k is the *spring constant* determining the strength of the interaction and $\vec{c}$ is the interaction site.

This appeal of using this potential in IMD is its simplicity and similarity to other applications, such as SMD and harmonic restraints. However, in these other applications an implicit assumption is that $\vec{c}$ and $\vec{q}_i$ are close together. This is why the harmonic potential is used — at the local minimum of a potential surface, we can approximate it as a harmonic well. By using a large value of k and using a static or slowly varying $\vec{c}$, techniques such as SMD and harmonic restraints ensure that the interacted particle is always kept close to the interaction site $\vec{c}$.

However, in IMD this can break down, either because the user is moving $\vec{c}$ at large speed or because the user has started an interaction at a great distance from $\vec{q}_i$. In these situations, the potential has a very large value, and hence leads to correspondingly large forces. This is because the force increases linearly as we increase the distance between $\vec{q}_i$ and $\vec{c}$. This makes a harmonic interaction potential ill-suited for long-range interactions as it can lead to extreme forces that crash a simulation.

This issue was addressed in Narupa iMD by introducing a second potential, the **Gaussian**

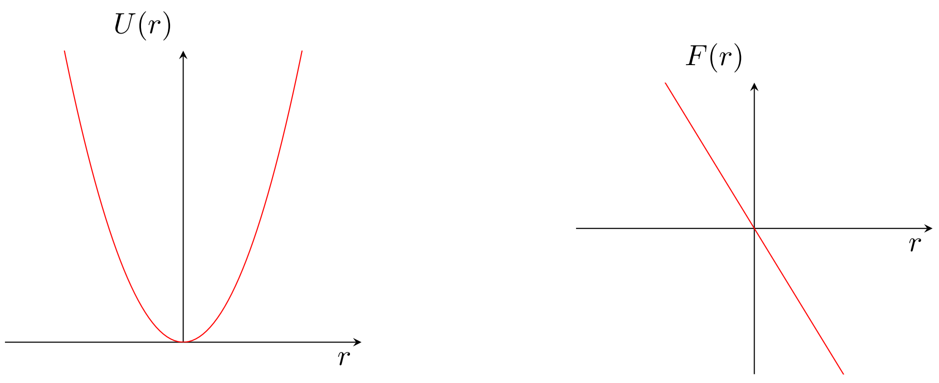


Figure 5.2: Harmonic Potential $U(r)$ and corresponding force $F(r)$.

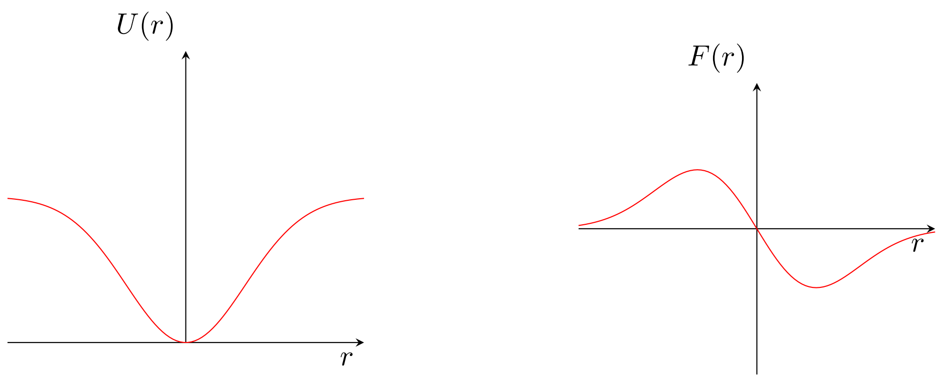


Figure 5.3: Gaussian Potential $U(r)$ and corresponding force $F(r)$.

well.[23] The Gaussian well has the potential energy:

$$U(\vec{q}_i) = -ke^{-\frac{\|\vec{q}_i - \vec{c}\|^2}{2}}$$

The Gaussian potential has the same quadratic behaviour at small distances, but tends to a constant value at larger distances. The corresponding force as a function of distance therefore initially increases linearly, but at large distances tends to 0. Therefore, at large distances a Gaussian potential has no effect, and the user has to deliberately keep their interactions close to the targeted atoms to have an effect.

Narupa iMD allows the user to counter the unstable nature of the harmonic force by specifying a maximum force, F_{max}. This clipped harmonic potential applies the same force as the harmonic potential, but the magnitude of the force cannot exceed F_{max}. This has the effect of limiting the slope of the potential, and hence preventing abnormally large forces at large distances. The potential energy of this interaction is given by:

$$U(\vec{q}_i) = \begin{cases} \frac{1}{2}k\|\vec{q}_i - \vec{c}\|^2 & \|\vec{q}_i - \vec{c}\| < \frac{F_{\text{max}}}{k} \\ -\frac{1}{2}\frac{F_{\text{max}}^2}{k} + F_{\text{max}}\|\vec{q}_i - \vec{c}\| & \|\vec{q}_i - \vec{c}\| \geq \frac{F_{\text{max}}}{k} \end{cases} \tag{5.17}$$

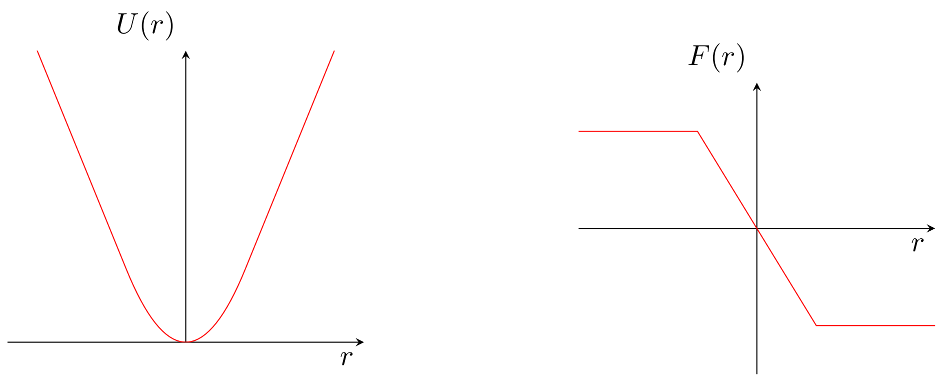
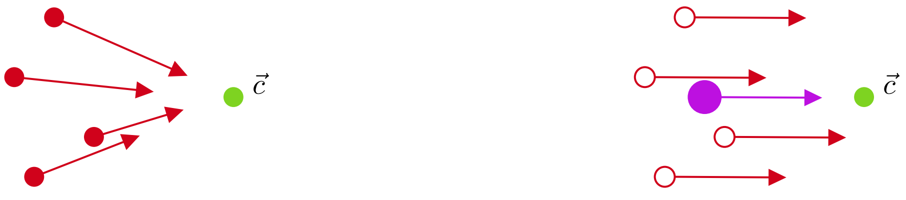


Figure 5.4: Clipped Harmonic Potential $U(r)$ and corresponding force $F(r)$.

In Narupa iMD, the user can alter the value of k to achieve an interactive force which applies enough force to alter the simulation sufficiently whilst not enough to cause excessive perturbations. Note that the units of k depends on the type of interaction — in the standard Narupa unit system, k has units of $\mathrm{kJ\,mol^{-1}\,nm^{-1}}$ for the harmonic and clipped harmonic potentials, and $\mathrm{kJ\,mol^{-1}}$ for the Gaussian potential.

Multi-particle interactions



(a) Forces applied individually to each particle. (b) Forces distributed using mass-weighting.

Figure 5.5: Comparison of applying an interaction to each particle individually, or applying it to a composite particle at the centre of mass and then redistributing the forces using mass-weighting.

The previous potentials work well for single particles, but often we wish to interact with whole fragments or molecules. In this situation, applying the interaction individually to each atom causes each atom to move towards the same point in space as shown in Figure 5.5a. This results in the molecule deforming as they are brought closer together.

In SMD, restraining multiple particles is achieved by restraining their centre of mass. The potential for this kind of interaction now becomes:

$$U(\{\vec{r}_i\}) = U_r\left(\left\|\frac{1}{M}\sum_i m_i(\vec{r}_i - \vec{c})\right\|\right)$$

Here, U_r is the potential in terms of a distance r, and M is the total mass of the interacting particles. Calculating the force on each participant atom yields:

$$\vec{F}_i = -\nabla_{\vec{q}_i} U_r(\|\vec{q}_{\mathrm{COM}} - \vec{c}\|) \tag{5.18}$$

$$= -\frac{m_i}{M}\nabla_{\vec{q}_{\mathrm{COM}}} U_r(\|\vec{q}_{\mathrm{COM}} - \vec{c}\|) \tag{5.19}$$

$$= \frac{m_i}{M}\vec{F}_{\mathrm{COM}} \tag{5.20}$$

Therefore, this approach can be considered as approximating the set of interacted particles as one virtual particle — located at the centre of mass and with mass equal to the total mass of the interacted particles. After calculating this force $\vec{F}_{\mathrm{COM}}$ that the composite particle would experience, this can then be redistributed amongst the other atoms by **mass-weighting**.

This equation is also derivable from the criteria that each particle must experience the same acceleration as the composite particle, such that they do not experience any force that would cause them to move relative to one another.

Note that in the Narupa paper by O'Connor et al.[23], Equation 5.20 is incorrectly given with a division by the number of atoms N, instead of the total molecular mass M. This mistake was repeated within Narupa iMD.[a] Whilst this does not have an effect on the calculations themselves, it means a benzene molecule (with molecular mass of $78\,\mathrm{g\,mol^{-1}}$) would be affected by a significantly weaker force than a single Selenium atom (with an atomic mass of $79\,\mathrm{g\,mol^{-1}}$). This is rectified in the `narupatools` implementation of IMD.

[a]As of version 0.1.2118

5.3 Rotational Interactions

Using the interactions described thus far, we can apply a translational force on a set of atoms, causing them to collectively move towards a point in space. However, using the existing framework for interactive molecular dynamics, we are unable to easily induce rotations of a fragment. This is vitally important in many chemically significant situations, such as the geometry-dependent docking of a ligand molecule into a protein. Implementing an ability to rotate molecules gives the user far greater control over its movement and orientation.

Previous approaches performing rotations within steered molecular dynamics have involved rotating a reference set of positions, and using standard linear harmonic restraints to affect the system.[241–244] This approach can be found in various packages such as NAMD[65] and GROMACS.[58] By specifying an initial reference arrangement and an axis of rotation (with corresponding angular velocity), reference positions at a later time t are given by rotating the initial reference positions. Each atom is pulled towards this new reference position by a linear spring force.

To rotate long molecular chains within pores, an improved SMD approach known as the **flexible-axis** approach has also been developed.[245] Here, different fragments of a longer chain are rotated around different axes, allowing an overall rotation around a curve. However, all these approaches have a common flaw — they pull each atom towards a reference position that freezes the molecule in a certain arrangement. Therefore, this approach limits the ability of the molecule to have internal degrees of freedom as it is rotated.

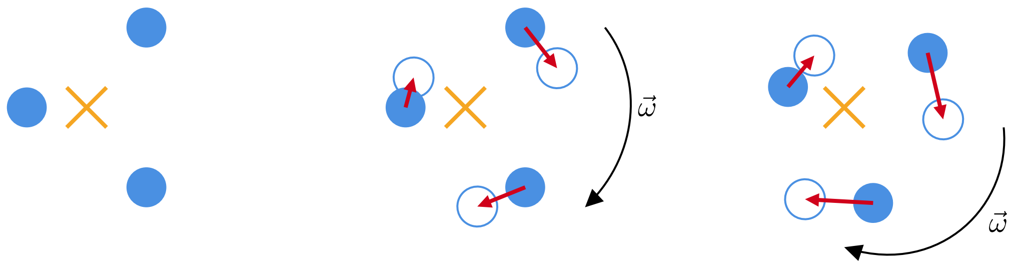


Figure 5.6: Example of rotational SMD, with each atom linearly attracted to a rotating constraint position, which rotates about an axis at constant velocity $\vec{\omega}$.

Whilst standard translational SMD is transitioned to IMD by replacing the pre-defined constraint point by the position of the controller, this existing approach to rotational SMD has no clear way to be adapted to IMD. Applying rotations to molecules in the existing framework of IMD involves applying a standard translational interaction to one or more particles near the edge of the molecule. By moving these particles in an arc, the internal forces holding the molecule together will cause the overall molecule to rotate as well.

However, this as several disadvantages. As Figure 5.7a illustrates, this approach causes the molecule to deform. This is because we are applying unbalanced forces on the molecule. It also relies on the internal forces of the molecule to prevent the atoms from moving apart, and hence is not applicable to non-bonded or weakly-bonded sets of particles.

Up to this point, when IMD has been applied to protein-ligand docking,[23,25,215] the ligand evolves freely with no constraints. This is termed flexible docking. In contrast, rigid body docking fixes the internal motion (bond lengths, angles and dihedrals) of the ligand, such that it acts as a single undeformable object. This approach of freezing the ligand to a single inflexible conformation treats the ligand as a **rigid body** — an arrangement of atoms whose relative positions and angles are fixed. Incorporating rigid bodies into dynamics relies on modifications to the equations of motion.[246–248] Ideally, we would like a method that combines the best of both flexible and rigid docking — the molecule would be free to vibrate and change conformation, whilst allowing us to apply overall translations and rotations.

In this section, I will discuss a new class of forces for interactive molecular dynamics that induce rotations within a molecular fragment. These interactions will be termed **rigid motion interactions**, to be contrasted with the harmonic and Gaussian interactive potentials described in the previous section. The term rigid motion refers to transformations which translate or rotate an object. The interactive force will apply a per-atom force which will induce overall rotation and translation of a molecule, without deforming the internal structure. In this manner, it will act similarly to rigid body docking, but still allow for internal motion within the fragment. By using per-atom forces rather than rigid body dynamics, this approach can also be incorporated easily within the existing molecular dynamics ecosystem, with packages such as ASE,[76] OpenMM[78] and LAMMPS[67] allowing custom forces to be defined and applied.

Interactions in IMD are usually applied to a specific subset of atoms within the system, usually a small molecule such as a ligand which is not chemically bonded to any other part of the system. This set of atom indices will be denoted I, with $\{m_i\}$ and $\vec{()}R_i$ denoting the mass and absolute position (relative to the global origin) of each particle. Likewise, $\vec{V}_i$ and $\vec{A}_i$ are the velocity and acceleration of the i-th particle relative to the global origin.

When discussing rotations, it is useful to consider the positions of the atoms involved in the interaction relative to their centre of mass. The centre of mass $\vec{R}_{\text{com}}$ is defined by:

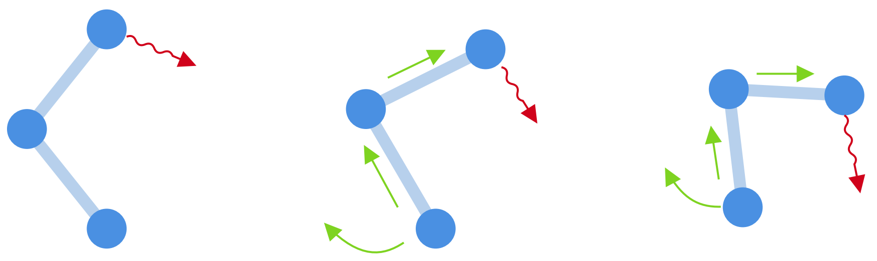
$$\vec{R}_{\text{com}} \equiv \frac{\sum_{i\in I} m_i \vec{R}_i}{M} \tag{5.21}$$

where $M = \sum_{i\in I} m_i$ is the total mass of the selection.

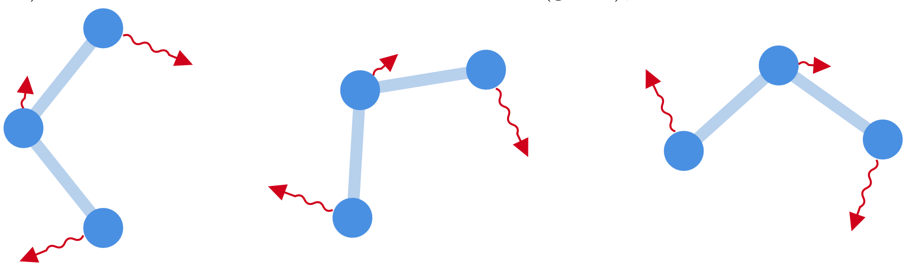
The coordinates of each atom may therefore be expressed relative to the centre of mass:

$$\vec{r}_i \equiv \vec{R}_i - \vec{R}_{\text{COM}} \tag{5.22}$$

Likewise, $\vec{v}_i$ and $\vec{a}_i$ are the velocity and acceleration of the particle relative to the velocity and acceleration of the centre of mass. From now on, we will refer to this subset of particles I as a molecule or fragment, though in general the atoms could be attached to other non-interacted atoms and not bonded to each other.



(a) Example of rotating a small molecule using an interactive force on one particle (red). This causes rotation due to internal forces (green), but deforms the molecule.



(b) Example of rotating a small molecule by applying interactive force to all particles. This causes a smooth rotation with no deformations.

Figure 5.7: Comparison of rotating a small molecule using a single interaction on an atom and by applying a set of interactive forces to induce a rotation.

In the rest of this chapter, I will describe the rigid motion interaction that will apply rotational forces such as Figure 5.7b. It will be applicable to not just sets of point masses, be also finite and asymmetric particles often encountered in coarse-grained potentials.

Rotational Dynamics

Molecular dynamics is usually carried out under the assumption of isotropic point masses. Under these assumptions, though commonly depicted as spheres, each atom is located at a single point in space, with all its mass located at at one point. As such, it has no defined orientation or direction, and its dynamics are uniquely specified by its position and momentum/velocity.

A **rigid body** is obtained if we fix some of these atoms such that they are rigidly connected together, and cannot move closer or further apart from one another. This composite body can only move in two ways — translation of the body as a whole, or rotation about its centre of mass. If this object rotates, the atoms further from the centre of mass have further to travel, and hence must move at a higher relative velocity. Whilst each constituent atom of the rigid body may be moving at a different velocity, the overall rotational motion may be described by an **angular velocity** $\vec{\omega}$ — a vector that passes through the centre of mass, whose direction determines the axis of rotation. The magnitude of this vector determines the speed of the rotation, in radians per second.

The velocity of each constituent atom therefore consists of two terms — an overall translational motion $\vec{V}_{\text{COM}}$ of the centre of mass and a rotational velocity about the centre of mass:

$$\vec{V}_i = \vec{V}_{\text{COM}} + \vec{\omega} \times \vec{r}_i \tag{5.23}$$

Another quantity used to describe a rotating object is the **angular momentum** $\vec{L}_i$. When talking about angular momentum, we must also specify the pivot point. For example, this contrasts the orbital angular momentum of an electron about a nucleus to the spin angular momentum of the electron about its own axis. Here, we are talking about an orbital angular momentum of each particle about the centre of mass, which is given by the cross product of the relative position to the centre of mass and its momentum relative to the centre of mass:

$$\vec{L}_i = m_i \vec{r}_i \times \vec{v}_i \tag{5.24}$$

Like linear momentum, the total angular momentum of the rigid body is given as the sum over the angular momentum of its constituent atoms:

$$\vec{L} = \sum_i m_i \vec{r}_i \times \vec{v}_i \tag{5.25}$$

As shown in equation 5.23, for a rigid body the relative velocity $\vec{v}_i$ is given by the cross product $\vec{\omega} \times \vec{r}_i$. Hence, for a rigid set of atoms the total angular momentum is given by:

$$\vec{L} = \sum_i m_i \vec{r}_i \times (\vec{\omega} \times \vec{r}_i) \tag{5.26}$$

By denoting the action $\vec{r}_i \times$ as the skew-symmetric matrix $[\vec{r}_i]$, the total angular momentum $\vec{L}$ is expressible as a matrix multiplication of the angular velocity $\vec{\omega}$:

$$\vec{L} = \mathbf{I}\vec{\omega}, \quad \mathbf{I} \equiv -\sum_i m_i [\vec{r}_i]^2 \tag{5.27}$$

This matrix $\mathbf{I}$ is known as the **moment of inertia tensor**, and acts as a rotational analogue to mass. In this manner, equation 5.27 is the rotational analogue of the equation $\vec{p} = m\vec{v}$ for linear momentum. Like mass, it is the proportionality between a momentum and a velocity, however it is a 3×3 symmetric matrix. This reflects the anisotropic nature of rotation, where motion about different axes may not be considered equal. For example, a long cylinder such as a broom has a larger angular momentum when rotating perpendicular to its length than it does rotating along its length at the same angular velocity.

Linear momentum is expressed as $\vec{p}_i = m_i \vec{v}_i$, and its time derivative yields Newton's second law (assuming constant mass):

$$\vec{F}_i = \frac{\mathrm{d}\vec{p}_i}{\mathrm{d}t} = m_i \vec{a}_i \tag{5.28}$$

As there are two ways of expressing the angular momentum (Equations 5.25 and 5.27), there are two expressions for the time derivative of angular momentum (referred to as the

torque $\vec{\tau}$):

$$\vec{\tau} = \sum_i \vec{r}_i \times \vec{F}_i \tag{5.29}$$

$$= \mathbf{I}\vec{\alpha} + \frac{\mathrm{d}\mathbf{I}}{\mathrm{d}t}\vec{\omega} \tag{5.30}$$

The first of these equations expresses the torque as a sum over the individual torques on each particle (where the cross product of $\vec{r}_i \times \vec{F}_i$ represents the component of the force on the particle that would cause a rotation). This expression illustrates the classical result that applying a force at a point further away from the centre of mass causes a larger torque to be applied. The second equation expresses it in terms of the overall motion of the set of particles, namely the angular velocity and its time derivative, the **angular acceleration $\vec{\alpha}$**. Note that unlike mass, we cannot assume that the moment of inertia tensor is constant. Even for a rigid body, as an object rotates, its moment of inertia tensor may change.

All these expressions have dealt with a finite set of discrete point-like atoms. However, they can be generalised for a continuous distribution of mass, which could now be seen as an infinite distribution of infinitesimal point masses. In this situation, our rigid body of many particles now acts as a single particle, and we no longer discuss its constituent parts. The more general expressions for angular momenta and torques that involve the moment of inertia still hold, and the moment of inertia is now treated as an intrinsic property of the particle, in a similar manner to its mass.

An important point that will be relevant later is that whilst we have a well-defined position for each atom (from which velocity, acceleration, momentum and force may be derived), we do not have a unique way to choose the orientation of a set of points. However, we still can calculate the rotational analogues of these other quantities (respectively, the angular velocity, angular acceleration, angular momentum and torque).

Definition of moving coordinate frame

The previous section discusses the kinematics of rigid bodies — discrete objects with no internal motion. As discussed previously, whilst rigid bodies can be incorporated into molecular dynamics, they require specific changes to the equations of motion. They also limit the internal degrees of freedom which may be important in the current task. To describe how to apply rotational interactive forces to molecules, we must first generalise the mathematics for rigid bodies to general sets of particles.

The centre of mass still gives a good well-defined reference point which describes the translational motion of a molecule over the simulation. By looking at its time derivatives, it also allows us to define exactly what is meant by the position, velocity and acceleration of the molecule as a whole. Describing the rotational motion of the molecule however is not as trivial. The problem of tracking the rotation of a molecule can be considered equivalent to assigning a coordinate frame (consisting of an origin and a right-hand triple of orthonormal axes) to the molecule at each point of the trajectory. By describing the motion of this coordinate frame, we therefore are describing the translation and rotation of the molecule. This is similar to the challenge of curve framing discussed in the previous chapter.

Given two consecutive timesteps, the translation and rotation between the two can be considered from two perspectives: as an **active** transformation, in which the molecule itself at the later step is translated and rotated to align it with the molecule at the previous step; and as a **passive** transformation, in which a coordinate system centred on the molecule at an earlier step is transformed to align with the molecule at a subsequent step.[249] This first interpretation of aligning two molecules is a common task in crystallography and cheminformatics,[250] usually performed using the Kabsch algorithm.[251,252] Here, the algorithm translates the molecules to share the same centre of mass, and determines a rotation matrix $\mathbf{C}$ that minimises the mass-weighted root-mean-square-deviation (RMSD) between the two conformations a and b:

$$\sum_i m_i \left\| \mathbf{C}\vec{r}_i^{\,a} - \vec{r}_i^{\,b} \right\|^2 \tag{5.31}$$

This rotation matrix $\mathbf{C}$ describes how to rotate the molecule in conformation a to align it best with conformation b. Therefore, by considering a and b to be subsequent timesteps of a simulation, the Kabsch algorithm allows the calculation of the rotation the molecule has undergone from one timestep to the next, by finding the value of $\mathbf{C}$ that minimises:

$$\sum_i m_i \|\mathbf{C}\vec{r}_i(t) - \vec{r}_i(t+\delta t)\|^2 \tag{5.32}$$

From here, we note that the matrix $\mathbf{C}$ represents a rotation from an initial orientation $\mathbf{R}(t)$ to a subsequent orientation $\mathbf{R}(t+\delta t)$, where both orientations are represented as 3×3 rotation matrices. We can hence represent this as:

$$\mathbf{C} = \mathbf{R}(t+\delta t)\mathbf{R}(t)^{-1} \tag{5.33}$$

In the limit of small timestep, we may Taylor expand this to first order:

$$\mathbf{C} = \big(\mathbf{R}(t) + [\vec{\omega}]\mathbf{R}(t) + \mathcal{O}(\delta t^2)\big)\mathbf{R}(t)^{-1} = \mathbf{1} + [\vec{\omega}]\delta t + \mathcal{O}(\delta t^2) \tag{5.34}$$

The $\vec{\omega}$ arises from the time-derivative of the rotation matrix, and represents the angular velocity of the orientation at time t.

Inserting this back into the Kabsch algorithm, we arrive at a new minimisation problem that requires finding the angular velocity $\vec{\omega}$ which minimises the expression:

$$\sum_i m_i |\vec{v}_i - \vec{\omega} \times \vec{r}_i|^2 \tag{5.35}$$

The problem has now been reduced to a minimisation problem defined not in terms of the rotation matrix $\mathbf{C}$, but in terms of the rate of change of the orientation $\vec{\omega}$. By taking the gradient with respect to $\vec{\omega}$, we can determine the value of $\vec{\omega}$ that minimises this expression:

$$\vec{\omega} = \left(- \sum_i m_i [\vec{r}_i]^2 \right)^{-1} \left(\sum_i m_i \vec{r}_i \times \vec{v}_i \right) = \mathbf{I}^{-1} \vec{L} \tag{5.36}$$

The resultant expression is exactly that for a rigid body, with the same expressions for the momentum of inertia tensor and for the angular momentum. However, we note that at no point have we said these atoms are rigidly bound — we have simply determined what angular velocity captures the most motion when using RMSD. Therefore, equation 5.36 describes both a rigid set of points, but also the angular velocity which minimises equation 5.35 for a non-rigid set of points

It is interesting to note that equation 5.35 is exactly the kinetic energy of the atoms, ignoring the motion due to the overall translation and rotation of the molecule. This minimisation hence finds the angular velocity $\vec{\omega}$ that best describes the motion of the particles by capturing the most kinetic energy.

In the field of spectroscopy, the Eckart frame is a coordinate frame for a molecule that minimises the coupling between rotational and vibrational motion. It has been shown that this problem is closely related to the rotational superposition problem solved by the Kabsch algorithm.[253,254] A detailed explanation for their relation can be found in Chevrot et al.[255].

Dynamics of the coordinate frame

With this, we now have a way of describing the position, velocity and acceleration of a molecule (through the motion of its centre of mass), as well as its angular velocity and acceleration (by using the same expressions as for a rigid collection of atoms). The evolution of the molecule through time is influenced by the forces applied to each atom, denoted $\left\{\vec{F}_i\right\}$. As the moving coordinate frame follows the molecule, these forces influence the motion of the coordinate frame. We will now stop considering the coordinate frame itself, and talk about the motion of the molecule as a whole. We hence consider $\vec{\omega}$ (as defined in equation 5.36) to be not just the

Force	Translational	Rotational	Internal
$\vec{F}_i = m_i \vec{G}$	$\vec{G}$	$\vec{0}$	$\vec{0}$
$\vec{F}_i = m_i \vec{H} \times \vec{r}_i$	$\vec{0}$	$\vec{H}$	$\vec{0}$

Table 5.1: Various accelerations caused by the two forces described in equations 5.41 and 5.42. From left to right, the accelerations describe the translational acceleration applied to the molecule as a whole, rotational acceleration applied to the molecule as a whole, and the acceleration as it appears in the moving coordinate frame that follows the molecule.

angular velocity of the coordinate frame that follows the molecule, but the angular velocity of the molecule itself.

The acceleration of the molecule is equal to the acceleration of the centre of mass, which is given by a sum over the individual forces:

$$\vec{A} = \frac{1}{M} \sum_i \vec{F}_i \tag{5.37}$$

Here we note that generally, all internal forces in the molecule due to the forcefield cancel out due to Newton's third law. Therefore, the molecule will only accelerate when external forces are applied to it.

Likewise, the angular acceleration of the molecule is given by the time derivation of equation 5.27:

$$\vec{\alpha} = \left(\frac{\mathrm{d}\mathbf{I}}{\mathrm{d}t}\right)^{-1} \vec{L} + \mathbf{I}^{-1} \left(\sum_i \vec{r}_i \times \vec{F}_i \right) \tag{5.38}$$

As both equations 5.37 and 5.38 are linear in $\vec{F}_i$, if an external force $\vec{F}_i^{\text{add}}$, such as that used in IMD, is applied to the system, the additional translational and rotational accelerations caused by this force are given by:

$$\vec{A}^{\text{add}} = \frac{1}{M} \sum_i \vec{F}_i^{\text{add}} \tag{5.39}$$

$$\vec{\alpha}^{\text{add}} = \mathbf{I}^{-1} \left(\sum_i \vec{r}_i \times \vec{F}_i^{\text{add}} \right) \tag{5.40}$$

The existing class of forces as applied in current IMD implementations all take the form of some mass-weighted constant vector:

$$\vec{F}_i^{\text{add}} = m_i \vec{G} \tag{5.41}$$

Using equations 5.39 and 5.40, forces of the form given in equation 5.41 cause an acceleration of $\vec{G}$ whilst not causing any angular acceleration.

I propose a second class of interactive forces, which cause explicit rotational motion of the whole molecule and with the general form:

$$\vec{F}_i^{\text{add}} = m_i \vec{H} \times \vec{r}_i \tag{5.42}$$

Again, using equations 5.39 and 5.40, forces of this form cause an angular acceleration of $\vec{H}$ whilst not causing any translational acceleration. This is summarised in Table 5.1.

Internal motion of the molecule

In addition to causing overall translations and rotations of the molecule, forces may also influence its internal motion. Given our moving coordinate frame, we can consider the atomic positions (relative to the centre of mass) in three different frames — the global coordinate frame (denoted $\vec{R}_i$) relative to the origin; the centre-of-mass frame (denoted $\vec{r}_i$); or the moving coordinate frame (denoted $\vec{r}_i^M$), which is both centred at the centre of mass and also rotating. The latter two are related by the current rotational matrix $\mathbf{R}$ that defines the rotation of the moving coordinate frame:

$$\vec{r}_i^M = \mathbf{R}^{-1}\vec{r}_i \tag{5.43}$$

The velocity of each particle in the moving frame is given by the time derivative of equation 5.43, noting that $\mathbf{R}$ also has a time dependence:

$$\vec{v}_i^M = \mathbf{R}^{-1}(\vec{v}_i - \vec{\omega} \times \vec{r}_i) \tag{5.44}$$

The second term that appears in equation 5.44 removes the component of the velocity that is captured by the rotational velocity of the coordinate frame.[256]

The acceleration of each particle as observed in the moving frame is given by the time derivative of equation 5.44, giving:

$$\vec{a}_i^M = \mathbf{R}^{-1}(\vec{a}_i - \vec{\alpha} \times \vec{r}_i - 2\vec{\omega} \times \vec{v}_i - \vec{\omega} \times (\vec{\omega} \times \vec{r}_i)) \tag{5.45}$$

The above equation illustrates the three additional accelerations (and hence forces) that a particle experiences as a consequence of rotational motion. These are the Euler force, Coriolis force and centrifugal force respectively.[256] Forces which appear in a rotating (also referred to as non-inertial) frame of reference are commonly described as fictitious or pseudo forces.

As was the case for equations 5.37 and 5.38, the acceleration can be decomposed into terms associated with each force applied to the system. The magnitude of the acceleration caused on each atom in the rotating frame due to some additional external force $\vec{F}_i^{\text{add}}$ can therefore be expressed as:

$$\left\lVert \vec{a}_i^{M,\text{add}} \right\rVert = \left\lVert \frac{\vec{F}_i^{\text{add}}}{m_i} - \vec{A}^{\text{add}} - \vec{\alpha}^{\text{add}} \times \vec{r}_i \right\rVert \tag{5.46}$$

Equation 5.46 describes the magnitude of the acceleration applied to an individual atom by an external force, as observed in the molecule-centered rotating frame of reference. It captures any 'left-over' force that does not cause an overall translation or rotation of the entire molecule. It can be confirmed that both proposed forces (equations 5.41 and 5.42) result in no additional acceleration occurring in the internal frame of reference, and hence these forces do not directly perturb the internal structure of the molecule. This can be contrasted with the existing approach of applying forces to only certain particles near the outside of the molecule. Here, this force will cause accelerations within the rotating frame of reference, and the molecule will deform.

We have now established that a good basis for a rotating and translating force should have the form:

$$\vec{F}_i = m_i\vec{G} + m_i\vec{H} \times \vec{r}_i \tag{5.47}$$

Any force of this form will cause a translational acceleration $\vec{G}$ and an angular acceleration $\vec{H}$, whilst not causing accelerations within the internal frame of the molecule. However, we must now establish how the user may control $\vec{G}$ and $\vec{H}$ from within virtual reality.

Driving Interactions with Virtual Reality

In interactive molecular dynamics, the approach to interactions involves specifying an interaction site, which acts as a target position or goal. Here, the interactive force pulls the target molecule towards this pivot point which is controlled by the user, using devices such as haptic feedback devices or virtual reality controllers. This force depends on the distance between the current centre of mass and the pivot point. For this, we will consider the harmonic potential as described in equation 5.16. Here, we have some variable $\xi(t)$ (the centre of mass) and a target value $\xi_0(t)$ (the interaction point). The harmonic potential with force constant k has a potential energy given by:

$$U(t) = \frac{k}{2}(\xi(t) - \xi_0(t))^2 \tag{5.48}$$

This equation clearly frames the problem as based on the difference between a desired target value and a current value. However, when considering rotations, we wish to avoid having to define an exact orientation of the molecule. This can be avoided by rephrasing equation 5.48 to be in terms of different variables: the desired overall change $\Delta_0\xi(t)$, and the change that has happened so far $\Delta\xi(t)$. Equation 5.48 now takes the form:

$$U(t) = \frac{k}{2}(\Delta_0\xi(t) - \Delta\xi(t))^2 \tag{5.49}$$

The reason to perform this change is that $\Delta\xi(t)$ may be calculated based upon its time derivative:

$$\Delta\xi(t) \equiv \int_{t_0}^{t} \xi'(t)\ \mathrm{d}t \tag{5.50}$$

This simple shift means that instead of measuring an exact state of our system, we need only be able to describe how much it has changed so far and how much we wish it to change. This is ideal for rotational motion, as we can define the angular velocity. By accumulating the angular velocity over the course of the interaction, we can measure the total rotation the molecule has undergone, without having to define the orientation of the molecule at any point. By comparing this to the amount of rotation the controller has performed during the interaction, we know how much rotation still needs to be applied for the two to match.

When discussing rotations, it is important to note that there are several representations, each with benefits and drawbacks:

1. Any rotation may be represented by a 3×3 **rotation matrix**. These matrices are the members of the special orthogonal group $SO(3)$, which is the group of all orthogonal matrices (with inverses equal to their transposes) with determinant 1.

2. The **axis-angle** or rotation vector representation stores a rotation as a vector in $\mathbb{R}^3$, whose direction indicates the rotation axis and whose magnitude indicates the magnitude. They are useful for storing angular velocities, and have the property that multiplication by a scalar k scales the rotation by that amount. Unlike rotation matrices or quaterions, they can represent rotations of greater than 360 degrees.

3. **Euler angles** represent a rotation as a vector, where each component is a rotation about one of the primary axes. The order in which these rotations are performed depends on the context. These are often encountered as the concept of yaw, pitch and roll in aviation.

4. Finally, **quaternions**, which can be viewed as a higher-order version of complex numbers (with one real part and three imaginary parts), can be employed as rotations. Appendix C goes into more detail on quaternions.

For example, within Unity, rotations are internally represented by quaternions, whilst being presented to the user in the user interface as Euler angles. Generally, we use quaternions to represent rotations and rotation vectors to represent angular velocities.

With this, we are equipped to define our rigid motion force. The controller has initial position and rotation $\vec{T}_{\text{controller}}(t_0)$ and $\mathbf{q}_{\text{controller}}(t_0)$, where t_0 is the time of the interaction starting and $\mathbf{q}$ is a quaternion. At some later point t, the controller may have moved to $\vec{T}_{\text{controller}}(t)$ and $\mathbf{q}_{\text{controller}}(t)$. From this, we can define the overall translation and rotation we wish to apply to the molecule as:

$$\vec{T}_{\text{desired}}(t) \equiv \vec{T}_{\text{controller}}(t) - \vec{T}_{\text{controller}}(t_0) \tag{5.51}$$

$$\mathbf{q}_{\text{desired}}(t) \equiv \mathbf{q}_{\text{controller}}(t)\mathbf{q}_{\text{controller}}(t_0)^{-1} \tag{5.52}$$

Note that while translations are compared using addition and subtraction, rotations (represented as quanterions) are multiplied.

We also have accumulated the total translation and rotation of the molecule *since the interaction has begun*:

$$\vec{T}_{\text{accum}}(t_0) = \vec{0} \tag{5.53}$$

$$\vec{T}_{\text{accum}}(t + \delta t) = \vec{T}_{\text{accum}}(t) + \frac{\vec{v}(t) + \vec{v}(t + \delta t)}{2}\delta t \tag{5.54}$$

$$\mathbf{q}_{\text{accum}}(t_0) = \mathbf{0} \tag{5.55}$$

$$\mathbf{q}_{\text{accum}}(t + \delta t) = \text{quat}\left(\frac{\vec{\omega}(t) + \vec{\omega}(t + \delta t)}{2}\delta t\right)\mathbf{q}_{\text{accum}}(t) \tag{5.56}$$

Here, quat converts from the rotation vector representation used for angular velocities to a quaternion representation.

From this, we can calculate the remaining translation and rotation that must be applied such that the molecule will have undergone the same motion as that of the controller:

$$\vec{T}_{\text{remaining}}(t) = \vec{T}_{\text{desired}}(t) - \vec{T}_{\text{accum}}(t) \tag{5.57}$$

$$\vec{\theta}_{\text{remaining}}(t) = \text{rot}(\mathbf{q}_{\text{desired}}(t)\mathbf{q}_{\text{accum}}(t)^{-1}) \tag{5.58}$$

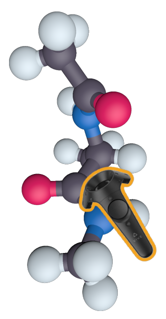
Note that now we use rot to convert a quaternion back to the rotation vector representation, which is more suited for defining torques.

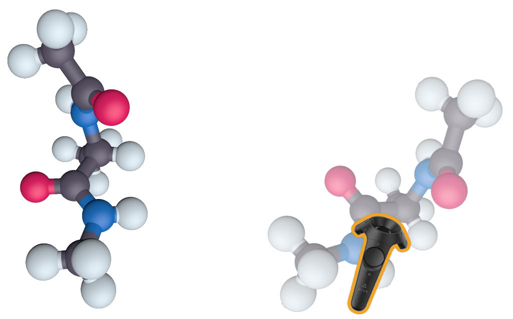
The force for this interaction will consist of two terms, one linear in the remaining translation and one linear in the remaining rotation:

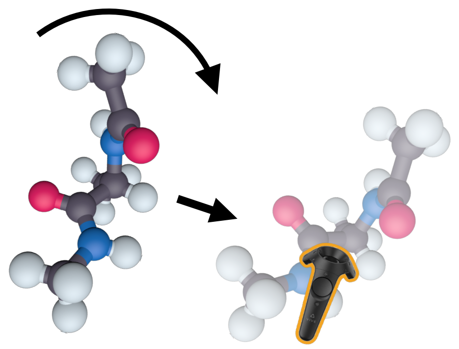
$$\vec{F} = \frac{m_i}{M}k\vec{T}_{\text{remaining}} + \frac{m_i}{M}k\vec{\theta}_{\text{remaining}} \times \vec{r}_i \tag{5.59}$$

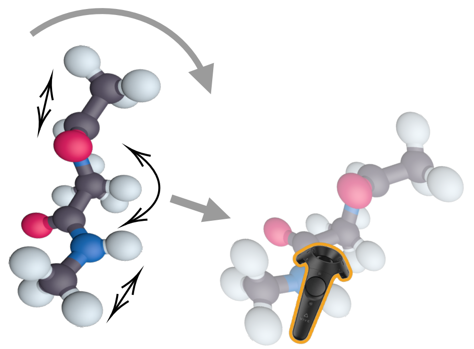
The first part of this force is exactly the same as the existing harmonic interaction widely used in IMD. The second part is the corresponding angular term, applying forces which rotate each particle about the centre of mass, proportional to its distance from the axis of rotation.

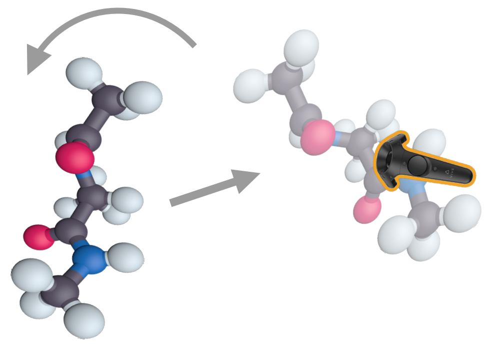
The setup of this force allows a user to use all six degrees of freedom available to them when performing an interaction. Figure 5.8 illustrates how the position and rotation of the VR controller is used to drive the interactive forcefield.


(a) The user highlights the fragment they wish to interact with and pull the trigger, starting the interaction.


(b) As the user moves and rotates their controller in 3D space, a translucent copy of the molecule is drawn to indicate the rigid motion about to be applied.


(c) The interaction forces translate and rotates the molecule towards the target position and orientation without perturbing the internal structure directly.


(d) Internal motion is still allowed, whether caused by internal vibrations or other nearby molecules. These changes are reflected in real time in the translucent target molecule.


(e) During the interaction, the user is free to continue moving and rotating their controller. This updates the target continuously.

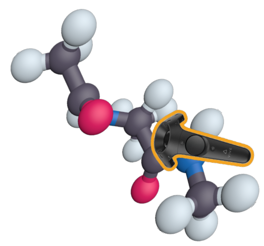

(f) The molecule will stop when it reaches the target due to the damping forces applied. The user can now release the trigger.

Figure 5.8: Steps involved in applying the rigid motion interaction in virtual reality.

Rotations and Centripetal Force

We now have a basic definition of the force we wish to use. Given some distance $\vec{T}$ to translate and rotation vector $\vec{\theta}$ to rotate through, the force on the i-th atom will be given by:

$$\vec{F}_i = \frac{m_i}{M} k(\vec{T} + \vec{\theta} \times \vec{r}_i) \tag{5.60}$$

To compare this to the existing approach to rotation, we will use the example system of alanine dipeptide, as provided in the `openmmtools` test systems. This system consists of two alanines connected by a peptide bond, and contains 22 atoms. The forcefield is AMBER ff96 and the molecule is in OBC GBSA implicit solvent. For integration, we use a Velocity-verlet scheme, with a timestep of 0.5 fs. No thermostat is used, as the aim of the simulations are to illustrate the kinetic energy placed into the system.

Figure 5.9 illustrates four methods to rotate a molecule. First, Figures 5.9a and 5.9b illustrate how rotations would be performed in the existing IMD-VR paradigm, by applying interactive forces to single atoms. As already stated, this approach causes deformation of the molecule by relying on the internal forces within the molecule to cause it to rotate to follow the manipulated atoms.

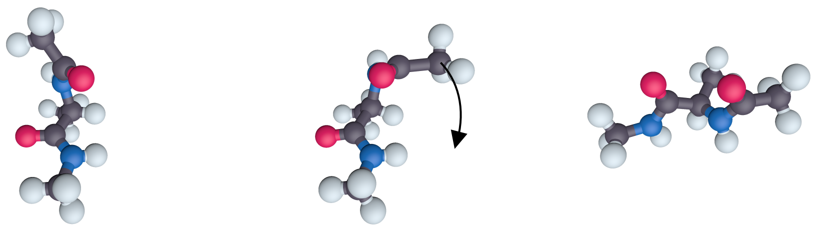
Applying Equation 5.60 gives us a rotation that appears in Figure 5.9c. This illustrates that applying a pure rotational force causes the atoms to spiral outwards. To see why this is, consider at some point t the molecule is moving as a rigid body in unison. At this point, the relative velocity $\vec{v}_i$ of each particle is equal to $\vec{\omega} \times \vec{r}_i$. If we wish to maintain the shape of the molecule, we wish that all internal angles and distances are unchanged. We can express this as conservation of all dot products of the form $\vec{r}_i \cdot \vec{r}_j$. First, we can confirm that the rate of change of these dot products is 0 if we assume our molecule is acting as a rigid body:

$$\begin{aligned}\frac{\mathrm{d}\vec{r}_i \cdot \vec{r}_j}{\mathrm{d}t} &= \vec{v}_i \cdot \vec{r}_j + \vec{r}_i \cdot \vec{v}_j \\ &= \vec{r}_j \cdot (\vec{\omega} \times \vec{r}_i) + \vec{r}_i \cdot (\vec{\omega} \times \vec{r}_j) \\ &= 0\end{aligned}$$

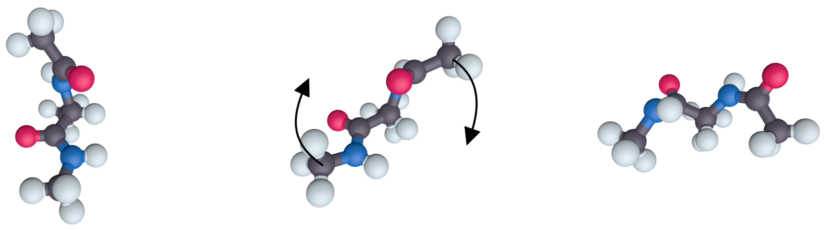
The next step is to ensure that the second derivative of this value is also 0:

$$\begin{aligned}\frac{\mathrm{d}^2\vec{r}_i \cdot \vec{r}_j}{\mathrm{d}t^2} &= \vec{a}_i \cdot \vec{r}_j + \vec{r}_i \cdot \vec{a}_j + 2\vec{v}_i \cdot \vec{v}_j \\ &= \vec{a}_i \cdot \vec{r}_j + \vec{r}_i \cdot \vec{a}_j + 2(\vec{\omega} \times \vec{r}_i) \cdot (\vec{\omega} \times \vec{r}_j) \\ &= (\vec{a}_i - \vec{\omega} \times (\vec{\omega} \times \vec{r}_i)) \cdot \vec{r}_j + \vec{r}_i \cdot (\vec{a}_j - \vec{\omega} \times (\vec{\omega} \times \vec{r}_j))\end{aligned}$$

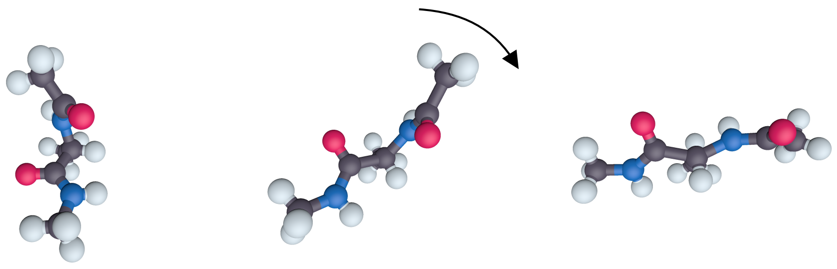
The highlighted terms are exactly the centripetal acceleration experienced by the particles, causing them to spiral outwards. This can be addressed by adding a centripetal force inwards, that pulls the atoms towards the axis of rotation.



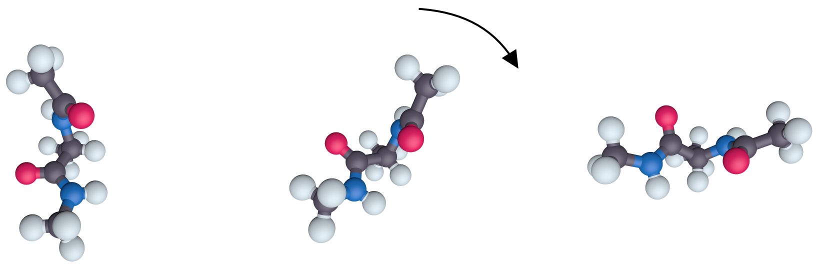
(a) Rotating the molecule by applying a force on a single atom. This causes a large deformation in the molecule.



(b) Rotating the molecule by applying a force on two atoms. Deformation still occurs, but is less of an issue than when a single atom is moved.



(c) Rotating the molecule by applying a force of Equation 5.60. Whilst each atom is rotated individually, the angular velocity causes the molecule to be stretched.



(d) Rotating the molecule by applying a force of Equation 5.3, which includes a centripetal term. There is now no deformation of the molecule.

Figure 5.9: Comparison of various methods to rotate an alanine dipeptide molecule.

Including this term, our force is:

$$\vec{F_i} = \frac{m_i}{M} k(\vec{T} + \vec{\theta} \times \vec{r_i}) + m_i \vec{\omega} \times (\vec{\omega} \times \vec{r_i})$$

Figure 5.9d shows how including this term prevents the molecule from being stretched as it is rotated. This allows large values of k to be used, ensuring a fast rotation to the desired orientation, without distorting the molecule.

In order to better quantify what is meant by 'deformation', consider the kinetic energy of the molecule. This can be split into three terms — a translational kinetic energy due to the overall velocity of the molecule; a rotational kinetic energy due to the overall rotational motion; and the remaining kinetic energy, which can be thought of as the 'internal' kinetic energy of vibration.

The internal kinetic energy is measured by removing from the total kinetic energy the kinetic energy associated with overall translations and rotations of the molecule:

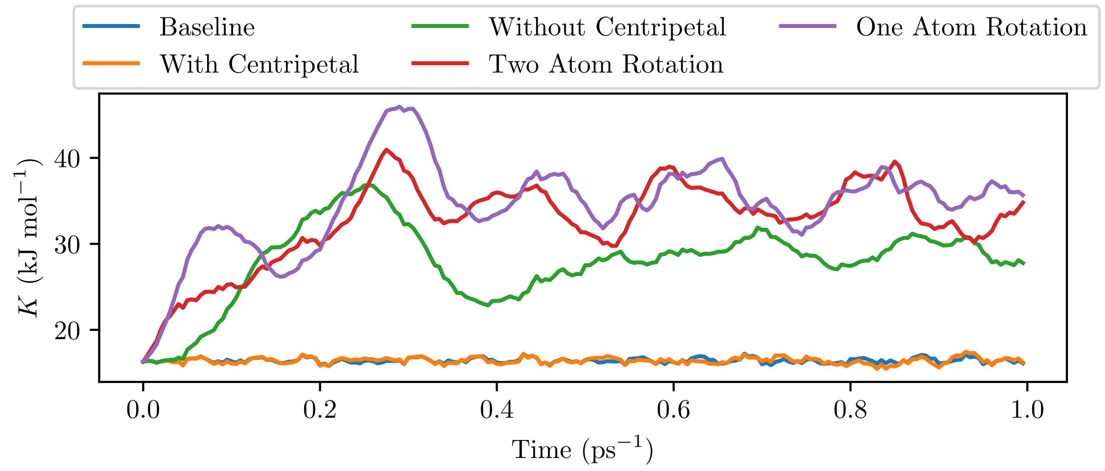
$$K_i^{\text{int}} = \frac{1}{2} m_i V_i^2 - \frac{1}{2} M V_{\text{COM}}^2 - \frac{1}{2} \vec{\omega}^T \mathbf{I} \vec{\omega} \tag{5.61}$$

In a standard NVT simulation, the thermostat will cause the internal kinetic energy to converge to an appropriate value for the given temperature. However, the sudden influx of internal kinetic energy that can occur when rotating a molecule can cause unwanted conformational changes before it is reduced by the thermostat.

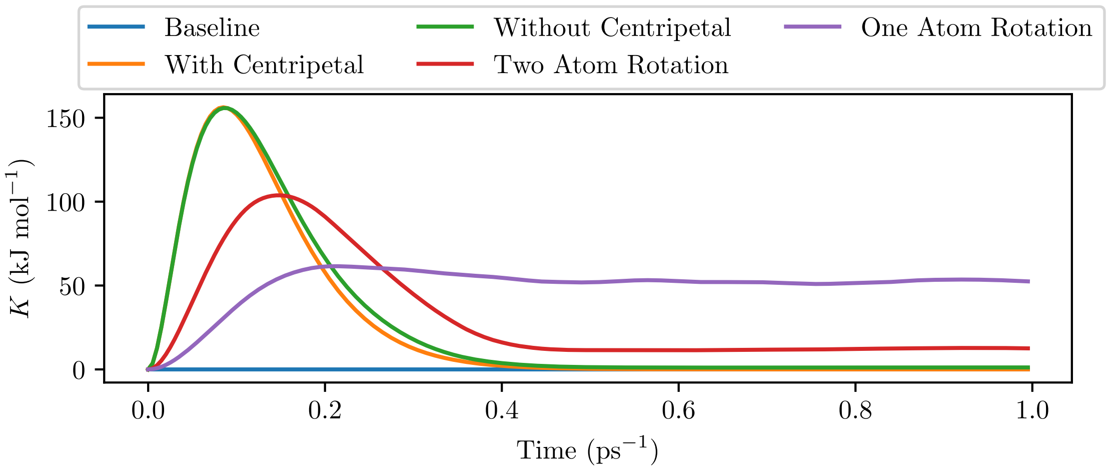
Figure 5.10 shows the internal, rotational and translational kinetic energies for the three approaches. First, it can be seen that the one-atom rotation performs poorly, both increasing the internal kinetic energy whilst also imbuing the molecule with rotational and translational kinetic energy that lasts even after the rotation is complete. These issues are similarly observed in the two-atom rotation.

It is clear that my rigid motion interactions do not cause any translation when only a rotation is required. This is one of the main aims of the interactive force. It is also notable that after the rotation is complete the rigid motion force reduces the rotational kinetic energy back towards 0. The internal kinetic energy graph (Figure 5.10a) shows the importance of the centripetal term in the rigid motion force. With the centripetal term, the internal kinetic energy is identical to that of the baseline (where no rotation is applied).

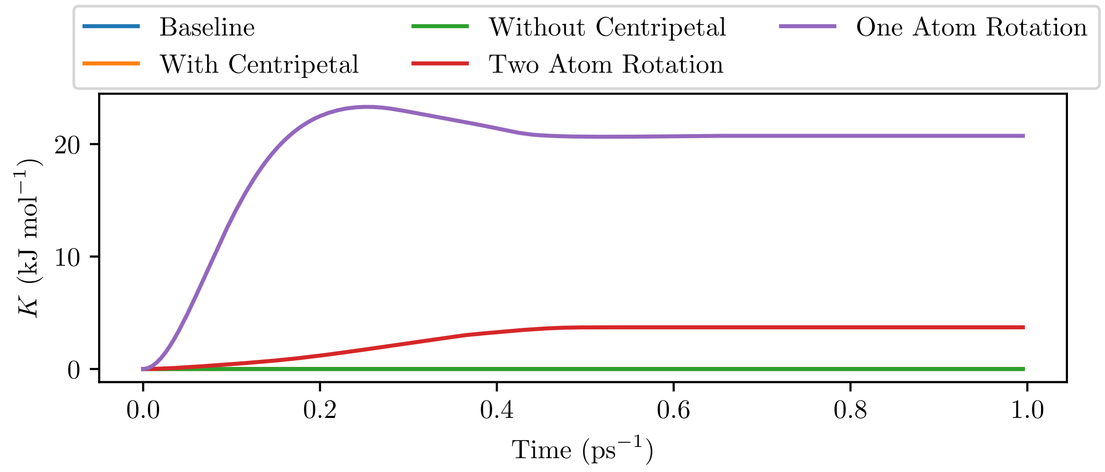
These graphs clearly show that the rigid motion force as defined in Equation 5.3 rotates a molecule in the ideal way — it leads to neither additional internal motion or any translations, with all the interaction energy going into the rotational kinetic energy. However, unlike treating the whole molecule as a rigid body, the molecule still vibrates with a similar kinetic energy to when no interaction is applied.



(a) Internal kinetic energy for each of the approaches of Figure 5.9



(b) Rotational kinetic energy for each of the approaches of Figure 5.9



(c) Translational kinetic energy for each of the approaches of Figure 5.9

Figure 5.10: Breakdown of kinetic energies for the rotation of alanine dipeptide, for the four methods of rotation shown in Figure 5.9

Damping

As currently applied in existing IMD implementations,[23,29] the harmonic potential used to apply interactions can lead to overshooting of the target position. This problem can be avoided by the use of a damped spring, which applies an additional dissipative force that is proportional and opposed to the motion of the object in question. For a spring constant k and damping constant γ, the equation of motion for an object displaced from its equilibrium position by some vector $\vec{x}$ is given by:

$$m\frac{\mathrm{d}^2\vec{x}(t)}{\mathrm{d}t^2} = -k\vec{x}(t) - \gamma\frac{\mathrm{d}\vec{x}(t)}{\mathrm{d}t}$$

By varying the value of γ, there are four regimes that can arise:

1. **undamped** ($\gamma = 0$) causes pure oscillatory motion around the target position.
2. **underdamped** ($\gamma < \gamma_C$) involves oscillatory motion about the target position that slowly decreases exponentially.
3. **critically damped** ($\gamma = \gamma_C$) is the point where there are no oscillations, and the system converges to the target point as fast as possible.
4. **overdamped** ($\gamma > \gamma_C$) involves exponential decay towards the target position, with no oscillations. As γ increases, the time taken to get close to the target position increases.

The value of γ at which critical damping occurs is a function of both the strength of the spring and the mass of the object:

$$\vec{\gamma}_C = 2\sqrt{km}$$

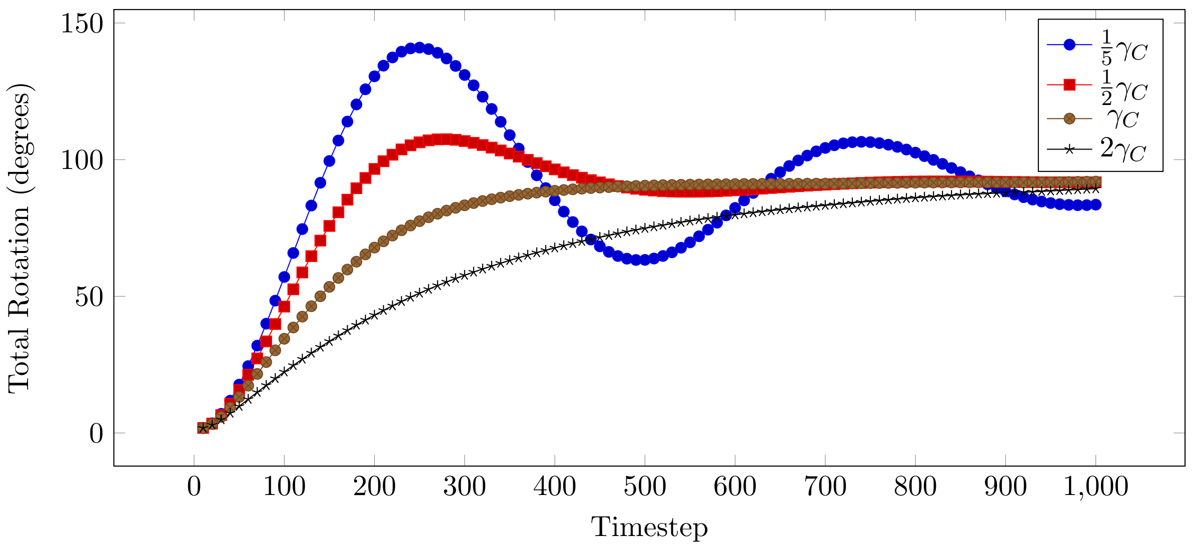


Figure 5.11: Effect of various damping coefficients on the rotation of an alanine dipeptide by 90°. Damping constants under the critical damping constant γ_C have oscillations, whilst damping constants larger than γ_C take longer to reach the target rotation.

This is the value to use for a damping constant that will prevent overshooting of the desired target. Figure 5.11 illustrates the effect that different values of γ have on the rotation of alanine dipeptide, showing that critical damping is the ideal choice for balancing the speed to the interaction whilst preventing oscillations.

The damping terms in the force are similar to those in the equation for Langevin dynamics, which is characterised by an additional damping constant (from here called γ_L). The difference between the two is that the friction as it appears in the interactive force only damps collective motion (either overall translation or rotational velocity), whilst Langevin friction damps all velocities. Using this force in a Langevin system yields an overdamped interaction, as there is damping from both the force and the Langevin thermostat. This can be addressed by modifying the damping constant γ to account for the existing Langevin thermostat, whilst ensuring that it never falls below 0:

$$\gamma = \max(2\sqrt{km} - \gamma_L, 0)$$

In our force, we require damping for both the translational and rotational motion. This damping forces will be proportional to the velocity of the centre of mass and the angular velocity respectively, giving:

$$\vec{F}_i = \frac{m_i}{M}\left[k(\vec{T} + \vec{\theta} \times \vec{r}_i) - \gamma(\vec{V} + \vec{\omega} \times \vec{r}_i)\right] + m_i\vec{\omega} \times (\vec{\omega} \times \vec{r}_i) \tag{5.62}$$

This is the full equation for rigid motion interactions. Figure 5.12 illustrates graphically how these five terms combine to affect a molecule.

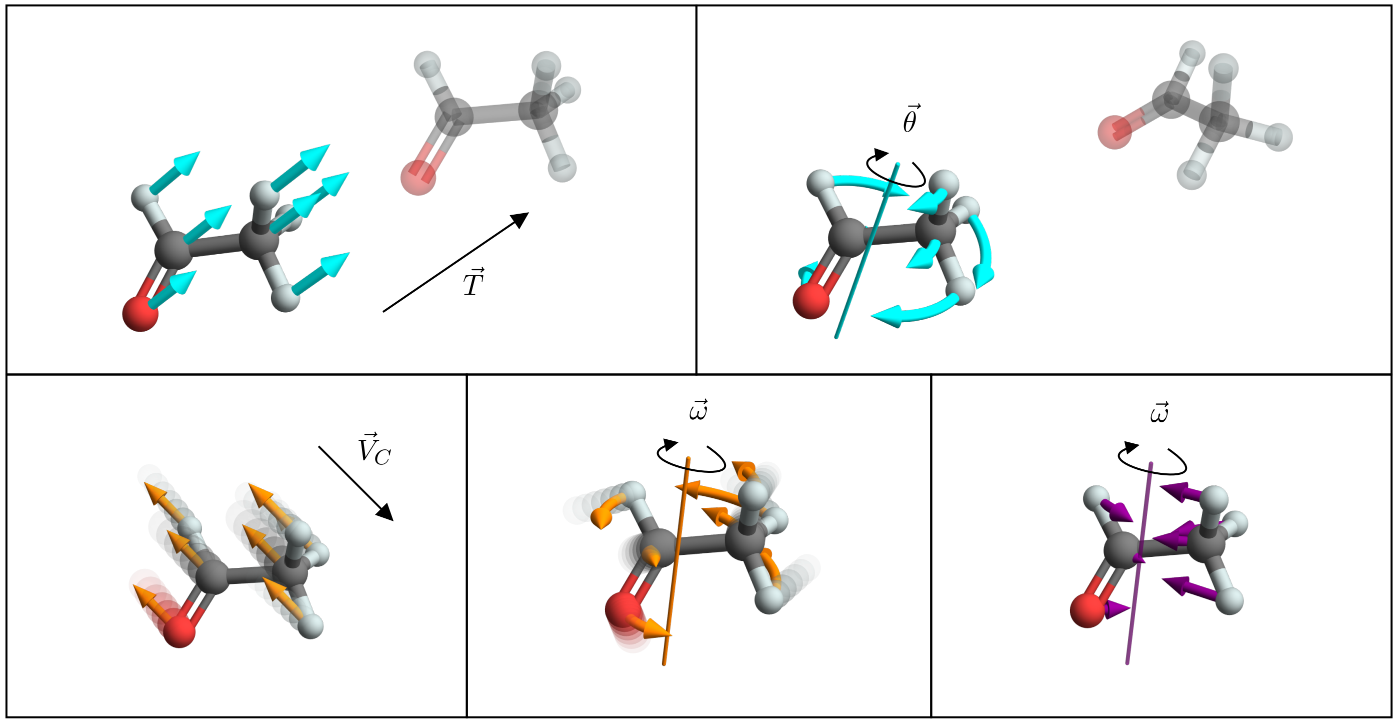


Figure 5.12: Breakdown of the total force into its five components: (A) the **translational force** (blue) which pulls the molecule along the desired translation $\Delta\vec{T}$; (B) the **rotational force** (blue) which rotates the molecule around the desired rotation axis $\Delta\vec{\theta}$; (C) the **translational drag** (orange) which opposes the overall velocity $\vec{V}_C$ of the molecule; (D) the **rotational drag** (orange) which opposes the overall angular velocity $\vec{\omega}$ of the molecule and (E) the **centrifugal force** (purple) which pulls the atom towards the axis about which the molecule is rotating.

5.4 Finite and Asymmetric Particles

Up until this point, we have only considered point masses. A point mass has a mass distribution of a Dirac delta function, with all its mass concentrated at one single point in space. We can extend this concept by considering particles that can be considered as objects with some distribution of mass. If we treat these particles as solid and undeformable, then they are an example of a rigid body.

Asymmetric and finite-sized particles can be found in coarse-grained forcefields such as oxDNA[257] and UNRES[258]. Unlike the point particles discussed previously, these particles have orientations, moments of inertias and angular momenta about their centre of mass. Because of this, modifications have to be made to the rotational forces to ensure they rotate correctly. As interactive simulations are limited by the amount of time a user can be immersed in a simulation, coarse-grained simulations are better suited for IMD-VR.[70]

Potentials which rely on particle orientation include the Gay-Berne[259], Everaers[260] and Kern-Frenkel[261]. The LAMMPS package supports aspherical particles, including the Gay-Berne pairwise force and running NVT, NPT and NVE molecular dynamics incorporating orientations, angular momenta and torques. These potentials have been used to address problems involving liquid crystals[262] and nanoassembly[263]. One approach to speed up molecular simulations is to apply rigid constraints on certain bonds and angles. Usually, these restraints are implemented using modifications to the integration, including SHAKE and RATTLE. However, another approach to enforcing these rigid constraints is to replace each set of rigidly connected bodies by a single composite particle. The interactions between these composite particles are calculated by the summing over all interactions between their components.

The LAMMPS package has several packages well-suited for performing molecular dynamics with asymmetric particles.[264] One such model supported by LAMMPS is the oxDNA model, a coarse-grained model where each nucleotide is represented as a single oriented bead.[265] The orientated nature of this force field is important, as the bead represents a whole nucleotide with its base and sugar backbone.

Coarse-grained techniques often abstract multiple point masses of the atomistic system into a single point mass. However, this loses the information on orientation, and fails to account for the uneven distribution of mass.[266] This can be addressed by performing rigid-body quaternion dynamics, where the orientations of each particle are propagated in a similar manner to that of the Cartesian positions of the particles.

Quaternion Dynamics

The point masses used in regular molecular dynamics have three translational degrees of freedom, which are propagated using Newton's second law. When the particles also possess orientation, their rotational dynamics must also be propagated.

The moment of inertia tensor, as described for a set of particles previously, is a 3×3 matrix which describes the distribution of mass about a specific point. Each finite-sized particle will have a moment of inertia about its own centre of mass, denoted $\mathbf{I}_i^{\mathrm{intr}}$. If the particle is asymmetric, then its moment of inertia will vary as the particle rotates. As this matrix is a real symmetric matrix, it may be decomposed into a rotation matrix $\mathbf{R}$ and a diagonal inertia matrix Λ_i:

$$\mathbf{I}_i^{\mathrm{intr}}(t) = \mathbf{R}(t)\mathbf{\Lambda}_i\mathbf{R}(t)^{-1} \tag{5.63}$$

For an asymmetric particle, this implies the existence of a natural definition of the orientation of the particle — the body frame of reference in which the moment of inertia tensor is a diagonal matrix $\mathbf{\Lambda}_i$. Assuming our particles are rigid and nondeformable, $\mathbf{\Lambda}_i$ is a constant property of each particle, in a similar manner to its mass. Therefore, for rotational dynamics the orientation $\mathbf{R}$ and the diagonal elements of $\mathbf{\Lambda}_i$ are stored, with the moment of inertia tensor in the laboratory frame calculated using Equation 5.63 when required.

In order to propagate the rotational degrees of freedom for each particle, we require a rotational analogue to Newton's second law. We can obtain this by considering the angular momentum of each particle about its own centre of mass:

$$\vec{L}_i^{\mathrm{intr}} = \mathbf{I}_i^{\mathrm{intr}}\vec{\omega}_i \tag{5.64}$$

Note that point masses have a moment of inertia of 0, and hence cannot possess an angular momenta about their own position. This angular momentum of each particle can be seen as a 'spin' angular momentum, as opposed to the orbital angular momentum of the particle about another point. Taking the time derivative of this yields[a]:

$$\begin{aligned}\frac{\mathrm{d}\vec{L}_i^{\mathrm{intr}}}{\mathrm{d}t} &= [\vec{\omega}_i]\mathbf{R}\mathbf{\Lambda}_i\mathbf{R}^{-1}\vec{\omega}_i + \mathbf{R}\frac{\mathrm{d}\mathbf{\Lambda}_i}{\mathrm{d}t}\mathbf{R}^{-1}\vec{\omega}_i - \mathbf{R}\mathbf{\Lambda}_i\mathbf{R}^{-1}[\vec{\omega}]\vec{\omega}_i + \mathbf{R}\mathbf{\Lambda}_i\mathbf{R}^{-1}\vec{\alpha}_i && (5.65)\\ &= \vec{\omega}_i \times \mathbf{I}_i^{\mathrm{intr}}\vec{\omega}_i + \mathbf{I}_i^{\mathrm{intr}}\vec{\alpha}_i && (5.66)\end{aligned}$$

This time derivative of the angular momentum is equal to the torque $\vec{\tau}_i$ applied to the particle, in the same manner that force is the time derivative of linear momentum. This equation is often referred to as Euler's equation for rigid body dynamics, and can be seen as a rotational analogue for Newton's second law. It contains a term that appears similar to $m\vec{a}_i$ ($\mathbf{I}_i\vec{\alpha}$), but also an additional term. Unlike Newton's second law, this equation is not

[a]The time derivative of a rotation matrix $\mathbf{R}$ is related to the angular velocity $\vec{\omega}$ by $\frac{\mathrm{d}\mathbf{R}}{\mathrm{d}t} = [\vec{\omega}]\mathbf{R}$, where $[\vec{\omega}]$ is the skew-symmetric matrix representing a cross product from the left

easily invertible to calculate the angular acceleration and velocity required to propagate the rotational dynamics through time.

This problem can be addressed by propagating the rotation and angular momentum instead. This yields two equations:

$$\frac{\mathrm{d}\vec{L}_i}{\mathrm{d}t} = \vec{\tau}_i \tag{5.67}$$

$$\frac{\mathrm{d}\mathbf{R}_i}{\mathrm{d}t} = [I_i^{-1}\vec{L}_i]\mathbf{R}_i \tag{5.68}$$

Instead of propagating the full rotation matrices $\mathbf{R}_i$, rotational dynamics is usually performed using another formalism. Early approaches to solving rigid body dynamics numerically used Euler angles, however this suffer from singularities that may lead to numerical instabilities.[267,268] Generally, this is done using a quaternion formalism instead of using rotation matrices. Miller et al.[247] proposed a symplectic method known as the NOSQUISH algorithm for propagating these dynamics using quanterions. This approach is used by HOOMD-blue for NVE calculations.

In the quaternion case, the first-order differential equation relating the angular velocity $\vec{\omega}$ to the quaternion $\mathbf{q}$ is given by:[268]

$$\frac{\mathrm{d}\mathbf{q}_i}{\mathrm{d}t} = \frac{1}{2}\vec{\omega}_i\mathbf{q}_i, \quad \vec{\omega}_i = \mathbf{I}_i^{-1}\vec{L}_i \tag{5.69}$$

In addition to performing NVE calculations, NVT for rotational dynamics can be performed by modifying Langevin dynamics to include the rotational degrees of freedom.[269]

Rotation Interactions

In the case for point masses, we calculate the angular momentum and moment of inertia tensor for the set of particles we interact with, in order to calculate the overall angular velocity. To apply my method of rotating interactions to asymmetric particles, we must consider the additional effects of finite-sized particles to these equations.

The angular momentum of a non-point mass about another point is equal to the sum of its angular momentum about that point, plus the angular momentum it possess itself:

$$\vec{L} = \sum \vec{L}_i, \quad \vec{L}_i = m_i \vec{r}_i \times \vec{v}_i + \vec{L}_i^{\text{intr}} \tag{5.70}$$

This first part is akin to orbital angular momentum, whilst the second term is the spin angular momentum which we propagate directly through rotational dynamics.

Likewise, the moment of inertia tensor could be considered to consist of an orbital part (about the centre of mass of the interacted system) and a spin part (about the centre of mass of the particle):

$$\mathbf{I} = \sum_i \mathbf{I}_i, \quad \mathbf{I}_i = -m_i [\vec{r}_i]^2 + \mathbf{I}_i^{\text{intr}} \tag{5.71}$$

If particles are symmetric, then their moment of inertia $\mathbf{\Lambda}_i$ are a multiple of the identity matrix. In this case, their moment of inertia can be expressed as a single scalar I_i, and is now independent of their rotation. This greatly simplifies their equations:

$$\vec{L}_i = m_i \vec{r}_i \times \vec{v}_i + I_i \vec{\omega}_i \tag{5.72}$$

$$\mathbf{I}_i = I_i \mathbf{1} - m_i [\vec{r}_i]^2 \tag{5.73}$$

As before, the total angular momentum and moment of inertia tensor for a set of particles is the sum of their individual contributions.

The angular component of the rigid motion force is equal to:

$$\vec{F}_i = \frac{m_i}{M}(k\vec{\theta} - \gamma\vec{\omega}) \times \vec{r}_i \tag{5.74}$$

This corresponds to an angular acceleration $\vec{\alpha}$ being applied to the *positions* of the particles being equal to:

$$\vec{\alpha} = \frac{1}{M}(k\vec{\theta} - \gamma\vec{\omega}) \tag{5.75}$$

As the positions of the particles rotate, the particles themselves must also rotate in order to keep the entire system aligned. Just as an equal acceleration has to be applied to all particles when translating them, when rotating we must apply an equal angular acceleration on each particle to keep them aligned. Figure From Equation 5.66, this torque must be:

$$\vec{\tau}_i = \frac{\mathbf{I}_i}{M}(k\vec{\theta} - \gamma\vec{\omega}) + \vec{\omega}_i \times (\mathbf{I}_i \vec{\omega}_i) \tag{5.76}$$

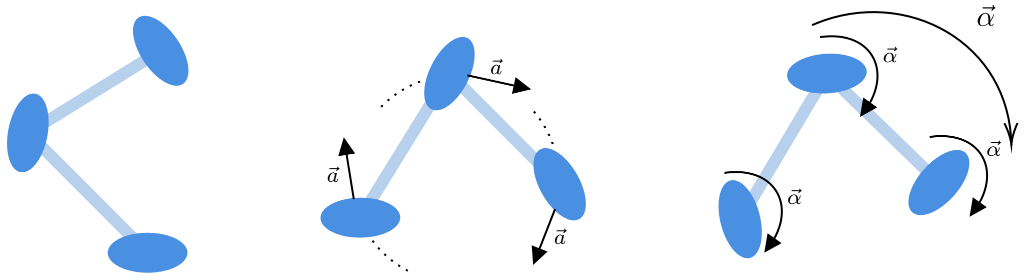
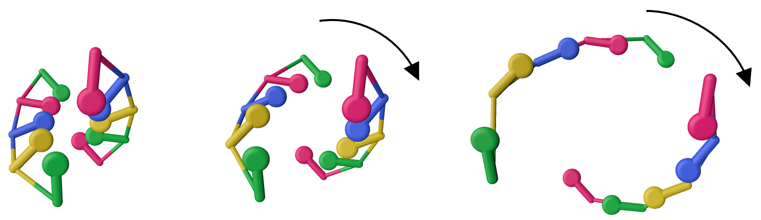


Figure 5.13: Illustration that when rotating a set of asymmetric particles, an angular acceleration must be applied to each particle in addition to the acceleration applied to rotate the three particles about their centre of mass.

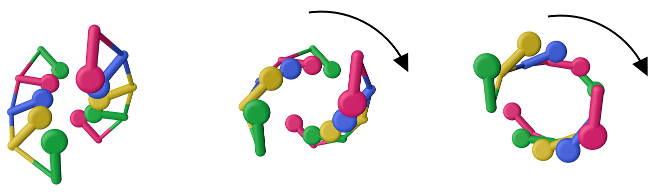
For symmetric finite-sized particles, this takes the simpler form:

$$\vec{\tau}_i = \frac{I_i}{M}(k\vec{\theta} - \gamma\vec{\omega}) \tag{5.77}$$

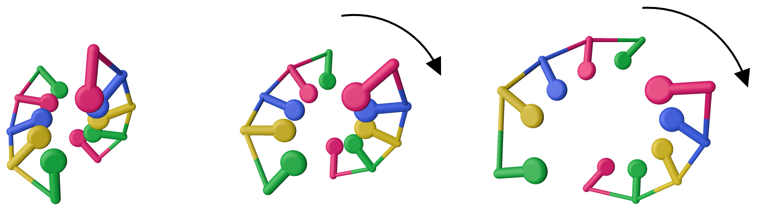
Figure 5.14 illustrates the importance of this per-particle torque when rotating asymmetric particles. This example uses a small double helix using the oxDNA model, with internal forces switched off. In the absence of the centripetal force, the helices spiral outwards as they are rotated. However, Figure 5.14b shows that even with the forces described in the previous section, the base pairs no longer face each other as the helix is rotated. This is because as we rotate the helix, we must also rotate each individual particle such that they continue to face the same way. The correct approach of applying per-particle torques as described in Equation 5.77 is shown in Figure 5.14d, where the correct motion is observed.



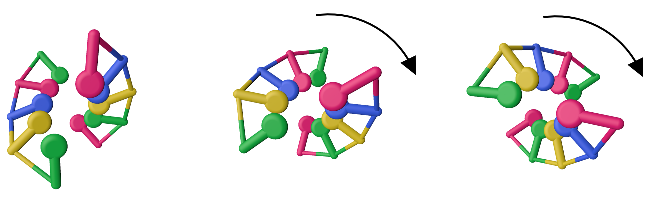
(a) Without per-particle torques or centripetal force.



(b) Without per-particle torques, but with centripetal force.



(c) With per-particle torques, but without centripetal force.



(d) With per-particle torques and centripetal force.

Figure 5.14: Rotation of an oxDNA duplex using my rotational force, illustration the effect of the centripetal force and the per-particle torques.

5.5 Conclusions

Interactive molecular dynamics is a technique that allows human intuition and guidance to be incorporated into simulations. However, it is important to understand its limitations. Although it shares many commonalities with steered molecular dynamics, due to the instant feedback required IMD uses forces which are orders of magnitude higher than SMD. The extreme perturbations this applies to the simulations means that direct interpretation of IMD-VR pathways is difficult.

However, as Lanrezac, Férey, and Baaden[270] and others have said, where IMD shines is as a tool for exploring hypotheses. It enables a user to understand their simulation, as dynamically probing the molecular system and its reaction to your perturbation gives insight into the forces involved. Authors such as Deeks et al.[240] have used IMD-VR pathways as a starting point for other enhanced sampling techniques such as metadynamics.

Harmonic restraints, ubiquitous in steered molecular dynamics, are generally ill-suited where the location of the constraint is moved quickly. This lead to the adoption of a Gaussian well as a second form of interaction in Narupa iMD. However, this comes with its own issue of having no effect at large distances. Another approach used in Narupa iMD is to clip the maximum force possible, though the potential energy must also be modified.

Deeks et al.[25] showed that using IMD-VR, novices with little training could recover crystallographic binding poses. This highlights the power that IMD-VR has to allow detailed atomistic manipulations to facilitate flexible drug docking. However, a fundamental limitation of IMD-VR as previously implemented is that it was simply a mirror of existing SMD approaches — the user merely controls a point to which the current fragment is pulled towards. I believe that this fails to capitalise on the far greater control possible using the full six degrees of freedom available with a VR controller.

To this end, I have developed an interaction paradigm where rotational and translational forces can be applied to a fragment, without perturbing its internal structure. By utilising a critically-damped harmonic spring, I also avoid a common issue in IMD-VR where a molecule 'overshoots' the target position and oscillates. As my approach applies forces, rather than rigidly fix the target fragment in one conformation, it both permits easy integration into our existing IMD-VR framework and allows internal motion of the molecule to still occur. By looking at the partition of kinetic energy, I have shown that my interaction rotates a molecule without altering the translational or internal kinetic energy.

Rigid motion interactions also allow IMD to be extended to molecular simulations where particles are no longer point masses, but can have their own orientation and rotational dynamics. The derivation for the interactions effectively treated the collection of point masses that were interacted with as a single body, and hence is easily applicable to individual rigid body particles. This is achieved by incorporating the individual particles' angular momenta and moment of inertia into the expressions. With this, IMD-VR can be extended to systems where it was not previously possible, such as coarse-grained simulations of DNA.

These interactions provide precise control over the motion of molecular fragments amd bring the user closer to experiencing a kind of ‘molecular surgery’ — a concept that has often been espoused as a goal for IMD-VR.[23] Rigid motion interactions also form a core part of the message of this thesis — adapting tools and visualisations for interactive molecular dynamics to the technology of virtual reality. By using both the position and rotation of the user’s controller, the interaction provides control over the 6 degrees of rotational and translation freedom simultaneously.

Chapter 6

Conclusions

The overarching message of this thesis is that the tools and visualisations we use must not simply be transferred but adapted when transitioning from a 2D interface to VR. Partly this is because VR presents unique challenges, such as requiring efficient and robust rendering algorithms to prevent frame drops and motion sickness. However, we should also take advantage of the freedom and control the user has over the simulation within their virtual world. With two controllers tracking both the position and orientation of their hands, we have access to many possible manipulations that can mimic real motions, and hence feel more intuitive than corresponding movements of a cursor on a screen.

Whilst our research has predominantly been focused on biochemical systems, there are great possibilities in applying IMD-VR to other domains such as materials science. These systems present unique and interesting challenges, such as periodic lattices contained within a well-defined region. I have developed and implemented an approach I hope can be adopted for visualising these systems, allowing new insight into periodic lattices. IMD-VR can be used to explore dynamic problems such as diffusion of small molecules through zeolite networks, which has vital importance for fuel storage and catalysis. Further engagement with industrial collaborators will lead to increased adoption and highlight the utility of this technology in a non-research capacity.

The sphere of rendering biological systems has seen a lot of progress in the previous decades, due to their relevance to the pharmaceutical industry. However, there is scope for improvement for developing ways of rendering solid state systems. Whilst most visualisation software has the capacity to render ribbon diagrams, fewer have the ability to render the polyhedral coordination spheres often used for crystal structures. I have aimed to include this visualisations within Narupa to ensure it provides utility to both biochemists and material scientists.

In addition to research applications, IMD-VR has applications in public engagement and education. The relatively easy setup and hardware required for a multi-user experience makes the technology well-suited for open days and other public events. In my experience, it's a technology that interests and excites people. This engagement with people who may one day become computational chemists is a vital part of ensuring IMD-VR is a sustainable technology which continues to be used. I believe that a core goal in science should be to present research and discovery in new and exciting ways, in order to inspire the next generation of scientists.

In the spirit of this thesis, I also challenge the simple transplantation of the harmonic constraint approach of SMD directly into IMD-VR, by simply allowing the user to manipulate a single point at which the interaction is centred. Whilst overall translations to the molecule can be applied, tt gives the appearance of 'throwing' the molecules around and small changes to the molecule's rotation are hence difficult to achieve precisely. The rigid motion interaction which I propose in Chapter 5 grants the user precise control over the motion of the molecule, leveraging the full six degrees of freedom that a virtual reality controller affords us. The minute and accurate control that IMD-VR offers over a simulation also would make it a valuable tool for studying the motion of molecular machines, such as rotors and switches.

My extension to the IMD-VR paradigm to allow rotational interactions also opens the door to applying the technology to simulations where traditionally these techniques have not been used. By allowing the rotation of asymmetric coarse-grained particles, we can use IMD-VR to interact with models such as the oxDNA model of DNA. The helices of DNA can fold to adopt a wide range of nanostructures, which makes it an ideal target for exploring in a fully immersive 3D environment.

Whilst IMD-VR shares some similarities with SMD, it is unwise to interpret the direct results as physically meaningful. The magnitude of the forces commonly used in IMD are often magnitudes larger than those used in SMD. As noted by Lanrezac, Férey, and Baaden[270] and others, this is an inherent limitation of IMD — the forces must be large enough such that a user can perturb the simulation over a short time period, in the order of tens of seconds. Whilst SMD can be used to determine free energies, it requires many replications and the use of small pulling velocities.

The trajectories obtained by IMD-VR must be interpreted carefully, however they can be used to determine routes which are more or less likely. The pathways may then guide more detailed calculations. Using techniques such as PCA, the IMD-VR pathways can define the collective variables used to run another enhanced-sampling technique such as metadynamics or BXD. It is in this way that IMD shines — as a method for exploring possible hypotheses.

Another possible remedy to the limited simulation time of a typical user session is to distribute the task. Great success has been had with itizen science projects such as Foldit[271] and UDock[272]. This gamification of scientific research taps into an inherent competitiveness in human nature, and is an exciting future avenue of research.[33] Narupa itself has been used for this kind of research, with Shannon et al.[27] showing that allowing a small cohort of users time to explore a reactive system using gamified form of IMD-VR generated similar reaction networks to existing literature.

The increasing overlap of serious scientific software with computer gaming also addresses another issue facing the field — that these projects are only a possibility due to the multidisciplinary collaborations between computer scientists and biochemists. As the video game industry addresses the visualisation and user interface challenges of the future and adapts to new technology, it is vital that we allow this to bleed through into our scientific software. By leveraging the software and game engines developed to make game development more streamlined, we reduce development time and enable scientists to focus on their research.[33] As someone with experience in both game development and scientific research, I believe it is this marriage of the two disciplines that makes IMD-VR an exciting and innovative technology.

Appendix A

Transformations & Affine Coordinates

Linear transformations of points and directions are involved in a vast majority of this thesis, from transforming points between the various points of the graphics pipeline, defining periodic unit cells and defining rotating frames. Here I will lay out the basics of transformations and affine coordinates within the context of this thesis.

A.1 Linear Transformations

Points and directions are represented in three-dimensional space by column vectors:

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{A.1}$$

This vector can be considered as the sum of three unit vectors, $\hat{e}^x$, $\hat{e}^y$ and $\hat{e}^z$:

$$\vec{v} = x\ \hat{e}^x + y\ \hat{e}^y + z\ \hat{e}^z \tag{A.2}$$

Linear transformations are transformations $f(\vec{v})$ for which the following two identities hold:

$$f(\vec{v} + \vec{w}) = f(\vec{v}) + f(\vec{w}) \tag{A.3}$$

$$f(a\vec{v}) = af(\vec{v}) \tag{A.4}$$

As any vector can be written as a linear combination of three unit vectors, any linear transformation need only describe how each unit vector maps to the other unit vectors. This is equivalent to saying that any linear transformation can be written as a 3×3 matrix. The columns of this matrix describe where each unit axis are mapped to by the transformation:

$$\begin{pmatrix} m_{xx} & m_{xy} & m_{xz} \\ m_{yx} & m_{yy} & m_{yz} \\ m_{zx} & m_{zy} & m_{zz} \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} m_{xx}a + m_{xy}b + m_{xz}c \\ m_{yx}a + m_{yy}b + m_{yz}c \\ m_{zx}a + m_{zy}b + m_{zz}c \end{pmatrix} \tag{A.5}$$

$$= a \begin{pmatrix} m_{xx} \\ m_{yx} \\ m_{zx} \end{pmatrix} + b \begin{pmatrix} m_{xy} \\ m_{yy} \\ m_{zy} \end{pmatrix} + c \begin{pmatrix} m_{xz} \\ m_{yz} \\ m_{zz} \end{pmatrix} \tag{A.6}$$

As linear transformations map points in space to other points, they can be interpreted as geometric transformations of space. If the matrix M is a multiple of the identity matrix, then it scales all points by a given value. If the matrix has determinant 1 and is orthogonal (its inverse is equal to its transpose), it can be interpreted as a rotation around the origin.

A.2 Affine Transformations

Whilst rotations, scalings and shears can all be represented as linear transformations, translations cannot. This is because a 3×3 matrix cannot map the zero vector (the origin) to any other point except $\vec{0}$. However, this can be addressed using **affine coordinates**. Here, we extend our vectors to be four-dimensional, with a fourth component w. This fourth component is set to 1 for points, and 0 for directions. Our transformation matrices are now also 4×4 matrices, of the general form:

$$\begin{pmatrix} \mathbf{R} & \vec{v} \\ \vec{0}^T & 1 \end{pmatrix} \tag{A.7}$$

Here, $\mathbf{R}$ is the 3×3 linear transformation as before, and $\vec{v}$ is a 3-dimensional vector that will translate the points. For points, they therefore transform as follows:

$$\begin{pmatrix} \mathbf{R} & \vec{v} \\ \vec{0}^T & 1 \end{pmatrix} \begin{pmatrix} \vec{p} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R}\vec{p} + \vec{v} \\ 1 \end{pmatrix} \tag{A.8}$$

Meanwhile, for directions we use a 0 as the w component for the affine vector:

$$\begin{pmatrix} \mathbf{R} & \vec{v} \\ \vec{0}^T & 1 \end{pmatrix} \begin{pmatrix} \vec{p} \\ 0 \end{pmatrix} = \begin{pmatrix} \mathbf{R}\vec{p} \\ 0 \end{pmatrix} \tag{A.9}$$

Our choice in matrices and w components ensures that:

1. Points are mapped to other points, and are translated appropriately.

2. Directions are mapped to other directions, and are unaffected by translations.

Affine transformations find widespread use in computer graphics, where they are employed in the graphics pipeline to convert between different coordinate systems.

Normals

In the graphics pipeline, it is important to not just transform points but also surface normals. However, there is an important distinction to be made when non-uniform scaling is applied to normals. This is because normals are defined as the cross product of two tangent directions on the surface, and hence are an example of a pseudovector. Another example is angular velocity. Consider a surface normal $\vec{n}$ defined as being tangent to a direction $\vec{a}$:

$$\vec{n} \cdot \vec{a} = 0 \tag{A.10}$$

As a is a tangent to the surface, it transforms normally by some transformation $\mathbf{R}$. If we wish for the above equation to still hold, we presume some transformation $\mathbf{W}$ will have to be applied to $\vec{n}$ to ensure the above equation still holds:

$$(\mathbf{W}\vec{n}) \cdot (\mathbf{R}\vec{a}) = 0 \tag{A.11}$$

$$\vec{n}\mathbf{W}^T\mathbf{R}\vec{a} = 0 \tag{A.12}$$

For this to hold irregardless of $\vec{n}$ and $\vec{a}$, the matrix product must be the identity. Therefore:

$$\mathbf{W} = \mathbf{R}^{-T} \tag{A.13}$$

Therefore, pseudovectors such as normals must transform according to the **inverse transpose** of the transformation matrix applied normally. If no scaling is involved and $\mathbf{R}$ is a rotation matrix, then this distinction is irrelevant as the inverse transpose of a rotation matrix is itself. However, when scaling is involved this is an important distinction.

Generally, we renormalise normals before using them, so we only need to specify $\mathbf{W}$ up to a multiplicative constant. Therefore, we can avoid calculating a full inverse transpose by noting that:

$$\mathbf{R}^{-T} = \frac{1}{\det \mathbf{R}} \operatorname{cof}(\mathbf{R}) \tag{A.14}$$

Here, $\operatorname{cof}(\mathbf{R})$ is the cofactor matrix of $\mathbf{R}$, where each element is the determinant of $\mathbf{R}$ when the corresponding row and column are removed. Hence, we can transform normals by the cofactor of the transformation matrix we use for directions.

A.3 Perspective Transformations

Affine transformations by definition preserve both the straightness of lines and parallelism — if two lines are parallel, then the affine transformation of the two lines are also parallel. However, one of the stages of rendering requires a **perspective transformation**, where objects further away from the camera are made smaller. In this case, lines may no longer stay parallel (consider a road getting thinner as it passes into the horizon).

To achieve a perspective transformation, we wish to have a matrix which divides the x and y coordinates by the z coordinate. Generally, this is not possible using a matrix, which can only achieve a linear combination of the three components. However, by using the 4^{th} affine component, we can achieve this.

Consider the matrix:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{A.15}$$

When transforming a point (x, y, z), this gives a new point:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix} \tag{A.16}$$

However, we expect a point to have an affine component of 1. To correct this, we perform a **perspective division** by dividing all the components by the last components, to give:

$$\Big(x/z, y/z, 1, 1\Big) \tag{A.17}$$

Perspective transformations are therefore achieved by modifying the last row of a transformation matrix, and then dividing through by the last component of the resulting vector.

Appendix B

Splines

Splines are curves made of individual segments, each of which is a polynomial in a parameter t. They are a wide field with many possible parameterisations and forms, and readers interested in more details can find information in the various textbooks on the subject. This appendix will summarise the main splines used for computer graphics, with particular interest to those utilised in ribbon diagrams.

B.1 Bezier Curves

Though not commonly employed directly in ribbon diagrams, Bezier curves are a fundamental curve which appear in many aspects of computer graphics.

The most common way of introducing Bezier curves is to consider linear interpolation. Linear interpolation takes a start point and an end point, and a parameter t in the range $[0, 1]$. As this parameter goes from 0 to 1, linear interpolation gives a point that linearly passes from the start to the end. A linear Bezier curve is defined in terms of two control points, and is exactly the same as linear interpolation.

Higher order Bezier curves are defined recursively using the **de Casteljau** algorithm. This algorithm can be seen graphically as repeated linear interpolation. For example, a quadratic Bezier curve requires three control points, $\vec{p}_0$, $\vec{p}_1$ and $\vec{p}_2$. Given a parameter t in the range of 0 to 1, we can consider two different linear interpolations: from p_0 to p_1, and from p_1 to p_2. We could then linearly interpolate between these two new points, using the same value of t. As t varies from 0 to 1, this traces out a quadratic curve in t:

$$\vec{B}(t) = (1-t)^2\vec{p}_0 + 2(1-t)t\vec{p}_1 + t^2\vec{p}_2 \tag{B.1}$$

In terms of the linear interpolation function, we have:

$$\vec{B}(t) = \mathrm{lerp}(\mathrm{lerp}(\vec{p}_0, \vec{p}_1; t), \mathrm{lerp}(\vec{p}_1, \vec{p}_2; t); t) \tag{B.2}$$

This is a **quadratic Bezier curve**, and is defined in terms of three control points $\{\vec{p}_0, \vec{p}_1, \vec{p}_2\}$. Figure B.1 graphically shows how a quadratic Bezier is calculated. We may continue adding

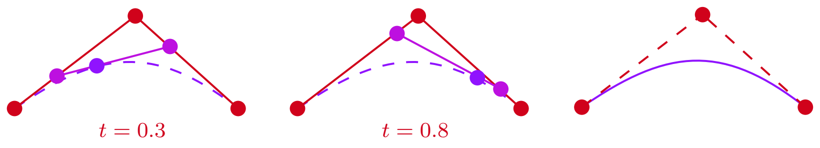


Figure B.1: Illustration of a quadratic Bezier based on three control points. Each point on the curve consists of a linear interpolation between two successive linear interpolations, all using the same value of t.

control points, increasing the order of the Bezier curve. For example, if we scale up to four control points, then we will have a three successive linear interpolations. This yields a **cubic Bezier curve**:

$$\vec{B}(t) = (1-t)^3\vec{p}_0 + 3(1-t)^2t\vec{p}_1 + 3(1-t)t^2\vec{p}_2 + t^3\vec{p}_3 \tag{B.3}$$

The polynomial coefficients of each control point for a degree n Bezier curve are the **Bernstein basis polynomials** of degree n, expressible as:

$$b_{\nu,n}(t) = \binom{n}{\nu} t^{\nu}(1-t)^{n-\nu}, \quad \nu = 0, ..., n \tag{B.4}$$

Bezier curves are widespread in computer graphics. The outline of computer fonts are stored as Bezier curves, ensuring they can be scaled up to arbitrary size.

B.2 Cubic Hermite Spline

We will now move on to the derivation of the form of the cubic Hermite spline. The following derivation of the form of a cubic Hermite spline is based on Chapter 3 of Bartels, Beatty, and Barsky[273].

We wish to connect a set of control points $\{\vec{p}_i\}$ together with piecewise splines, which will be cubic in a parameter t from 0 to 1. As we have N control points, we define a set of $N-1$ cubic curves $\vec{s}_i(t)$ with the general form:

$$\vec{s}_i(t) = \vec{a}_i + \vec{b}_i t + \vec{c}_i t^2 + \vec{d}_i t^3 \tag{B.5}$$

We have not yet specified the form of $\left\{\vec{a}_i, \vec{b}_i, \vec{c}_i, \vec{d}_i\right\}$, but they will be some function of the control points $\{\vec{p}_i\}$. Equation B.5 is therefore the general form for **any** cubic piecewise spline. If we wish the spline segments to connect subsequent points (namely, be an interpolating C^0 spline), then each spline $\vec{s}_i(t)$ must pass from $\vec{p}_i$ to $\vec{p}_{i+1}$. Therefore:

$$\vec{s}_i(0) = \vec{p}_i = \vec{a}_i \tag{B.6}$$

$$\vec{s}_i(1) = \vec{p}_{i+1} = \vec{a}_i + \vec{b}_i + \vec{c}_i + \vec{d}_i \tag{B.7}$$

If continuity in the tangents at each curve is also enforced, then at each point $\vec{p}_i$ a tangent $\vec{m}_i$ must exist, that is shared by the end of the preceding segment and the start and the next segment. This requirement enforces additional constraints:

$$\vec{s}_i'(0) = \vec{m}_i = \vec{b}_i \tag{B.8}$$

$$\vec{s}_i'(1) = \vec{m}_{i+1} = \vec{b}_i + 2\vec{c}_i + 3\vec{d}_i \tag{B.9}$$

Combining these four equations determines the values for the four unknowns for the cubic spline:

$$\begin{aligned}
\vec{a}_i &= \vec{p}_i \\
\vec{b}_i &= \vec{m}_i \\
\vec{c}_i &= -3\vec{p}_i + 3\vec{p}_{i+1} - 2\vec{m}_i - \vec{m}_{i+1} \\
\vec{d}_i &= 2\vec{p}_i - 2\vec{p}_{i+1} + \vec{m}_i + \vec{m}_{i+1}
\end{aligned}$$

When inserted back into Equation B.5 for a general cubic spline, this yields the general form of the **cubic Hermite spline**:

$$\vec{s}_i(t) = (2t^3 - 3t^2 + 1)\vec{p}_i + (-2t^3 + 3t^2)\vec{p}_{i+1} + (t^3 - 2t^2 + t)\vec{m}_i + (t^3 - t^2)\vec{m}_{i+1} \tag{B.10}$$

The cubic Hermite spline therefore represents the most general case of a cubic spline, that passes through a set of points and has C^1 continuity. However, it requires the specification of an additional N tangent values $\{\vec{m}_i\}$.

The polynomials that appear as coefficients in Equation B.10 are known as the **Hermite basis functions**. One useful parallel to draw is that there is a relationship between the cubic Hermite curve and a cubic Bezier curve. Instead of considering two points $\vec{p}_i$ and $\vec{p}_{i+1}$ and their tangents $\vec{m}_i$ and $\vec{m}_{i+1}$, define four control points as $\vec{p}_i$, $\vec{p}_i + \frac{\vec{m}_i}{3}$, $\vec{p}_{i+1} - \frac{\vec{m}_i}{3}$ and $\vec{p}_{i+1}$. These four control points define a cubic Bezier curve that is exactly the same as the cubic Hermite curve. This emphasises that cubic Hermite curves and cubic Bezier curves are not fundamentally different — they are merely two different ways of describing the same curve, either as two endpoints and two tangents, or as two endpoints and two control points.

Catmull-Rom

Cubic Hermite splines require both a set of control points $\{\vec{p}_i\}$ and a set of tangents $\{\vec{m}_i\}$. A common technique is to calculate the tangents based on nearby points, with the most common approach being the Catmull-Rom spline:

$$\vec{m}_i = \frac{1}{2}(\vec{p}_{i+1} - \vec{p}_{i-1}) \tag{B.11}$$

With this, each spline segment is a weighted sum of four control points (those forming the endpoints of the segment and the two preceding and following the segment):

$$\vec{s}_i(t) = \left(-\frac{t}{2} + t^2 - \frac{t^3}{2}\right)\vec{p}_{i-1} + \left(1 - \frac{5t^2}{2}t^2 + \frac{3t^3}{2}\right)\vec{p}_i + \left(\frac{t}{2} + 2t^2 - \frac{3t^3}{2}\right)\vec{p}_{i+1} + \left(-\frac{t^2}{2} + \frac{t^3}{2}\right)\vec{p}_{i+2} \tag{B.12}$$

Note that these four polynomials sum to 1 for all t.

B.3 Natural Cubic Spline

If second order continuity is also enforced, then at each point $\vec{p}_i$ there must exist a second derivative $\vec{n}_i$, which is shared by the end of the previous segment and the start of the next. As was the case for the tangents $\{\vec{m}_i\}$, this constraint yields additional equations:

$$\begin{aligned}\vec{s}_i''(0) &= \vec{n}_i = 2\vec{c}_i \\ \vec{s}_i''(1) &= \vec{n}_{i+1} = 2\vec{c}_i + 6\vec{d}_i\end{aligned}$$

These equations hence link together gradients due to the relationship between consecutive $\vec{c}_i$ and $\vec{d}_i$:

$$\begin{aligned}2\vec{c}_i =& 2\vec{c}_{i-1} + 6\vec{d}_{i-1} \\ 2(-3\vec{p}_i + 3\vec{p}_{i+1} - 2\vec{m}_i - \vec{m}_{i+1}) =& 2(-3\vec{p}_{i-1} + 3\vec{p}_i - 2\vec{m}_{i-1} - \vec{m}_i) \\ &+ 6(2\vec{p}_{i-1} - 2\vec{p}_i + \vec{m}_{i-1} + \vec{m}_i) \\ 3(\vec{p}_{i+1} - \vec{p}_{i-1}) =& \vec{m}_{i-1} + 4\vec{m}_i + \vec{m}_{i+1}\end{aligned}$$

The requirement of C^2 continuity therefore leads to $N-2$ equations that determine the set of N gradients $\{\vec{m}_i\}$. To uniquely define the gradients, two additional constraints are required. The natural boundary condition is to assert that the second derivative tends to 0 at the endpoints:

$$\vec{n}_0 = \vec{n}_{N-1} = 0 \tag{B.13}$$

The first of these leads to:

$$\begin{aligned}\vec{c}_0 = 0 \quad \rightarrow \quad -3\vec{p}_0 + 3\vec{p}_1 - 2\vec{m}_0 - \vec{m}_1 &= 0 \\ 3(\vec{p}_1 - \vec{p}_0) &= 2\vec{m}_0 - \vec{m}_1\end{aligned}$$

The second boundary conditions yields:

$$\begin{aligned}\vec{c}_{N-1} = 0 \quad \rightarrow \quad \vec{c}_{N-2} + 3\vec{d}_{N-2} &= 0 \\ 3(\vec{p}_{N-1} - \vec{p}_{N-2}) &= \vec{m}_{N-2} + 2\vec{m}_{N-1}\end{aligned}$$

Therefore, there are a set of N equations to determine the values of the tangents $\{\vec{m}_i\}$. These may be collected into a matrix of the form

$$\begin{pmatrix} 2 & 1 & & & & \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & & \ddots & & \\ & & & 1 & 4 & 1 \\ & & & & 1 & 2 \end{pmatrix} \begin{pmatrix} \vec{m}_0 \\ \vec{m}_1 \\ \vec{m}_2 \\ \vdots \\ \vec{m}_{N-2} \\ \vec{m}_{N-1} \end{pmatrix} = 3 \begin{pmatrix} \vec{p}_1 - \vec{p}_0 \\ \vec{p}_2 - \vec{p}_0 \\ \vec{p}_3 - \vec{p}_1 \\ \vdots \\ \vec{p}_{N-1} - \vec{p}_{N-3} \\ \vec{p}_{N-1} - \vec{p}_{N-2} \end{pmatrix}$$

Solving the above matrix yields the tangents m_i that determine a natural cubic spline.

The tangents $\{\vec{m}_i\}$ are therefore completely determined by the control points $\{\vec{p}_i\}$. However, solving the above problem means that each tangent does depend on all the control points. This means that if a single control point is altered, every segment of the entire spline will be affected.

B.4 B-Splines

The following derivation is based on Chapter 4 of Bartels, Beatty, and Barsky[273].

B-splines are a broad category of splines that generalise a lot of the features of other splines. Here we will specifically consider the uniform B-spline.

To start, consider how the Catmull-Rom spline segment is a cubic in t that depends on four points: $\vec{p}_{i-1}$, $\vec{p}_i$, $\vec{p}_{i+1}$ and $\vec{p}_{i+2}$. Consider the general case for any cubic spline which depends on these four points:

$$\vec{s}_i(t) = b_0(t)\vec{p}_{i-1} + b_1(t)\vec{p}_i + b_2(t)\vec{p}_{i+1} + b_3(t)\vec{p}_{i+2} \tag{B.14}$$

Each of these basis function $b_k(t)$ will be a cubic in t, and hence there are 16 coefficient that determine the shape of the spline. We are inherently forcing our result to have local support — we presume that each segment can only depend on four points. We now seek to find the cubic factors $b_k(t)$ that will define the spline.

First, requiring C^0 continuity requires that $\vec{s}_i(1) = \vec{s}_{i+1}(0)$ and $\vec{s}_i(0) = \vec{s}_{i-1}(1)$. This results in two requirements:

$$b_0(1)\vec{p}_{i-1} + b_1(1)\vec{p}_i + b_2(1)\vec{p}_{i+1} + b_3(1)\vec{p}_{i+2} = b_0(0)\vec{p}_i + b_1(0)\vec{p}_{i+1} + b_2(0)\vec{p}_{i+2} + b_3(0)\vec{p}_{i+3}$$
$$b_0(0)\vec{p}_{i-1} + b_1(0)\vec{p}_i + b_2(0)\vec{p}_{i+1} + b_3(0)\vec{p}_{i+2} = b_0(1)\vec{p}_{i-2} + b_1(1)\vec{p}_{i-1} + b_2(1)\vec{p}_i + b_3(1)\vec{p}_{i+1}$$

As these must hold for any value that $\{\vec{p}_i\}$ can take, this gives:

$$b_0(1) = 0 \tag{B.15}$$
$$b_3(0) = 0 \tag{B.16}$$
$$b_1(1) = b_0(0) \tag{B.17}$$
$$b_2(1) = b_1(0) \tag{B.18}$$
$$b_3(1) = b_3(0) \tag{B.19}$$

Our next requirement of C^1 continuity applies similar requirements, but on the derivatives of the basis functions. This yields another set of 5 constraints:

$$b_0'(1) = 0 \tag{B.20}$$
$$b_3'(0) = 0 \tag{B.21}$$
$$b_1'(1) = b_0'(0) \tag{B.22}$$
$$b_2'(1) = b_1'(0) \tag{B.23}$$
$$b_3'(1) = b_3'(0) \tag{B.24}$$

Finally, if we require C^2 continuity, then applying the same logic yields a third set of 5 constraints:

$$b_0''(1) = 0 \tag{B.25}$$
$$b_3''(0) = 0 \tag{B.26}$$
$$b_1''(1) = b_0''(0) \tag{B.27}$$
$$b_2''(1) = b_1''(0) \tag{B.28}$$
$$b_3''(1) = b_3''(0) \tag{B.29}$$

All together, we require 16 coefficients to define our basis functions. Simply by assuming a C^2 continuity cubic spline with local support, we have 15 constraints to determine these coefficients. A final constraint is to assert that summing these four basis functions equals 1 for all t:

$$b_0(t) + b_1(t) + b_2(t) + b_3(t) = 1 \tag{B.30}$$

With these 16 constraints, we can solve this for the uniform cubic B-spline basis functions:

$$b_0(t) = \frac{1}{6}t^3 \tag{B.31}$$
$$b_1(t) = \frac{1}{6}(1 + 3t + 3t^2 - 3t^3) \tag{B.32}$$
$$b_2(t) = \frac{1}{6}(4 - 6t^2 + 3t^3) \tag{B.33}$$
$$b_3(t) = \frac{1}{6}(1 - 3t + 3t^2 - t^3) \tag{B.34}$$

This derivation shows that B-splines arise naturally from our requirements — all we have enforced is C^1 continuity, local support and ensured the weights summed to 1. As a consequence, the spline is no longer interpolating. This is a fundamental fact — enforcing a spline to go through the control points and have C^2 continuity results in global support (for example the natural cubic spline), whilst enforcing local support causes the spline to no longer pass through the control points.

Appendix C

Quaternions

As a brief reminder, complex numbers may be expressed in terms of the imaginary unit i, which fulfils the following identity:

$$i^2 = -1 \tag{C.1}$$

From this, we define the **complex numbers** as numbers constituting a real part and an imaginary part, which is a multiple of i:

$$z = x + yi \tag{C.2}$$

To define quaternions, we define three separate units: i, j and k. These fulfil the identity:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{C.3}$$

Analogously to complex numbers, we can define a quaternion as having four components — a real component and three imaginary components:

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{C.4}$$

It can therefore be taken to be similar to a complex number, but with a three-dimensional imaginary part:

$$\operatorname{Re}\mathbf{q} = q_0, \quad \operatorname{Im}\mathbf{q} = \begin{pmatrix} q_1 \\ q_2 \\ q_3 \end{pmatrix} = \vec{q} \tag{C.5}$$

We may chose to to represent a quaternion as a column vector:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix} \tag{C.6}$$

The multiplication of two quaternions $\mathbf{q}$ and $\mathbf{p}$ may be expanded out using Equation C.3:

$$\begin{aligned}(q_0 + q_1 i + q_2 j + q_3 k)(p_0 + p_1 i + p_2 j + p_3 k) =& (p_0 q_0 - q_1 p_1 - q_2 p_2 - q_3 p_3) \\ &+ (q_1 p_0 + q_0 p_1 + q_2 p_3 - q_3 p_2) i \\ &+ (q_2 p_0 + q_0 p_2 + q_3 p_1 - q_1 p_3) j \\ &+ (q_3 p_0 + q_0 p_3 + q_1 p_2 - q_2 p_1) k\end{aligned}$$

This may be simplified by considering vector identities, and expressing each quaternion in column vector form:

$$\begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix} \begin{bmatrix} p_0 \\ \vec{p} \end{bmatrix} = \begin{bmatrix} q_0 p_0 - \vec{q} \cdot \vec{p} \\ q_0 \vec{p} + p_0 \vec{q} + \vec{q} \times \vec{p} \end{bmatrix} \tag{C.7}$$

Importantly, the multiplication of two quaternions is non-commutative — **qp** is not the same as **pq**.

Like complex numbers, the complex conjugate can be defined by negating the imaginary part of the quaternion:

$$\bar{\mathbf{q}} = \begin{bmatrix} q_0 \\ -\vec{q} \end{bmatrix} \tag{C.8}$$

Also, the length of a quaternion may be defined in a similar manner:

$$|\mathbf{q}| = \sqrt{q_0^2 + |\vec{q}|^2} \tag{C.9}$$

Each quaternion does commute with its complex conjugate, ans the same relationship between conjugates and magnitudes holds:

$$\mathbf{q}\bar{\mathbf{q}} = \bar{\mathbf{q}}\mathbf{q} = |\mathbf{q}|^2 \tag{C.10}$$

C.1 Quaternions as rotations

Firstly, we will invoke (without proof) Rodrigues' rotation formula, which states that the rotation of $\vec{v}$ about a unit vector $\vec{k}$ by an angle θ is given by:

$$\vec{v}' = \vec{v} \cos\theta + (\vec{k} \times \vec{v}) \sin\theta + \vec{k}(\vec{k} \cdot \vec{v})(1 - \cos\theta) \tag{C.11}$$

Now consider some arbitrary vector $\vec{x}$. We can project a 3D vector $\vec{x}$ into a quaternion by using it as the imaginary part, and setting the real part to 0. By doing this, we can define multiplications of quaternions and vector. Now, we consider the expression obtained by sandwiching our vector $\vec{v}$ between the quaternion and its conjugate:

$$\begin{aligned}\bar{\mathbf{q}}\vec{x}\mathbf{q} &= \begin{bmatrix} q_0 \\ -\vec{q} \end{bmatrix} \begin{bmatrix} 0 \\ \vec{x} \end{bmatrix} \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix} \\ &= \begin{bmatrix} q_0 \\ -\vec{q} \end{bmatrix} \begin{bmatrix} -\vec{x}\cdot\vec{q} \\ q_0\vec{x} + \vec{x}\times\vec{q} \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 2(\vec{q}\cdot\vec{x})\vec{q} + q_0^2\vec{x} + 2q_0(\vec{x}\times\vec{q}) - |\vec{q}|^2\vec{x} \end{bmatrix}\end{aligned}$$

The result of sandwiching a vector $\vec{x}$ like this yields a quaternion with 0 real part, which can be interpreted as another vector. We can therefore interpret this action as a transformation of a vector $\vec{x}$ to another $\vec{x}'$ by the quaternion $\mathbf{q}$.

Now, consider the following quaternion expressed in terms of a unit vector $\vec{k}$ and angle θ as:

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \sin\frac{\theta}{2}\vec{k} \end{bmatrix} \tag{C.12}$$

Note that the magnitude of this quaternion is 1, and hence we refer to this as a **unit quaternion**. If we transform a vector $\vec{x}$ by this quaternion, we obtain:

$$\begin{aligned}\bar{\mathbf{q}}\vec{x}\mathbf{q} &= 2\sin\frac{\theta}{2}(\vec{k}\cdot\vec{x})\sin\frac{\theta}{2}\vec{k} + \cos^2\frac{\theta}{2}\vec{x} + 2\cos\frac{\theta}{2}\sin\frac{\theta}{2}\vec{x}\times\vec{k} - \sin^2\frac{\theta}{2}\vec{x} \\ &= \vec{x}\cos\theta + (\vec{k}\times\vec{x})\sin\theta + \vec{k}(\vec{k}\cdot\vec{x})(1-\cos\theta)\end{aligned}$$

This is exactly Equation C.11. Therefore, a unit quaternion of the form Equation C.12 may be used to rotate a vector.

The inverse of this rotation is obtained by flipping either $\vec{k}$ or θ. This evidently flips the imaginary part of Equation C.12, and hence if a unit quaternion represents a rotation, its conjugate represents the inverse rotation.

Bibliography

[1] E. J. Maginn and J. R. Elliott. “Historical Perspective and Current Outlook for Molecular Dynamics As a Chemical Engineering Tool”. In: *Industrial & Engineering Chemistry Research* 49.7 (Apr. 7, 2010). Publisher: American Chemical Society, pp. 3059–3078. ISSN: 0888-5885. DOI: 10.1021/ie901898k.

[2] Neal E. Seymour et al. “Virtual Reality Training Improves Operating Room Performance”. In: *Annals of Surgery* 236.4 (Oct. 2002), pp. 458–464. ISSN: 0003-4932.

[3] Christian Moro et al. “The effectiveness of virtual and augmented reality in health sciences and medical anatomy”. In: *Anatomical Sciences Education* 10.6 (Nov. 2017), pp. 549–559. ISSN: 1935-9780. DOI: 10.1002/ase.1696.

[4] Wee Sim Khor et al. “Augmented and virtual reality in surgery—the digital surgical environment: applications, limitations and legal pitfalls”. In: *Annals of Translational Medicine* 4.23 (Dec. 2016), p. 454. ISSN: 2305-5839. DOI: 10.21037/atm.2016.12.23.

[5] Sangsu Choi, Kiwook Jung, and Sang D Noh. “Virtual reality applications in manufacturing industries: Past research, present findings, and future directions”. In: *Concurrent Engineering-Research and Applications* 23.1 (2015). ISBN: 1063-293x, pp. 40–63. DOI: 10.1177/1063293x14568814.

[6] Abraham Anderson and Zhiping Weng. “VRDD: applying virtual reality visualization to protein docking and design”. In: *Journal of Molecular Graphics and Modelling* 17.3 (June 1, 1999), pp. 180–186. ISSN: 1093-3263. DOI: 10.1016/S1093-3263(99)00029-7.

[7] Rebecca K. Walters et al. *Interactivity: the missing link between virtual reality technology and drug discovery pipelines.* Feb. 8, 2022. arXiv: 2202.03953.

[8] Xiao-Huan Liu et al. “Using virtual reality for drug discovery: a promising new outlet for novel leads”. In: *Expert Opinion on Drug Discovery* 13.12 (Dec. 2, 2018). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17460441.2018.1546286, pp. 1103–1114. ISSN: 1746-0441. DOI: 10.1080/17460441.2018.1546286.

[9] Andrea Salvadori et al. “Immersive virtual reality in computational chemistry: Applications to the analysis of QM and MM data”. In: *International Journal of Quantum Chemistry* 116.22 (2016). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/qua.25207, pp. 1731–1746. ISSN: 1097-461X. DOI: 10.1002/qua.25207.

[10] Jeremy N. Block et al. “KinImmerse: Macromolecular VR for NMR ensembles”. In: *Source Code for Biology and Medicine* 4.1 (Feb. 17, 2009), p. 3. ISSN: 1751-0473. DOI: 10.1186/1751-0473-4-3.

[11] Maria Limniou, David Roberts, and Nikos Papadopoulos. “Full immersive virtual environment CAVETM in chemistry education”. In: *Computers & Education* 51.2 (Sept. 1, 2008), pp. 584–593. ISSN: 0360-1315. DOI: 10.1016/j.compedu.2007.06.014.

[12] J. Georgiou, K. Dimitropoulos, and A. Manitsaris. “A Virtual Reality Laboratory for Distance Education in Chemistry”. In: *International Journal of Educational and Pedagogical Sciences* 1.11 (Nov. 20, 2007), pp. 617–624.

[13] Alan Richardson et al. “Use of a Three-Dimensional Virtual Environment to Teach Drug-Receptor Interactions”. In: *American Journal of Pharmaceutical Education* 77.1 (Feb. 12, 2013). Publisher: American Association of Colleges of Pharmacy. DOI: 10.5688/ajpe77111.

[14] Simon J. Bennie et al. “Teaching Enzyme Catalysis Using Interactive Molecular Dynamics in Virtual Reality”. In: *Journal of Chemical Education* 96.11 (Nov. 12, 2019). Publisher: American Chemical Society, pp. 2488–2496. ISSN: 19381328. DOI: 10.1021/acs.jchemed.9b00181.

[15] E. Moritz and J. Meyer. “Interactive 3D protein structure visualization using virtual reality”. In: *Proceedings. Fourth IEEE Symposium on Bioinformatics and Bioengineering.* Proceedings. Fourth IEEE Symposium on Bioinformatics and Bioengineering. May 2004, pp. 503–507. DOI: 10.1109/BIBE.2004.1317384.

[16] Martín Calvelo, Ángel Piñeiro, and Rebeca Garcia-Fandino. “An immersive journey to the molecular structure of SARS-CoV-2: Virtual reality in COVID-19”. In: *Computational and Structural Biotechnology Journal* 18 (Jan. 1, 2020). Publisher: Elsevier B.V., pp. 2621–2628. ISSN: 20010370. DOI: 10.1016/j.csbj.2020.09.018.

[17] Ching-Man Tse et al. “Interactive Drug Design in Virtual Reality”. In: *2011 15th International Conference on Information Visualisation.* 2011 15th International Conference on Information Visualisation. ISSN: 2375-0138. July 2011, pp. 226–231. DOI: 10.1109/IV.2011.72.

[18] Laura J. Kingsley et al. “Development of a virtual reality platform for effective communication of structural data in drug discovery”. In: *Journal of Molecular Graphics and Modelling* 89 (June 1, 2019). Publisher: Elsevier Inc., pp. 234–241. ISSN: 18734243. DOI: 10.1016/j.jmgm.2019.03.010.

[19] Benjamin N. Doblack, Tim Allis, and Lilian P. Dávila. "Novel 3D/VR Interactive Environment for MD Simulations, Visualization and Analysis". In: *JoVE (Journal of Visualized Experiments)* 94 (Dec. 18, 2014), e51384. ISSN: 1940-087X. DOI: 10.3791/51384.

[20] Zhuming Ai and Torsten Fröhlich. "Molecular Dynamics Simulation in Virtual Environments". In: *Computer Graphics Forum* 17.3 (Aug. 1, 1998). Publisher: Blackwell Publishing Ltd., pp. 267–273. ISSN: 01677055. DOI: 10.1111/1467-8659.00273.

[21] Joseph J. LaViola. "A discussion of cybersickness in virtual environments". In: *ACM SIGCHI Bulletin* 32.1 (Jan. 1, 2000), pp. 47–56. ISSN: 0736-6906. DOI: 10.1145/333329.333344.

[22] Michael O'Connor et al. "Sampling molecular conformations and dynamics in a multiuser virtual reality framework". In: *Science Advances* 4.6 (June 29, 2018). Publisher: American Association for the Advancement of Science, eaat2731. ISSN: 2375-2548. DOI: 10.1126/sciadv.aat2731.

[23] Michael B. O'Connor et al. "Interactive molecular dynamics in virtual reality from quantum chemistry to drug binding: An open-source multi-person framework". In: *The Journal of Chemical Physics* 150.22 (June 14, 2019). Publisher: American Institute of Physics Inc., p. 220901. ISSN: 0021-9606. DOI: 10.1063/1.5092590.

[24] Alexander D. Jamieson-Binnie and David R. Glowacki. "Visual Continuity of Protein Secondary Structure Rendering: Application to SARS-CoV-2 Mpro in Virtual Reality". In: *Frontiers in Computer Science* 0 (July 12, 2021). Publisher: Frontiers, p. 63. ISSN: 2624-9898. DOI: 10.3389/FCOMP.2021.642172.

[25] Helen M. Deeks et al. "Interactive molecular dynamics in virtual reality for accurate flexible protein-ligand docking". In: *PLOS ONE* 15.3 (Mar. 11, 2020). Ed. by Emanuele Paci. Publisher: Public Library of Science, e0228461. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0228461.

[26] Silvia Amabilino et al. "Training atomic neural networks using fragment-based data generated in virtual reality". In: *The Journal of Chemical Physics* 153.15 (Oct. 21, 2020). Publisher: American Institute of Physics, p. 154105. ISSN: 0021-9606. DOI: 10.1063/5.0015950.

[27] Robin J. Shannon et al. "Exploring human-guided strategies for reaction network exploration: Interactive molecular dynamics in virtual reality as a tool for citizen scientists". In: *The Journal of Chemical Physics* 155.15 (Oct. 21, 2021). Publisher: American Institute of Physics, p. 154106. ISSN: 0021-9606. DOI: 10.1063/5.0062517.

[28] Rhoslyn Roebuck Williams et al. "Subtle Sensing: Detecting Differences in the Flexibility of Virtually Simulated Molecular Objects". In: *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems.* New York, NY, USA: ACM, Apr. 25, 2020, pp. 1–8. ISBN: 978-1-4503-6819-3. DOI: 10.1145/3334480.3383026.

[29] Alexander D Jamieson-Binnie et al. “Narupa iMD: A VR-Enabled Multiplayer Framework for Streaming Interactive Molecular Simulations”. In: *ACM SIGGRAPH 2020 Immersive Pavilion*. New York, NY, USA: ACM, Aug. 17, 2020, pp. 1–2. ISBN: 978-1-4503-7968-7. DOI: 10.1145/3388536.3407891.

[30] David R. Glowacki et al. “danceroom Spectroscopy: Interactive quantum molecular dynamics accelerated on GPU architectures using OpenCL”. In: *UK Many Core Development Conference 2012 (UKMAC. ’12)*. 2012.

[31] D. R. Glowacki. “Sculpting molecular dynamics in real-time using human energy fields”. In: *Molecular Aesthetics*. Ed. by Peter Weibel and Ljiljana Fruk. In collab. with Phillip Tew et al. MIT Press, Sept. 2013. ISBN: 978-0-262-01878-4.

[32] David R. Glowacki et al. “A GPU-accelerated immersive audio-visual framework for interaction with molecular dynamics using consumer depth sensors”. In: *Faraday Discuss.* 169.0 (Oct. 23, 2014). Publisher: The Royal Society of Chemistry, pp. 63–87. ISSN: 1359-6640. DOI: 10.1039/C4FD00008K.

[33] Zhihan Lv et al. “Game On, Science - How Video Game Technology May Help Biologists Tackle Visualization Challenges”. In: *PLoS ONE* 8.3 (Mar. 6, 2013). Ed. by Paul Taylor. Publisher: Public Library of Science, e57990. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0057990.

[34] Martin Mittring. “Finding next gen: CryEngine 2”. In: *ACM SIGGRAPH 2007 courses on - SIGGRAPH ’07*. New York, New York, USA: Association for Computing Machinery (ACM), 2007, p. 97. DOI: 10.1145/1281500.1281671.

[35] Marco Tarini, Paolo Cignoni, and Claudio Montani. “Ambient Occlusion and Edge Cueing for Enhancing Real Time Molecular Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 12.5 (Sept. 2006), pp. 1237–1244. ISSN: 1077-2626. DOI: 10.1109/TVCG.2006.115.

[36] Sébastien Doutreligne et al. “UnityMol: Interactive scientific visualization for integrative biology”. In: *IEEE Symposium on Large Data Analysis and Visualization 2014, LDAV 2014 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., Jan. 16, 2014, pp. 109–110. ISBN: 978-1-4799-5215-1. DOI: 10.1109/LDAV.2014.7013213.

[37] Sebastien Doutreligne et al. “UnityMol: interactive and ludic visual manipulation of coarse-grained RNA and other biomolecules”. In: *2015 IEEE 1st International Workshop on Virtual and Augmented Reality for Molecular Science (VARMS@IEEEVR)*. 2015 IEEE 1st International Workshop on Virtual and Augmented Reality for Molecular Science (VARMS@IEEEVR). Mar. 2015, pp. 1–6. DOI: 10.1109/VARMS.2015.7151718.

[38] S. Perez et al. “Three-dimensional representations of complex carbohydrates and polysaccharides–SweetUnityMol: A video game-based computer graphic software”. In: *Glycobiology* 25.5 (May 1, 2015). Publisher: Oxford University Press, pp. 483–491. ISSN: 0959-6658. DOI: 10.1093/glycob/cwu133.

[39] Joseph Laureanti et al. “Visualizing biomolecular electrostatics in virtual reality with UnityMol-APBS”. In: *Protein Science* 29.1 (Jan. 25, 2020). Publisher: Blackwell Publishing Ltd, pp. 237–246. ISSN: 0961-8368. DOI: 10.1002/pro.3773.

[40] X. Martinez and M. Baaden. “UnityMol prototype for FAIR sharing of molecular-visualization experiences: from pictures in the cloud to collaborative virtual reality exploration in immersive 3D environments”. In: *Acta Crystallographica Section D: Structural Biology* 77.6 (June 1, 2021). Number: 6 Publisher: International Union of Crystallography, pp. 746–754. ISSN: 2059-7983. DOI: 10.1107/S2059798321002941.

[41] Magnus Norrby et al. “Molecular Rift: Virtual Reality for Drug Designers”. In: *Journal of Chemical Information and Modeling* 55.11 (Nov. 23, 2015). Publisher: American Chemical Society, pp. 2475–2484. ISSN: 1549-9596. DOI: 10.1021/acs.jcim.5b00544.

[42] Alexander von Wedelstedt, Gunther Goebel, and Grit Kalies. “MOF-VR: A Virtual Reality Program for Performing and Visualizing Immersive Molecular Dynamics Simulations of Guest Molecules in Metal–Organic Frameworks”. In: *Journal of Chemical Information and Modeling* (Feb. 21, 2022). Publisher: American Chemical Society. ISSN: 1549-9596. DOI: 10.1021/acs.jcim.2c00158.

[43] Oscar Legetth et al. “CellexalVR: A virtual reality platform to visualize and analyze single-cell omics data”. In: *iScience* 24.11 (Nov. 19, 2021), p. 103251. ISSN: 2589-0042. DOI: 10.1016/j.isci.2021.103251.

[44] David G. Doak et al. “Peppy: A virtual reality environment for exploring the principles of polypeptide structure”. In: *Protein Science* 29.1 (Jan. 11, 2020). Publisher: Blackwell Publishing Ltd, pp. 157–168. ISSN: 0961-8368. DOI: 10.1002/pro.3752.

[45] Jimmy F. Zhang et al. “BioVR: A platform for virtual reality assisted biological data integration and visualization”. In: *BMC Bioinformatics* 20.1 (Feb. 15, 2019). Publisher: BioMed Central Ltd., p. 78. ISSN: 14712105. DOI: 10.1186/s12859-019-2666-z.

[46] Michael Wiebrands et al. “Molecular Dynamics Visualization (MDV): Stereoscopic 3D Display of Biomolecular Structure and Interactions Using the Unity Game Engine”. In: *Journal of Integrative Bioinformatics* 15.2 (June 26, 2018). ISSN: 1613-4516. DOI: 10.1515/jib-2018-0010.

[47] Heta A. Gandhi et al. “Real-Time Interactive Simulation and Visualization of Organic Molecules”. In: *Journal of Chemical Education* 97.11 (Nov. 10, 2020). Publisher: American Chemical Society, pp. 4189–4195. ISSN: 0021-9584. DOI: 10.1021/acs.jchemed.9b01161.

[48] Stefan Seritan et al. “InteraChem: Virtual Reality Visualizer for Reactive Interactive Molecular Dynamics”. In: *Journal of Chemical Education* (Oct. 8, 2021). Publisher: American Chemical Society. ISSN: 0021-9584. DOI: 10.1021/acs.jchemed.1c00654.

[49] Kevin C. Cassidy et al. “ProteinVR: Web-based molecular visualization in virtual reality”. In: *PLOS Computational Biology* 16.3 (Mar. 31, 2020). Ed. by Dina Schneidman-Duhovny. Publisher: Public Library of Science, e1007747. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1007747.

[50] Kui Xu et al. “VRmol: an Integrative Cloud-Based Virtual Reality System to Explore Macromolecular Structure”. In: *bioRxiv* (Mar. 28, 2019). Publisher: Cold Spring Harbor Laboratory, p. 589366. DOI: 10.1101/589366.

[51] Henrique S. Fernandes, Nuno M. F. S. A. Cerqueira, and Sérgio F. Sousa. “Developing and Using BioSIMAR, an Augmented Reality Program to Visualize and Learn about Chemical Structures in a Virtual Environment on Any Internet-Connected Device”. In: *Journal of Chemical Education* 98.5 (May 11, 2021). Publisher: American Chemical Society, pp. 1789–1794. ISSN: 0021-9584. DOI: 10.1021/acs.jchemed.0c01317.

[52] Fabio Cortés Rodríguez et al. “MoleculARweb: A Web Site for Chemistry and Structural Biology Education through Interactive Augmented Reality out of the Box in Commodity Devices”. In: *Journal of Chemical Education* 98.7 (July 13, 2021). Publisher: American Chemical Society, pp. 2243–2255. ISSN: 0021-9584. DOI: 10.1021/acs.jchemed.1c00179.

[53] Ikuo Kamei et al. “CoVR: Co-located Virtual Reality Experience Sharing for Facilitating Joint Attention via Projected View of HMD Users”. In: *SIGGRAPH Asia 2020 Emerging Technologies*. SA ’20. New York, NY, USA: Association for Computing Machinery, Dec. 4, 2020, pp. 1–2. ISBN: 978-1-4503-8110-9. DOI: 10.1145/3415255.3422883.

[54] Jan Gugenheimer et al. “ShareVR: Enabling Co-Located Experiences for Virtual Reality between HMD and Non-HMD Users”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. New York, NY, USA: Association for Computing Machinery, May 2, 2017, pp. 4021–4033. ISBN: 978-1-4503-4655-9. DOI: 10.1145/3025453.3025683.

[55] Jan Gugenheimer et al. “FaceDisplay: Towards Asymmetric Multi-User Interaction for Nomadic Virtual Reality”. In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, Apr. 19, 2018, pp. 1–13. ISBN: 978-1-4503-5620-6.

[56] Akira Ishii et al. “ReverseCAVE experience: providing reverse perspectives for sharing VR experience”. In: *SIGGRAPH Asia 2017 VR Showcase*. SA ’17. New York, NY, USA: Association for Computing Machinery, Nov. 27, 2017, pp. 1–2. ISBN: 978-1-4503-5408-0. DOI: 10.1145/3139468.3139482.

[57] Jordan M. McGraw. "Implementation and Analysis of Co-located Virtual Reality for Collaborative Scientific Data Visualization". thesis. Purdue University Graduate School, May 7, 2020. DOI: 10.25394/PGS.12252797.v1.

[58] H. J.C. Berendsen, D. van der Spoel, and R. van Drunen. "GROMACS: A message-passing parallel molecular dynamics implementation". In: *Computer Physics Communications* 91.1 (Sept. 2, 1995). Publisher: North-Holland, pp. 43–56. ISSN: 00104655. DOI: 10.1016/0010-4655(95)00042-E.

[59] B.R. Brooks et al. "CHARMM: The Biomolecular Simulation Program". In: *Journal of computational chemistry* 30.10 (July 30, 2009), pp. 1545–1614. ISSN: 0192-8651. DOI: 10.1002/jcc.21287.

[60] Noel M. O'Boyle et al. "Open Babel: An open chemical toolbox". In: *Journal of Cheminformatics* 3.1 (Oct. 7, 2011), p. 33. ISSN: 1758-2946. DOI: 10.1186/1758-2946-3-33.

[61] Naveen Michaud-Agrawal et al. "MDAnalysis: A toolkit for the analysis of molecular dynamics simulations". In: *Journal of Computational Chemistry* 32.10 (July 30, 2011). Publisher: John Wiley & Sons, Ltd, pp. 2319–2327. ISSN: 01928651. DOI: 10.1002/jcc.21787.

[62] Richard Gowers et al. "MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations". In: *Proceedings of the 15th Python in Science Conference.* 2016, pp. 98–105. DOI: 10.25080/Majora-629e541a-00e.

[63] Mark Nelson et al. "MDScope - a visual computing environment for structural biology". In: *Computer Physics Communications* 91.1 (Sept. 2, 1995). Publisher: North-Holland, pp. 111–133. ISSN: 00104655. DOI: 10.1016/0010-4655(95)00045-H.

[64] William Humphrey, Andrew Dalke, and Klaus Schulten. "VMD: Visual molecular dynamics". In: *Journal of Molecular Graphics* 14.1 (Feb. 1, 1996). Publisher: Elsevier Inc., pp. 33–38. ISSN: 02637855. DOI: 10.1016/0263-7855(96)00018-5.

[65] James C. Phillips et al. "Scalable molecular dynamics with NAMD". In: *Journal of Computational Chemistry* 26.16 (Dec. 2005), pp. 1781–1802. ISSN: 0192-8651, 1096-987X. DOI: 10.1002/jcc.20289.

[66] John E. Stone, Justin Gullingsrud, and Klaus Schulten. "A system for interactive molecular dynamics simulation". In: *Proceedings of the 2001 symposium on Interactive 3D graphics - SI3D '01.* New York, New York, USA: ACM Press, Mar. 1, 2001, pp. 191–194. ISBN: 1-58113-292-1. DOI: 10.1145/364338.364398.

[67] Aidan P. Thompson et al. "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". In: *Computer Physics Communications* 271 (Feb. 1, 2022), p. 108171. ISSN: 0010-4655. DOI: 10.1016/j.cpc.2021.108171.

[68] Joshua A. Anderson, Jens Glaser, and Sharon C. Glotzer. "HOOMD-blue: A Python package for high-performance molecular dynamics and hard particle Monte Carlo simulations". In: *Computational Materials Science* 173 (Feb. 15, 2020), p. 109363. ISSN: 0927-0256. DOI: `10.1016/j.commatsci.2019.109363`.

[69] Athina Meletiou, James Gebbie-Rayet, and Charles Laughton. "Tios: The Internet of Simulations. Turning Molecular Dynamics into a Data Streaming Web Application". In: *Journal of Chemical Information and Modeling* 59.8 (Aug. 26, 2019). Publisher: American Chemical Society, pp. 3359–3364. ISSN: 15205142. DOI: `10.1021/acs.jcim.9b00351`.

[70] Olivier Delalande et al. "Complex molecular assemblies at hand via interactive simulations". In: *Journal of Computational Chemistry* 30.15 (Nov. 30, 2009). Publisher: John Wiley and Sons Inc., pp. 2375–2387. ISSN: 01928651. DOI: `10.1002/jcc.21235`.

[71] Johanna K. S. Tiemann et al. "MDsrv: viewing and sharing molecular dynamics simulations on the web". In: *Nature Methods* 14.12 (Dec. 2017). Number: 12 Publisher: Nature Publishing Group, pp. 1123–1124. ISSN: 1548-7105. DOI: `10.1038/nmeth.4497`.

[72] *HTMoL: full-stack solution for remote access, visualization, and analysis of molecular dynamics trajectory data — SpringerLink*. URL: `https://link.springer.com/article/10.1007/s10822-018-0141-y` (visited on 03/03/2022).

[73] Matthew Wright and Adrian Freed. "Open SoundControl: A New Protocol for Communicating with Sound Synthesizers". In: International Computer Music Conference (ICMC). 1997, pp. 101–104.

[74] *gRPC*. gRPC. URL: `https://grpc.io/` (visited on 03/20/2022).

[75] Stefan Seritan et al. "TeraChem: Accelerating electronic structure and ab initio molecular dynamics with graphical processing units". In: *The Journal of Chemical Physics* 152.22 (June 14, 2020). Publisher: American Institute of Physics, p. 224110. ISSN: 0021-9606. DOI: `10.1063/5.0007615`.

[76] Ask Hjorth Larsen et al. "The atomic simulation environment—a Python library for working with atoms". In: *Journal of Physics: Condensed Matter* 29.27 (July 12, 2017). Publisher: IOP Publishing, p. 273002. ISSN: 0953-8984. DOI: `10.1088/1361-648X/aa680e`.

[77] Peter Eastman et al. "OpenMM 4: A Reusable, Extensible, Hardware Independent Library for High Performance Molecular Simulation". In: *Journal of Chemical Theory and Computation* 9.1 (Jan. 8, 2013). Publisher: American Chemical Society, pp. 461–469. ISSN: 1549-9618. DOI: `10.1021/ct300857j`.

[78] Peter Eastman et al. "OpenMM 7: Rapid development of high performance algorithms for molecular dynamics". In: *PLOS Computational Biology* 13.7 (July 26, 2017). Publisher: Public Library of Science, e1005659. ISSN: 1553-7358. DOI: `10.1371/journal.pcbi.1005659`.

[79] W.B. Streett, D.J. Tildesley, and G. Saville. "Multiple time-step methods in molecular dynamics". In: *Molecular Physics* 35.3 (Mar. 1, 1978). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00268977800100471, pp. 639–648. ISSN: 0026-8976. DOI: `10.1080/00268977800100471`.

[80] *RDKit: Open-source cheminformatics.*

[81] A. K. Rappé et al. "UFF, a Full Periodic Table Force Field for Molecular Mechanics and Molecular Dynamics Simulations". In: *Journal of the American Chemical Society* 114.25 (Dec. 1, 1992). Publisher: American Chemical Society, pp. 10024–10035. ISSN: 15205126. DOI: `10.1021/ja00051a040`.

[82] Thomas A. Halgren. "Merck molecular force field. I. Basis, form, scope, parameterization, and performance of MMFF94". In: *Journal of Computational Chemistry* 17.5 (Apr. 1, 1996). Publisher: John Wiley and Sons Inc., pp. 490–519. ISSN: 01928651. DOI: `10.1002/(SICI)1096-987X(199604)17:5/6<490::AID-JCC1>3.0.CO;2-P`.

[83] Paolo Tosco, Nikolaus Stiefl, and Gregory Landrum. "Bringing the MMFF force field to the RDKit: implementation and validation". In: *Journal of Cheminformatics* 6.1 (July 12, 2014), p. 37. ISSN: 1758-2946. DOI: `10.1186/s13321-014-0037-3`.

[84] David Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of Chemical Information and Computer Sciences* 28.1 (Feb. 1, 1988). Publisher: American Chemical Society, pp. 31–36. ISSN: 0095-2338. DOI: `10.1021/ci00057a005`.

[85] Alexander S. Rose and Peter W. Hildebrand. "NGL Viewer: a web application for molecular visualization". In: *Nucleic Acids Research* 43 (W1 July 1, 2015), W576–W579. ISSN: 0305-1048. DOI: `10.1093/nar/gkv402`.

[86] Hai Nguyen, David A Case, and Alexander S Rose. "NGLview–interactive molecular graphics for Jupyter notebooks". In: *Bioinformatics* 34.7 (Apr. 1, 2018). Publisher: Oxford Academic, pp. 1241–1242. ISSN: 1367-4803. DOI: `10.1093/BIOINFORMATICS/BTX789`.

[87] T.C. Palmer. "A language for molecular visualization". In: *IEEE Computer Graphics and Applications* 12.3 (May 1992). Conference Name: IEEE Computer Graphics and Applications, pp. 23–32. ISSN: 1558-1756. DOI: `10.1109/38.135911`.

[88] J.P.M. Hultquist and E.L. Raible. “SuperGlue: a programming environment for scientific visualization”. In: *Proceedings Visualization ’92*. Proceedings Visualization ’92. Oct. 1992, pp. 243–250. DOI: `10.1109/VISUAL.1992.235202`.

[89] Peter Rautek et al. “ViSlang: A System for Interpreted Domain-Specific Languages for Scientific Visualization”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 2388–2396. ISSN: 1941-0506. DOI: `10.1109/TVCG.2014.2346318`.

[90] Ronell Sicat et al. “DXR: A Toolkit for Building Immersive Data Visualizations”. In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 715–725. ISSN: 1941-0506. DOI: `10.1109/TVCG.2018.2865152`.

[91] Arvind Satyanarayan et al. “Vega-Lite: A Grammar of Interactive Graphics”. In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 341–350. ISSN: 1941-0506. DOI: `10.1109/TVCG.2016.2599030`.

[92] Christopher W. Wood et al. “ISAMBARD: an open-source computational environment for biomolecular analysis, modelling and design”. In: *Bioinformatics (Oxford, England)* 33.19 (Oct. 1, 2017), pp. 3043–3050. ISSN: 1367-4811. DOI: `10.1093/bioinformatics/btx352`.

[93] Andrew Leaver-Fay et al. “ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules”. In: *Methods in Enzymology* 487 (2011), pp. 545–574. ISSN: 1557-7988. DOI: `10.1016/B978-0-12-381270-4.00019-6`.

[94] The HDF Group. *Hierarchical Data Format, version 5*. 1997. URL: `https://www.hdfgroup.org/hdf5/` (visited on 01/15/2022).

[95] James A Perkins. “A history of molecular representation. Part one: 1800 to the 1960s”. In: *The Journal of Biocommunication* 31.1 (2005), p. 1.

[96] Martin Turner. “Ball and stick models for organic chemistry”. In: *Journal of Chemical Education* 48.6 (June 1, 1971). Publisher: American Chemical Society, p. 407. ISSN: 0021-9584. DOI: `10.1021/ed048p407`.

[97] Robert B. Corey and Linus Pauling. “Molecular Models of Amino Acids, Peptides, and Proteins”. In: *Review of Scientific Instruments* 24.8 (Aug. 1, 1953). Publisher: American Institute of Physics, pp. 621–627. ISSN: 0034-6748. DOI: `10.1063/1.1770803`.

[98] Walter L. Koltun. “Precision space-filling atomic models”. In: *Biopolymers* 3.6 (1965). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bip.360030606, pp. 665–679. ISSN: 1097-0282. DOI: `10.1002/bip.360030606`.

[99] Martin Kemp. “Kendrew constructs; Geis gazes”. In: *Nature* 396.6711 (Dec. 1998). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 6711 Primary_atype: Comments & Opinion Publisher: Nature Publishing Group, pp. 525–525. ISSN: 1476-4687. DOI: 10.1038/25019.

[100] Eric Francoeur. “Beyond dematerialization and inscription: Does the materiality of molecular models really matter?” In: *HYLE – International Journal for Philosophy of Chemistry* 6.1 (2000), pp. 63–84.

[101] C. K. Johnson. *ORTEP: a FORTRAN Thermal-Ellipsoid Plot Program for Crystal Structure Illustrations.* ORNL-3794. Oak Ridge National Lab., Tenn., Jan. 6, 1965.

[102] C. K. Johnson. *ORTEP-II: a FORTRAN Thermal-Ellipsoid Plot Program for crystal structure illustrations.* ORNL-5138. Oak Ridge National Lab., Tenn., Jan. 3, 1976. DOI: 10.2172/7364501.

[103] M N Burnett and C. K. Johnson. *ORTEP-III: Oak Ridge Thermal Ellipsoid Plot Program for crystal structure illustrations.* ORNL-6895. Oak Ridge National Lab. (ORNL), Oak Ridge, TN, Jan. 7, 1996.

[104] R Sayle and A Bissell. “RasMol: A Program for Fast Realistic Rendering of Molecular Structures with Shadows”. In: *Proceedings of the 10th Eurographics UK ’92 Conference.* 10th Eurographics UK ’92. University of Edinburgh, Scotland, 1992.

[105] R Sayle and E. James Milner-White. “RASMOL: biomolecular graphics for all”. In: *Trends in Biochemical Sciences* 20.9 (Sept. 1, 1995). Publisher: Elsevier Current Trends, pp. 374–376. ISSN: 09680004. DOI: 10.1016/S0968-0004(00)89080-5.

[106] E. A. Merritt and D. J. Bacon. “Raster3D: photorealistic molecular graphics”. In: *Methods in Enzymology* 277 (1997), pp. 505–524. ISSN: 0076-6879. DOI: 10.1016/s0076-6879(97)77028-9.

[107] Per J. Kraulis. “MOLSCRIPT: a program to produce both detailed and schematic plots of protein structures”. In: *Journal of Applied Crystallography* 24.5 (Oct. 1, 1991), pp. 946–950. ISSN: 00218898. DOI: 10.1107/S0021889891004399.

[108] Eric F. Pettersen et al. “UCSF Chimera?A visualization system for exploratory research and analysis”. In: *Journal of Computational Chemistry* 25.13 (Oct. 1, 2004). Publisher: John Wiley & Sons, Ltd, pp. 1605–1612. ISSN: 0192-8651. DOI: 10.1002/jcc.20084.

[109] Thomas D. Goddard et al. “UCSF ChimeraX: Meeting modern challenges in visualization and analysis”. In: *Protein Science* 27.1 (Jan. 1, 2018). Publisher: Blackwell Publishing Ltd, pp. 14–25. ISSN: 09618368. DOI: 10.1002/pro.3235.

[110] Angel Herráez. “Biomolecules in the computer: Jmol to the rescue”. In: *Biochemistry and Molecular Biology Education* 34.4 (2006). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/bm pp. 255–261. ISSN: 1539-3429. DOI: 10.1002/bmb.2006.494034042644.

[111] David Sehnal et al. “Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures”. In: *Nucleic Acids Research* 49 (W1 July 2, 2021), W431–W437. ISSN: 0305-1048. DOI: 10.1093/nar/gkab314.

[112] Jane S. Richardson. “Schematic drawings of protein structures”. In: *Methods in Enzymology*. Vol. 115. Issue: C ISSN: 15577988. Academic Press, Jan. 1, 1985, pp. 359–380. DOI: 10.1016/0076-6879(85)15026-3.

[113] B. Kozlíková et al. “Visualization of Biomolecular Structures: State of the Art Revisited”. In: *Computer Graphics Forum* 36.8 (Dec. 1, 2017). Publisher: Blackwell Publishing Ltd, pp. 178–204. ISSN: 01677055. DOI: 10.1111/cgf.13072.

[114] B. Lee and F. M. Richards. “The interpretation of protein structures: Estimation of static accessibility”. In: *Journal of Molecular Biology* 55.3 (Feb. 14, 1971), 379–IN4. ISSN: 0022-2836. DOI: 10.1016/0022-2836(71)90324-X.

[115] F M Richards. “Areas, Volumes, Packing, and Protein Structure”. In: *Annual Review of Biophysics and Bioengineering* 6.1 (June 28, 1977). Publisher: Annual Reviews 4139 El Camino Way, P.O. Box 10139, Palo Alto, CA 94303-0139, USA, pp. 151–176. ISSN: 0084-6589. DOI: 10.1146/annurev.bb.06.060177.001055.

[116] J. Greer and B. L. Bush. “Macromolecular shape and surface maps by solvent exclusion”. In: *Proceedings of the National Academy of Sciences of the United States of America* 75.1 (Jan. 1978), pp. 303–307. ISSN: 0027-8424. DOI: 10.1073/pnas.75.1.303.

[117] Maxim Totrov and Ruben Abagyan. “The Contour-Buildup Algorithm to Calculate the Analytical Molecular Surface”. In: *Journal of Structural Biology* 116.1 (Jan. 1, 1996), pp. 138–143. ISSN: 1047-8477. DOI: 10.1006/jsbi.1996.0022.

[118] Norbert Lindow et al. “Accelerated Visualization of Dynamic Molecular Surfaces”. In: *Computer Graphics Forum* 29.3 (Aug. 12, 2010). Publisher: John Wiley & Sons, Ltd (10.1111), pp. 943–952. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2009.01693.x.

[119] Michael Krone, Sebastian Grottel, and Thomas Ertl. “Parallel Contour-Buildup algorithm for the molecular surface”. In: *2011 IEEE Symposium on Biological Data Visualization (BioVis)*. 2011 IEEE Symposium on Biological Data Visualization (BioVis). Oct. 2011, pp. 17–22. DOI: 10.1109/BioVis.2011.6094043.

[120] M F Sanner, A J Olson, and J C Spehner. “Reduced surface: an efficient way to compute molecular surfaces.” In: *Biopolymers* 38.3 (Mar. 1996), pp. 305–20. ISSN: 0006-3525. DOI: 10.1002/(SICI)1097-0282(199603)38:3%3C305::AID-BIP4%3E3.0.CO;2-Y.

[121] M. F. Sanner and A. J. Olson. “Real time surface reconstruction for moving molecular fragments”. In: *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing* (1997), pp. 385–396. ISSN: 2335-6928.

[122] Michael Krone, Katrin Bidmon, and Thomas Ertl. “Interactive Visualization of Molecular Surface Dynamics”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 1391–1398. ISSN: 1941-0506. DOI: 10.1109/TVCG.2009.157.

[123] William E. Lorensen and Harvey E. Cline. “Marching cubes: A high resolution 3D surface construction algorithm”. In: *Proceedings of the 14th annual conference on Computer graphics and interactive techniques.* SIGGRAPH ’87. New York, NY, USA: Association for Computing Machinery, Aug. 1, 1987, pp. 163–169. ISBN: 978-0-89791-227-3. DOI: 10.1145/37401.37422.

[124] H. Edelsbrunner. “Deformable smooth surface design”. In: *Discrete and Computational Geometry* 21.1 (1999). Publisher: Springer New York, pp. 87–115. ISSN: 01795376. DOI: 10.1007/PL00009412.

[125] Matthieu Chavent, Bruno Levy, and Bernard Maigret. “MetaMol: High-quality visualization of molecular skin surface”. In: *Journal of Molecular Graphics and Modelling* 27.2 (Sept. 2008). Publisher: J Mol Graph Model, pp. 209–216. ISSN: 10933263. DOI: 10.1016/j.jmgm.2008.04.007.

[126] Stefan Bruckner. “Dynamic Visibility-Driven Molecular Surfaces”. In: *Computer Graphics Forum* 38.2 (May 7, 2019). Publisher: Blackwell Publishing Ltd, pp. 317–329. ISSN: 0167-7055. DOI: 10.1111/cgf.13640.

[127] Norbert Lindow, Daniel Baum, and Hans-Christian Hege. “Ligand Excluded Surface: A New Type of Molecular Surface”. In: *IEEE Transactions on Visualization and Computer Graphics* 20.12 (Dec. 2014). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 2486–2495. ISSN: 1941-0506. DOI: 10.1109/TVCG.2014.2346404.

[128] Steve Marschner and Peter Shirley. *Fundamentals of Computer Graphics.* Fourth edition. Nov. 18, 2015. 748 pp.

[129] Jörg Schmittler, Ingo Wald, and Philipp Slusallek. “SaarCOR: a hardware architecture for ray tracing”. In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware.* HWWS ’02. Goslar, DEU: Eurographics Association, Sept. 1, 2002, pp. 27–36. ISBN: 978-1-58113-580-0.

[130] Nate Oh. *NVIDIA Announces RTX Technology: Real Time Ray Tracing Acceleration for Volta GPUs and Later.* Mar. 19, 2018. URL: https://www.anandtech.com/show/12546/nvidia-unveils-rtx-technology-real-time-ray-tracing-acceleration-for-volta-gpus-and-later (visited on 05/22/2022).

[131] *Introducing the NVIDIA RTX Ray Tracing Platform.* NVIDIA Developer. Running Time: 64. Mar. 6, 2018. URL: https://developer.nvidia.com/rtx/ray-tracing (visited on 05/22/2022).

[132] Lukas Marsalek et al. “Real-Time Ray Tracing of Complex Molecular Scenes”. In: *2010 14th International Conference Information Visualisation.* 2010 14th International Conference Information Visualisation. ISSN: 2375-0138. July 2010, pp. 239–245. DOI: 10.1109/IV.2010.43.

[133] John E. Stone, William R. Sherman, and Klaus Schulten. “Immersive molecular visualization with omnidirectional stereoscopic ray tracing and remote rendering”. In: *Proceedings - 2016 IEEE 30th International Parallel and Distributed Processing Symposium, IPDPS 2016.* Institute of Electrical and Electronics Engineers Inc., July 18, 2016, pp. 1048–1057. ISBN: 978-1-5090-2140-6. DOI: 10.1109/IPDPSW.2016.121.

[134] N. Lindow, D. Baum, and H.-C. Hege. “Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale”. In: *Computer Graphics Forum* 31.3 (June 1, 2012). Publisher: Wiley, pp. 1325–1334. ISSN: 01677055. DOI: 10.1111/j.1467-8659.2012.03128.x.

[135] Finian Mwalongo et al. “Visualization of molecular structures using state-of-the-art techniques in WebGL”. In: *Proceedings of the 19th International ACM Conference on 3D Web Technologies, Web3D 2014.* New York, New York, USA: Association for Computing Machinery, 2014, pp. 133–141. ISBN: 978-1-4503-3015-2. DOI: 10.1145/2628588.2628597.

[136] Stefan Guthe, Stefan Gumhold, and Wolfgang Straßer. “Interactive Visualization of Volumetric Vector Fields Using Texture Based Particles”. In: Journal of WSCG. Vol. 10. Jan. 1, 2002, pp. 33–42.

[137] Rodrigo Toledo and Bruno Lévy. *Extending the graphic pipeline with new GPU-accelerated primitives.* INRIA, 2004.

[138] Eric Risser. “True Impostors”. In: *GPU Gems 3.* Addison-Wesley Professional, 2007. ISBN: 978-0-321-51526-1.

[139] Chandrajit Bajaj et al. “TexMol: Interactive visual exploration of large flexible multi-component molecular complexes”. In: *IEEE Visualization 2004 - Proceedings, VIS 2004.* 2004, pp. 243–250. ISBN: 0-7803-8788-0. DOI: 10.1109/visual.2004.103.

[140] Guido Reina and Thomas Ertl. “Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization.” In: Visualization (EuroVis), EG/IEEE VGTC Symposium on. Jan. 1, 2005, pp. 177–182. DOI: 10.2312/VisSym/EuroVis05/177-182.

[141] Christian Sigg et al. “GPU-based ray-casting of quadratic surfaces”. In: *SPBG’06: Proceedings of the 3rd Eurographics / IEEE VGTC conference on Point-Based Graphics.* July 2006, pp. 59–65. DOI: 10.5555/2386388.2386396.

[142] Matthieu Chavent et al. “GPU-accelerated atom and dynamic bond visualization using hyperballs: A unified algorithm for balls, sticks, and hyperboloids”. In: *Journal of Computational Chemistry* 32.13 (Oct. 2011), pp. 2924–2935. ISSN: 01928651. DOI: 10.1002/jcc.21861.

[143] Pranav D. Bagur, Nithin Shivashankar, and Vijay Natarajan. “Improved quadric surface impostors for large bio-molecular visualization”. In: *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing - ICVGIP ’12.* New York, New York, USA: ACM Press, 2012, pp. 1–8. ISBN: 978-1-4503-1660-6. DOI: 10.1145/2425333.2425366.

[144] Mathieu Le Muzic et al. “cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets”. In: *Eurographics Workshop on Visual Computing for Biology and Medicine.* Accepted: 2015-09-14T04:48:59Z ISSN: 2070-5786. The Eurographics Association, 2015. ISBN: 978-3-905674-82-8.

[145] Adam Jurčík et al. “Accelerated visualization of transparent molecular surfaces in molecular dynamics”. In: *IEEE Pacific Visualization Symposium.* Vol. 2016-May. ISSN: 21658773. IEEE Computer Society, May 4, 2016, pp. 112–119. ISBN: 978-1-5090-1451-4. DOI: 10.1109/PACIFICVIS.2016.7465258.

[146] Sebastian Grottel et al. “Particle-based Rendering for Porous Media”. In: *Proceedings of SIGRAD 2010.* Jan. 1, 2010, pp. 45–51.

[147] Sebastian Grottel, Guido Reina, and Thomas Ertl. “Optimized Data Transfer for Time-dependent, GPU-based Glyphs”. In: Proceedings of IEEE Pacific Visualization Symposium. Vol. 2009. Apr. 1, 2009, pp. 65–72. DOI: 10.1109/PACIFICVIS.2009.4906839.

[148] Stefan Gumhold. “Splatting Illuminated Ellipsoids with Depth Correction.” In: *Proceedings of the Vision, Modeling, and Visualization Conference 2003 (VMV 2003).* Jan. 1, 2003, pp. 245–252.

[149] “Real Closed Fields”. In: *Algorithms in Real Algebraic Geometry.* Ed. by Saugata Basu, Richard Pollack, and Marie-Françoise Roy. Algorithms and Computation in Mathematics. Berlin, Heidelberg: Springer, 2006, pp. 29–82. ISBN: 978-3-540-33099-8. DOI: 10.1007/3-540-33099-2_3.

[150] David Sehnal et al. “Mol*: towards a common library and tools for web molecular graphics”. In: *Proceedings of the workshop on molecular graphics and visual analysis of molecular data.* Ed. by J. Byška, M. Krone, and B. Sommer. The Eurographics Association, 2018, pp. 29–33. DOI: 10.2312/molva.20181103.

[151] Matthieu Dreher et al. “ExaViz: a flexible framework to analyse, steer and interact with molecular dynamics simulations”. In: *Faraday Discussions* 169.0 (Oct. 23, 2014). Publisher: The Royal Society of Chemistry, pp. 119–142. ISSN: 1364-5498. DOI: 10.1039/C3FD00142C.

[152] Sergei Borzov. “Use of signed distance functions for the definition of protein cartoon representation”. Masters. Brno: Masaryk University, 2021.

[153] John C. Hart. “Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces”. In: *The Visual Computer* 12.10 (Dec. 1, 1996), pp. 527–545. ISSN: 1432-2315. DOI: 10.1007/s003710050084.

[154] Benjamin Keinert et al. “Enhanced Sphere Tracing”. In: Accepted: 2014-12-16T07:17:30Z. The Eurographics Association, 2014. ISBN: 978-3-905674-72-9. DOI: 10.2312/stag.20141233.

[155] Peter Atkins, Julio de Paula, and James Keeler. *Atkins’ Physical Chemistry*. Eleventh Edition. Oxford, New York: Oxford University Press, Dec. 28, 2017. 944 pp. ISBN: 978-0-19-876986-6.

[156] Mark Weller et al. *Inorganic Chemistry*. Seventh Edition. Oxford, New York: Oxford University Press, June 6, 2018. 968 pp. ISBN: 978-0-19-876812-8.

[157] Andy Burrows et al. *Chemistry^3: Introducing inorganic, organic and physical chemistry*. Oxford ; New York: OUP Oxford, Apr. 9, 2009. 1416 pp. ISBN: 978-0-19-927789-6.

[158] James D. Foley et al. *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, 1996. 1294 pp. ISBN: 978-0-201-84840-3.

[159] David Eisenberg. “The discovery of the -helix and -sheet, the principal structural features of proteins”. In: *Proceedings of the National Academy of Sciences* 100.20 (Sept. 30, 2003). Publisher: National Academy of Sciences, pp. 11207–11210. ISSN: 0027-8424. DOI: 10.1073/pnas.2034522100.

[160] L. Pauling and R. B. Corey. “The Pleated Sheet, A New Layer Configuration of Polypeptide Chains”. In: *Proceedings of the National Academy of Sciences* 37.5 (May 1, 1951). Publisher: National Academy of Sciences, pp. 251–256. ISSN: 0027-8424. DOI: 10.1073/pnas.37.5.251.

[161] L. Pauling, R. B. Corey, and H. R. Branson. “The structure of proteins: Two hydrogen-bonded helical configurations of the polypeptide chain”. In: *Proceedings of the National Academy of Sciences* 37.4 (Apr. 1, 1951). Publisher: National Academy of Sciences, pp. 205–211. ISSN: 0027-8424. DOI: 10.1073/pnas.37.4.205.

[162] G. N. Ramachandran, C. Ramakrishnan, and V. Sasisekharan. “Stereochemistry of polypeptide chain configurations”. In: *Journal of Molecular Biology* 7.1 (July 1, 1963), pp. 95–99. ISSN: 0022-2836. DOI: 10.1016/S0022-2836(63)80023-6.

[163] G. N. Ramachandran and V. Sasisekharan. “Conformation of Polypeptides and Proteins”. In: *Advances in Protein Chemistry*. Ed. by C. B. Anfinsen et al. Vol. 23. Academic Press, Jan. 1, 1968, pp. 283–437. DOI: 10.1016/S0065-3233(08)60402-7.

[164] Frances C. Bernstein et al. "The protein data bank: A computer-based archival file for macromolecular structures". In: *Journal of Molecular Biology* 112.3 (May 25, 1977). Publisher: Academic Press, pp. 535–542. ISSN: 00222836. DOI: 10.1016/S0022-2836(77)80200-3.

[165] Michael Levitt and Jonathan Greer. "Automatic identification of secondary structure in globular proteins". In: *Journal of Molecular Biology* 114.2 (Aug. 5, 1977). Publisher: Academic Press, pp. 181–239. ISSN: 00222836. DOI: 10.1016/0022-2836(77)90207-8.

[166] Wolfgang Kabsch and Christian Sander. "Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features". In: *Biopolymers* 22.12 (Dec. 1, 1983). Publisher: John Wiley & Sons, Ltd, pp. 2577–2637. ISSN: 0006-3525. DOI: 10.1002/bip.360221211.

[167] Dmitrij Frishman and Patrick Argos. "Knowledge-based protein secondary structure assignment". In: *Proteins: Structure, Function, and Genetics* 23.4 (Dec. 1, 1995). Publisher: John Wiley & Sons, Ltd, pp. 566–579. ISSN: 0887-3585. DOI: 10.1002/prot.340230412.

[168] F. M. Richards and C. E. Kundrot. "Identification of structural motifs from protein coordinate data: secondary structure and first-level supersecondary structure". In: *Proteins* 3.2 (1988), pp. 71–84. ISSN: 0887-3585. DOI: 10.1002/prot.340030202.

[169] Juliette Martin et al. "Protein secondary structure assignment revisited: A detailed analysis of different assignment methods". In: *BMC Structural Biology* 5.1 (Sept. 15, 2005). Publisher: BioMed Central, p. 17. ISSN: 14726807. DOI: 10.1186/1472-6807-5-17.

[170] Jane S. Richardson, David C. Richardson, and David S. Goodsell. "Seeing the pdb". In: *Journal of Biological Chemistry* 296.0 (May 2021). Publisher: Elsevier, p. 100742. ISSN: 1083351X. DOI: 10.1016/J.JBC.2021.100742.

[171] M.C. Escher. *Spirals.* 1953.

[172] M.C. Escher. *Rind.* 1955.

[173] M.C. Escher. *Bonds of Union.* 1956.

[174] M.C. Escher. *Sphere Spirals.* 1958.

[175] Jane S. Richardson. "The Anatomy and Taxonomy of Protein Structure". In: *Advances in Protein Chemistry.* Vol. 34. ISSN: 00653233. Academic Press, Jan. 1, 1981, pp. 167–339. DOI: 10.1016/S0065-3233(08)60520-3.

[176] A. Lesk and K. Hardman. "Computer-generated schematic diagrams of protein structures". In: *Science* 216.4545 (Apr. 30, 1982). Publisher: American Association for the Advancement of Science, pp. 539–540. ISSN: 0036-8075. DOI: 10.1126/science.7071602.

[177] Mike Carson and Charles E. Bugg. “Algorithm for ribbon models of proteins”. In: *Journal of Molecular Graphics* 4.2 (June 1, 1986). Publisher: Elsevier, pp. 121–122. ISSN: 02637855. DOI: 10.1016/0263-7855(86)80010-8.

[178] Mike Carson. “Ribbon models of macromolecules”. In: *Journal of Molecular Graphics* 5.2 (June 1, 1987). Publisher: Elsevier, pp. 103–106. ISSN: 02637855. DOI: 10.1016/0263-7855(87)80010-3.

[179] M. Carson and C. E. Bugg. “BSRIBBON – program for producing 3D ribbon models of macromolecules suitable for interactive graphics display”. In: *Journal of Applied Crystallography* 21.5 (Oct. 1, 1988). Publisher: Wiley-Blackwell, pp. 578–578. ISSN: 0021-8898. DOI: 10.1107/S0021889888005825.

[180] Mike Carson. “RIBBONS 2.0”. In: *Journal of Applied Crystallography* 24.5 (Oct. 1, 1991). Publisher: International Union of Crystallography, pp. 958–961. ISSN: 00218898. DOI: 10.1107/S0021889891007240.

[181] Mike Carson. “Ribbons”. In: *Methods in Enzymology*. Vol. 277. ISSN: 00766879. Academic Press, Jan. 1, 1997, pp. 493–505. DOI: 10.1016/S0076-6879(97)77027-7.

[182] Lawrence D. Bergman, Jane S. Richardson, and David C. Richardson. “An algorithm for smoothly tessellating β-sheet structures in proteins”. In: *Journal of Molecular Graphics* 13.1 (Feb. 1, 1995). Publisher: Elsevier, pp. 36–45. ISSN: 02637855. DOI: 10.1016/0263-7855(94)00003-B.

[183] J. P. Priestle. “RIBBON : a stereo cartoon drawing program for proteins”. In: *Journal of Applied Crystallography* 21.5 (Oct. 1, 1988). Publisher: Wiley-Blackwell, pp. 572–576. ISSN: 0021-8898. DOI: 10.1107/S0021889888005746.

[184] Jane S. Richardson. “Early ribbon drawings of proteins”. In: *Nature Structural Biology* 7.8 (Aug. 2000). Publisher: Nature Publishing Group, pp. 624–625. ISSN: 10728368. DOI: 10.1038/77912.

[185] P Hermosilla et al. “Instant visualization of secondary structures of molecular models”. In: *Eurographics Workshop on Visual Computing for Biology and Medicine*. Publication Title: VCBM 15: Eurographics Workshop on Visual Computing for Biology and Medicine. The Eurographics Association, 2015, pp. 51–60. ISBN: 978-3-905674-82-8. DOI: 10.2312/VCBM.20152018.

[186] Matus Zamborsky, Tibor Szabo, and Barbora Kozlikova. “Dynamic visualization of protein secondary structures”. In: *Proceedings of the 13th Central European Seminar on Computer Graphics (CESCG)*. Corpus ID: 7882516. 2009, pp. 147–152.

[187] Simon Cross et al. “Visualisation of cyclic and multi-branched molecules with VMD”. In: *Journal of Molecular Graphics and Modelling* 28.2 (Sept. 1, 2009). Publisher: Elsevier, pp. 131–139. ISSN: 10933263. DOI: 10.1016/j.jmgm.2009.04.010.

[188] Lincong Wang et al. “An accurate model for biomolecular helices and its application to helix visualization”. In: *PLoS ONE* 10.6 (June 30, 2015). Ed. by Freddie Salsbury. Publisher: Public Library of Science, e0129653. ISSN: 19326203. DOI: 10.1371/journal.pone.0129653.

[189] Edwin Catmull and Raphael Rom. “A Class of Locally Interpolating Splines”. In: *Computer Aided Geometric Design*. Elsevier, Jan. 1, 1974, pp. 317–326. DOI: 10.1016/B978-0-12-079050-0.50020-5.

[190] Doris H. U. Kochanek and Richard H. Bartels. “Interpolating splines with local tension, continuity, and bias control”. In: *ACM SIGGRAPH Computer Graphics* 18 (July 1984). ISBN: 0897911385, pp. 33–41. DOI: 10.1145/800031.808575.

[191] Gerald E. Farin. *NURBS for Curve and Surface Design*. USA: Society for Industrial and Applied Mathematics, 1991. 161 pp. ISBN: 978-0-89871-286-5.

[192] G. Farin. “From conics to NURBS: A tutorial and survey”. In: *IEEE Computer Graphics and Applications* 12.5 (Sept. 1992). Conference Name: IEEE Computer Graphics and Applications, pp. 78–86. ISSN: 1558-1756. DOI: 10.1109/38.156017.

[193] Gregory S. Couch, Donna K. Hendrix, and Thomas E. Ferrin. “Nucleic acid visualization with UCSF Chimera”. In: *Nucleic Acids Research* 34.4 (Feb. 1, 2006), e29. ISSN: 0305-1048. DOI: 10.1093/nar/gnj031.

[194] C. Massire, C. Gaspin, and E. Westhof. “DRAWNA: A program for drawing schematic views of nucleic acids”. In: *Journal of Molecular Graphics* 12.3 (Sept. 1, 1994), pp. 201–206. ISSN: 0263-7855. DOI: 10.1016/0263-7855(94)80088-X.

[195] Lars Offen and Dieter Fellner. “BioBrowser — Visualization of and Access to Macro-Molecular Structures”. In: Springer, Berlin, Heidelberg, 2008, pp. 257–273. DOI: 10.1007/978-3-540-72630-2_15.

[196] Stephen V. Evans. “SETOR: Hardware-lighted three-dimensional solid model representations of macromolecules”. In: *Journal of Molecular Graphics* 11.2 (June 1, 1993), pp. 134–138. ISSN: 0263-7855. DOI: 10.1016/0263-7855(93)87009-T.

[197] Matthew Z. Tien et al. *PeptideBuilder*. Version 1.1.0.

[198] Michael Krone, Katrin Bidmon, and Thomas Ertl. “GPU-based visualisation of protein secondary structure”. In: *Theory and Practice of Computer Graphics 2008, TPCG 2008 - Eurographics UK Chapter Proceedings*. Ed. by Ik Soo Lim and Wen Tang. The Eurographics Association, 2008, pp. 115–122. ISBN: 978-3-905673-67-8. DOI: 10.2312/LocalChapterEvents/TPCG/TPCG08/115-122.

[199] Fopke Klok. “Two moving coordinate frames for sweeping along a 3D trajectory”. In: *Computer Aided Geometric Design* 3.3 (Nov. 1, 1986), pp. 217–229. ISSN: 0167-8396. DOI: 10.1016/0167-8396(86)90039-7.

[200] Nataraj Akkiraju et al. "Viewing geometric protein structures from inside a CAVE". In: *IEEE Computer Graphics and Applications* 16.4 (July 1996). Publisher: IEEE Computer Society, pp. 58–61. ISSN: 02721716. DOI: 10.1109/38.511855.

[201] M. Krone et al. "Visual Analysis of Biomolecular Cavities: State of the Art". In: *Computer Graphics Forum* 35.3 (June 1, 2016). Publisher: Blackwell Publishing Ltd, pp. 527–551. ISSN: 01677055. DOI: 10.1111/cgf.12928.

[202] Xavier Martinez et al. "Molecular Graphics: Bridging Structural Biologists and Computer Scientists". In: *Structure* 27.11 (Nov. 5, 2019). Publisher: Cell Press, pp. 1617–1623. ISSN: 18784186. DOI: 10.1016/j.str.2019.09.001.

[203] Cem Yuksel. "A Class of C 2 Interpolating Splines". In: *ACM Transactions on Graphics* 39.5 (Sept. 4, 2020). Publisher: Association for Computing Machinery (ACM), pp. 1–14. ISSN: 0730-0301. DOI: 10.1145/3400301.

[204] Norman L. Allinger. "Conformational analysis. 130. MM2. A hydrocarbon force field utilizing V1 and V2 torsional terms". In: *Journal of the American Chemical Society* 99.25 (Dec. 1977). Publisher: UTC, pp. 8127–8134. ISSN: 0002-7863. DOI: 10.1021/ja00467a001.

[205] Norman L. Allinger, Young H. Yuh, and Jenn-Huei Lii. "Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 1". In: *Journal of the American Chemical Society* 111.23 (1989). Publisher: American Chemical Society, pp. 8551–8566. ISSN: 0002-7863. DOI: 10.1021/ja00205a001.

[206] Jenn-Huei Lii and Norman L. Allinger. "Molecular Mechanics. The MM3 Force Field for Hydrocarbons. 2. Vibrational Frequencies and Thermodynamics". In: *Journal of the American Chemical Society* 111.23 (1989). Publisher: American Chemical Society, pp. 8566–8575. ISSN: 15205126. DOI: 10.1021/ja00205a002.

[207] Jenn-Huei Lii and Norman L. Allinger. "Molecular mechanics. The MM3 force field for hydrocarbons. 3. The van der Waals' potentials and crystal data for aliphatic and aromatic hydrocarbons". In: *Journal of the American Chemical Society* 111.23 (Nov. 1989). Publisher: UTC, pp. 8576–8582. ISSN: 0002-7863. DOI: 10.1021/ja00205a003.

[208] Norman L. Allinger, Fanbing Li, and Liqun Yan. "Molecular Mechanics. The MM3 Force Field for Alkenes". In: *Journal of Computational Chemistry* 11.7 (1990). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/jcc.540110708, pp. 848–867. ISSN: 1096-987X. DOI: 10.1002/jcc.540110708.

[209] Norman L. Allinger, Kuohsiang Chen, and Jenn-Huei Lii. "An improved force field (MM4) for saturated hydrocarbons". In: *Journal of Computational Chemistry* 17.5 (Apr. 1, 1996). Publisher: John Wiley and Sons Inc., pp. 642–668. ISSN: 01928651. DOI: 10.1002/(SICI)1096-987X(199604)17:5/6<642::AID-JCC6>3.0.CO;2-U.

[210] Neysa Nevins, Kuohsiang Chen, and Norman L. Allinger. "Molecular mechanics (MM4) calculations on alkenes". In: *Journal of Computational Chemistry* 17.5 (1996). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291096-987X%28199604%2917%3A5/6%3C66 JCC7%3E3.0.CO%3B2-S, pp. 669–694. ISSN: 1096-987X. DOI: `10.1002/(SICI)1096-987X(199604)17:5/6<669::AID-JCC7>3.0.CO;2-S`.

[211] Neysa Nevins and Norman L. Allinger. "Molecular mechanics (MM4) vibrational frequency calculations for alkenes and conjugated hydrocarbons". In: *Journal of Computational Chemistry* 17.5 (1996). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291096-987X%28199604%2917%3A5/6%3C730%3A%3AAID-JCC9%3E3.0.CO%3B2-V, pp. 730–746. ISSN: 1096-987X. DOI: `10.1002/(SICI)1096-987X(199604)17:5/6<730::AID-JCC9>3.0.CO;2-V`.

[212] Junmei Wang et al. "Development and testing of a general amber force field". In: *Journal of Computational Chemistry* 25.9 (July 15, 2004). Publisher: John Wiley and Sons Inc., pp. 1157–1174. ISSN: 0192-8651. DOI: `10.1002/jcc.20035`.

[213] Hendrik Heinz et al. "Force Field for Mica-Type Silicates and Dynamics of Octadecylammonium Chains Grafted to Montmorillonite". In: *Chemistry of Materials* 17.23 (Nov. 1, 2005). Publisher: American Chemical Society, pp. 5658–5669. ISSN: 0897-4756. DOI: `10.1021/cm0509328`.

[214] Alex Tek et al. "Advances in Human-Protein Interaction - Interactive and Immersive Molecular Simulations". In: *Protein-Protein Interactions*. Publication Title: Protein-Protein Interactions - Computational and Experimental Tools. IntechOpen, Mar. 30, 2012. ISBN: 978-953-51-0397-4. DOI: `10.5772/36568`.

[215] Helen M. Deeks et al. "Interactive Molecular Dynamics in Virtual Reality Is an Effective Tool for Flexible Substrate and Inhibitor Docking to the SARS-CoV-2 Main Protease". In: *Journal of Chemical Information and Modeling* 60.12 (Dec. 28, 2020). Publisher: American Chemical Society, pp. 5803–5814. ISSN: 1549-9596. DOI: `10.1021/acs.jcim.0c01030`.

[216] Matthieu Dreher et al. "Interactive molecular dynamics: Scaling up to large systems". In: *Procedia Computer Science*. Vol. 18. ISSN: 18770509. Elsevier B.V., Jan. 1, 2013, pp. 20–29. DOI: `10.1016/j.procs.2013.05.165`.

[217] Nathan Luehr, Alex G.B. Jin, and Todd J. Martínez. "Ab Initio Interactive Molecular Dynamics on Graphical Processing Units (GPUs)". In: *Journal of Chemical Theory and Computation* 11.10 (Sept. 2, 2015). Publisher: American Chemical Society, pp. 4536–4544. ISSN: 15499626. DOI: `10.1021/acs.jctc.5b00419`.

[218] Mark C. Surles et al. "Sculpting proteins interactively: Continual energy minimization embedded in a graphical modeling system". In: *Protein Science* 3.2 (1994). Publisher: John Wiley & Sons, Ltd, pp. 198–210. ISSN: 09618368. DOI: `10.1002/pro.5560030205`.

[219] J. Leech, J.F. Prins, and J. Hermans. “SMD: visual steering of molecular dynamics for protein design”. In: *IEEE Computational Science and Engineering* 3.4 (1996). Conference Name: IEEE Computational Science and Engineering, pp. 38–45. ISSN: 1558-190X. DOI: 10.1109/99.556511.

[220] H. Grubmüller, B. Heymann, and P. Tavan. “Ligand binding: molecular mechanics calculation of the streptavidin-biotin rupture force”. In: *Science (New York, N.Y.)* 271.5251 (Feb. 16, 1996), pp. 997–999. ISSN: 0036-8075. DOI: 10.1126/science.271.5251.997.

[221] Xiongwu Wu and Shaomeng Wang. “Self-Guided Molecular Dynamics Simulation for Efficient Conformational Search”. In: *The Journal of Physical Chemistry B* 102.37 (Sept. 1, 1998). Publisher: American Chemical Society, pp. 7238–7250. ISSN: 1520-6106. DOI: 10.1021/jp9817372.

[222] Xiongwu Wu, Shaomeng Wang, and Bernard R. Brooks. “Direct Observation of the Folding and Unfolding of a β-Hairpin in Explicit Water through Computer Simulation”. In: *Journal of the American Chemical Society* 124.19 (May 1, 2002). Publisher: American Chemical Society, pp. 5282–5283. ISSN: 0002-7863. DOI: 10.1021/ja0257321.

[223] Xiongwu Wu and Shaomeng Wang. “Helix Folding of an Alanine-Based Peptide in Explicit Water”. In: *The Journal of Physical Chemistry B* 105.11 (Mar. 1, 2001). Publisher: American Chemical Society, pp. 2227–2235. ISSN: 1520-6106. DOI: 10.1021/jp004048a.

[224] Wataru Shinoda and Masuhiro Mikami. “Self-guided molecular dynamics in the isothermal–isobaric ensemble”. In: *Chemical Physics Letters* 335.3 (Feb. 23, 2001), pp. 265–272. ISSN: 0009-2614. DOI: 10.1016/S0009-2614(01)00054-9.

[225] Xiongwu Wu and Bernard R. Brooks. “Self-guided Langevin dynamics simulation method”. In: *Chemical Physics Letters* 381.3 (Nov. 14, 2003), pp. 512–518. ISSN: 0009-2614. DOI: 10.1016/j.cplett.2003.10.013.

[226] Michal Koutek et al. “Virtual Spring Manipulators for Particle Steering in Molecular Dynamics on the ResponsiveWorkbench”. In: Accepted: 2014-01-27T10:15:24Z ISSN: 1727-530X. The Eurographics Association, 2002. ISBN: 978-1-58113-535-0. DOI: 10.2312/EGVE/EGVE02/053-062.

[227] G. M. Torrie and J. P. Valleau. “Nonphysical sampling distributions in Monte Carlo free-energy estimation: Umbrella sampling”. In: *Journal of Computational Physics* 23.2 (Feb. 1, 1977), pp. 187–199. ISSN: 0021-9991. DOI: 10.1016/0021-9991(77)90121-8.

[228] Alessandro Laio and Michele Parrinello. “Escaping free-energy minima”. In: *Proceedings of the National Academy of Sciences* 99.20 (Oct. 2002). Publisher: Proceedings of the National Academy of Sciences, pp. 12562–12566. DOI: 10.1073/pnas.202427399.

[229] Alessandro Barducci, Giovanni Bussi, and Michele Parrinello. “Well-Tempered Metadynamics: A Smoothly Converging and Tunable Free-Energy Method”. In: *Physical Review Letters* 100.2 (Jan. 18, 2008). Publisher: American Physical Society, p. 020603. DOI: 10.1103/PhysRevLett.100.020603.

[230] David R. Glowacki, Emanuele Paci, and Dmitrii V. Shalashilin. “Boxed molecular dynamics: A simple and general technique for accelerating rare event kinetics and mapping free energy in large molecular systems”. In: *Journal of Physical Chemistry B* 113.52 (Dec. 31, 2009). Publisher: American Chemical Society, pp. 16603–16611. ISSN: 15206106. DOI: 10.1021/jp9074898.

[231] Jonathan Booth et al. “Recent applications of boxed molecular dynamics: A simple multiscale technique for atomistic simulations”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 372.2021 (Aug. 6, 2014). Publisher: Royal Society. ISSN: 1364503X. DOI: 10.1098/rsta.2013.0384.

[232] Robin J. Shannon et al. “Adaptively Accelerating Reactive Molecular Dynamics Using Boxed Molecular Dynamics in Energy Space”. In: *Journal of Chemical Theory and Computation* 14.9 (Sept. 11, 2018). Publisher: American Chemical Society, pp. 4541–4552. ISSN: 15499626. DOI: 10.1021/acs.jctc.8b00515.

[233] Rafael A. Jara-Toro et al. “Enhancing Automated Reaction Discovery with Boxed Molecular Dynamics in Energy Space”. In: *ChemSystemsChem* 2.1 (Jan. 23, 2020). Publisher: arXiv, e1900024. ISSN: 2570-4206. DOI: 10.1002/syst.201900024.

[234] A. Amadei, A. B. Linssen, and H. J. Berendsen. “Essential dynamics of proteins”. In: *Proteins* 17.4 (Dec. 1993), pp. 412–425. ISSN: 0887-3585. DOI: 10.1002/prot.340170408.

[235] Stephanie R. Hare et al. “Low dimensional representations along intrinsic reaction coordinates and molecular dynamics trajectories using interatomic distance matrices”. In: *Chemical Science* 10.43 (2019). Publisher: Royal Society of Chemistry, pp. 9954–9968. DOI: 10.1039/C9SC02742D.

[236] L. Molgedey and H. G. Schuster. “Separation of a mixture of independent signals using time delayed correlations”. In: *Physical Review Letters* 72.23 (June 6, 1994). Publisher: American Physical Society, pp. 3634–3637. DOI: 10.1103/PhysRevLett.72.3634.

[237] Yusuke Naritomi and Sotaro Fuchigami. “Slow dynamics of a protein backbone in molecular dynamics simulation revealed by time-structure based independent component analysis”. In: *The Journal of Chemical Physics* 139.21 (Dec. 7, 2013). Publisher: American Institute of Physics, p. 215102. ISSN: 0021-9606. DOI: 10.1063/1.4834695.

[238] Steffen Schultze and Helmut Grubmüller. "Time-Lagged Independent Component Analysis of Random Walks and Protein Dynamics". In: *Journal of Chemical Theory and Computation* 17.9 (Sept. 14, 2021). Publisher: American Chemical Society, pp. 5766–5776. ISSN: 1549-9618. DOI: 10.1021/acs.jctc.1c00273.

[239] Mohammad M. Sultan and Vijay S. Pande. "tICA-Metadynamics: Accelerating Metadynamics by Using Kinetically Selected Collective Variables". In: *Journal of Chemical Theory and Computation* 13.6 (June 13, 2017). Publisher: American Chemical Society, pp. 2440–2447. ISSN: 1549-9618. DOI: 10.1021/acs.jctc.7b00182.

[240] Helen Deeks et al. *Virtual reality sampled pathways guide free energy calculation of protein-ligand binding.* Apr. 19, 2022. DOI: 10.26434/chemrxiv-2022-w89tc.

[241] S. Izrailev et al. "Molecular dynamics study of unbinding of the avidin-biotin complex". In: *Biophysical Journal* 72.4 (1997). Publisher: Biophysical Society, pp. 1568–1581. ISSN: 00063495. DOI: 10.1016/S0006-3495(97)78804-0.

[242] Rainer A. Böckmann and Helmut Grubmüller. "Nanoseconds molecular dynamics simulation of primary mechanical energy transfer steps in F1-ATP synthase". In: *Nature Structural Biology* 9.3 (Mar. 2002). Bandiera_abtest: a Cg_type: Nature Research Journals Number: 3 Primary_atype: Research Publisher: Nature Publishing Group, pp. 198–202. ISSN: 1545-9985. DOI: 10.1038/nsb760.

[243] Aleksij Aksimentiev et al. "Insights into the Molecular Mechanism of Rotation in the Fo Sector of ATP Synthase". In: *Biophysical Journal* 86.3 (Mar. 2004), pp. 1332–1344. ISSN: 0006-3495.

[244] Jan Saam et al. "Molecular Dynamics Investigation of Primary Photoinduced Eventsin the Activation of Rhodopsin". In: *Biophysical Journal* 83.6 (Dec. 1, 2002). Publisher: Elsevier, pp. 3097–3112. ISSN: 0006-3495. DOI: 10.1016/S0006-3495(02)75314-9.

[245] Carsten Kutzner, Jacek Czub, and Helmut Grubmüller. "Keep It Flexible: Driving Macromolecular Rotary Motions in Atomistic Simulations with GROMACS". In: *Journal of Chemical Theory and Computation* 7.5 (May 10, 2011). Publisher: American Chemical Society, pp. 1381–1393. DOI: 10.1021/CT100666V.

[246] Andreas Dullweber, Benedict Leimkuhler, and Robert McLachlan. "Symplectic splitting methods for rigid body molecular dynamics". In: *The Journal of Chemical Physics* 107.15 (Oct. 15, 1997). Publisher: American Institute of Physics, pp. 5840–5851. ISSN: 0021-9606. DOI: 10.1063/1.474310.

[247] T. F. Miller et al. "Symplectic quaternion scheme for biophysical molecular dynamics". In: *The Journal of Chemical Physics* 116.20 (May 22, 2002). Publisher: American Institute of PhysicsAIP, pp. 8649–8659. ISSN: 0021-9606. DOI: 10.1063/1.1473654.

[248] Ana J. Silveira and Charlles R. A. Abreu. “Molecular dynamics with rigid bodies: Alternative formulation and assessment of its limitations when employed to simulate liquid water”. In: *The Journal of Chemical Physics* 147.12 (Sept. 28, 2017). Publisher: American Institute of Physics, p. 124104. ISSN: 0021-9606. DOI: 10.1063/1.5003636.

[249] Oliver Johns. “Kinematics of Rotation”. In: *Analytical Mechanics for Relativity and Quantum Mechanics*. 2011, pp. 152–199. ISBN: 978-0-19-100162-8.

[250] E. Krissinel and K. Henrick. “Secondary-structure matching (SSM), a new tool for fast protein structure alignment in three dimensions”. In: *Acta Crystallographica. Section D, Biological Crystallography* 60 (Pt 12 Pt 1 Dec. 2004), pp. 2256–2268. ISSN: 0907-4449. DOI: 10.1107/S0907444904026460.

[251] W. Kabsch. “A solution for the best rotation to relate two sets of vectors”. In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 32.5 (Sept. 1, 1976). Number: 5 Publisher: International Union of Crystallography, pp. 922–923. ISSN: 0567-7394. DOI: 10.1107/S0567739476001873.

[252] W. Kabsch. “A discussion of the solution for the best rotation to relate two sets of vectors”. In: *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography* 34.5 (Sept. 1, 1978). Number: 5 Publisher: International Union of Crystallography, pp. 827–828. ISSN: 0567-7394. DOI: 10.1107/S0567739478001680.

[253] Konstantin N. Kudin and Anatoly Y. Dymarsky. “Eckart axis conditions and the minimization of the root-mean-square deviation: Two closely related problems”. In: *The Journal of Chemical Physics* 122.22 (June 8, 2005). Publisher: American Institute of Physics, p. 224105. ISSN: 0021-9606. DOI: 10.1063/1.1929739.

[254] Gerald R. Kneller. “Eckart axis conditions, Gauss’ principle of least constraint, and the optimal superposition of molecular structures”. In: *The Journal of Chemical Physics* 128.19 (May 21, 2008). Publisher: American Institute of Physics, p. 194101. ISSN: 0021-9606. DOI: 10.1063/1.2902290.

[255] Guillaume Chevrot et al. “Least constraint approach to the extraction of internal motions from molecular dynamics trajectories of flexible macromolecules”. In: *The Journal of Chemical Physics* 135.8 (Aug. 28, 2011). Publisher: American Institute of Physics, p. 084110. ISSN: 0021-9606. DOI: 10.1063/1.3626275.

[256] Douglas R. Gregory. “Rotating reference frames”. In: *Classical Mechanics*. Cambridge University Press, Apr. 13, 2006, pp. 469–491. ISBN: 978-0-521-82678-5.

[257] Petr Šulc et al. “Sequence-dependent thermodynamics of a coarse-grained DNA model”. In: *The Journal of Chemical Physics* 137.13 (Oct. 7, 2012). Publisher: American Institute of PhysicsAIP, p. 135101. ISSN: 0021-9606. DOI: 10.1063/1.4754132.

[258] A. Liwo et al. "A united-residue force field for off-lattice protein-structure simulations. I. Functional forms and parameters of long-range side-chain interaction potentials from protein crystal data". In: *Journal of Computational Chemistry* 18.7 (Dec. 7, 1997), pp. 849–873. ISSN: 1096-987X. DOI: `10.1002/(SICI)1096-987X(199705)18:7<849::AID-JCC1>3.0.CO;2-R`.

[259] Roberto Berardi, Carlo Fava, and Claudio Zannoni. "A Gay–Berne potential for dissimilar biaxial particles". In: *Chemical Physics Letters* 297.1 (Nov. 20, 1998), pp. 8–14. ISSN: 0009-2614. DOI: `10.1016/S0009-2614(98)01090-2`.

[260] R. Everaers and M. R. Ejtehadi. "Interaction potentials for soft and hard ellipsoids". In: *Physical Review E* 67.4 (Apr. 21, 2003). Publisher: American Physical Society, p. 041710. DOI: `10.1103/PhysRevE.67.041710`.

[261] Norbert Kern and Daan Frenkel. "Fluid–fluid coexistence in colloidal systems with short-ranged strongly directional attraction". In: *The Journal of Chemical Physics* 118.21 (June 2003). Publisher: American Institute of Physics, pp. 9882–9889. ISSN: 0021-9606. DOI: `10.1063/1.1569473`.

[262] W. Michael Brown et al. "Liquid crystal nanodroplets in solution". In: *The Journal of Chemical Physics* 130.4 (Jan. 28, 2009). Publisher: American Institute of Physics, p. 044901. ISSN: 0021-9606. DOI: `10.1063/1.3058435`.

[263] Shengfeng Cheng and Mark J. Stevens. "Self-assembly of chiral tubules". In: *Soft Matter* 10.3 (Dec. 17, 2013). Publisher: The Royal Society of Chemistry, pp. 510–518. ISSN: 1744-6848. DOI: `10.1039/C3SM52631C`.

[264] Trung Dac Nguyen and Steven J. Plimpton. "Aspherical particle models for molecular dynamics simulation". In: *Computer Physics Communications* 243 (Oct. 1, 2019), pp. 12–24. ISSN: 0010-4655. DOI: `10.1016/j.cpc.2019.05.010`.

[265] Oliver Henrich et al. "Coarse-grained simulation of DNA using LAMMPS". In: *The European Physical Journal E* 41.5 (May 10, 2018), p. 57. ISSN: 1292-895X. DOI: `10.1140/epje/i2018-11669-8`.

[266] Adam Liwo et al. "Theory and Practice of Coarse-Grained Molecular Dynamics of Biologically Important Systems". In: *Biomolecules* 11.9 (Sept. 2021). Number: 9 Publisher: Multidisciplinary Digital Publishing Institute, p. 1347. ISSN: 2218-273X. DOI: `10.3390/biom11091347`.

[267] Denis J. Evans. "On the representatation of orientation space". In: *Molecular Physics* 34.2 (Aug. 1, 1977). Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00268977700101751 pp. 317–325. ISSN: 0026-8976. DOI: `10.1080/00268977700101751`.

[268] D.C Rapaport. “Molecular dynamics simulation using quaternions”. In: *Journal of Computational Physics* 60.2 (1985), pp. 306–314. ISSN: 00219991. DOI: 10.1016/0021-9991(85)90009-9.

[269] R. L. Davidchack, T. E. Ouldridge, and M. V. Tretyakov. “New Langevin and gradient thermostats for rigid body dynamics”. In: *The Journal of Chemical Physics* 142.14 (Apr. 14, 2015). Publisher: AIP Publishing LLCAIP Publishing, p. 144114. ISSN: 0021-9606. DOI: 10.1063/1.4916312.

[270] André Lanrezac, Nicolas Férey, and Marc Baaden. “Wielding the power of interactive molecular simulations”. In: *WIREs Computational Molecular Science* n/a (n/a 2021). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wcms.1594, e1594. ISSN: 1759-0884. DOI: 10.1002/wcms.1594.

[271] Seth Cooper et al. “Predicting protein structures with a multiplayer online game”. In: *Nature* 466.7307 (Aug. 2010). Number: 7307 Publisher: Nature Publishing Group, pp. 756–760. ISSN: 1476-4687. DOI: 10.1038/nature09304.

[272] Guillaume Levieux et al. “Udock, the interactive docking entertainment system”. In: *Faraday Discussions* 169 (2014), pp. 425–441. ISSN: 1359-6640. DOI: 10.1039/c3fd00147d.

[273] Richard H. Bartels, John C. Beatty, and Brian A. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers, Inc., 1987. ISBN: 0-934613-27-3.