Interactive Molecular Docking with Haptics and Advanced Graphics

Nicholas Matthews

School of Computing Sciences

University of East Anglia

This dissertation is submitted for the degree of Doctor of Philosophy

School of Computing Sciences October 2019

Abstract

Biomolecular interactions underpin many of the processes that make up life. Molecular docking is the study of these interactions *in silico*. Interactive docking applications put the user in control of the docking process, allowing them to use their knowledge and intuition to determine how molecules bind together.

Interactive molecular docking applications often use haptic devices as a method of controlling the docking process. These devices allow the user to easily manipulate the structures in 3D space, whilst feeling the forces that occur in response to their manipulations. As a result of the force refresh rate requirements of haptic devices, haptic assisted docking applications are often limited, in that they model the interacting proteins as rigid, use low fidelity visualisations or require expensive propriety equipment to use.

The research in this thesis aims to address some of these limitations. Firstly, the development of a visualisation algorithm capable of rendering a depiction of a deforming protein at an interactive refresh rate, with per-pixel shadows and ambient occlusion, is discussed. Then, a novel approach to modelling molecular flexibility whilst maintaining a stable haptic refresh rate is developed.

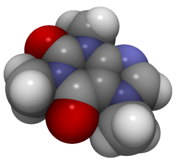
Together these algorithms are presented within Haptimol FlexiDock, the first haptic-assisted molecular docking application to support receptor flexibility with high fidelity graphics, whilst also maintaining interactive refresh rates on both the haptic device and visual display.

Using Haptimol FlexiDock, docking experiments were performed between two protein-ligand pairs: Maltodextrin Binding Protein and Maltose, and Glutamine Bind-

ing Protein and Glucose. When the ligand was placed in its approximate binding site, the direction of over 80% of the intra-molecular movement aligned with that seen in the experimental structures. Furthermore, over 50% of the expected backbone motion was present in the structures generated with FlexiDock. Calculating the deformation of a biomolecule in real time, whilst maintaining an interactive refresh rate on the haptic device (> 500Hz) is a breakthrough in the field of interactive molecular docking, as, previous approaches either model protein flexibility, but fail to achieve the required haptic refresh rate, or do not consider biomolecular flexibility at all.

It's the job that's never started

as takes longest to finish - J.R.R Tolkien



Acknowledgements

I would like to acknowledge and thank my supervisory team, Dr Stephen Laycock and Dr Steven Hayward, for their good advice, guidance and endless patience over the course of the development of this thesis. I have learnt an incredible amount about how to properly conduct and document research, in addition to the subject knowledge required for the success of this thesis. Thank you! Thank you also to my examiners, Andy and Kurt, for their insightful comments regarding this thesis.

In addition, I would like to thank Akio Kitao for providing the molecular trajectories required to demonstrate the effectiveness of Haptimol Protein Trajectory Viewer and Haptimol FlexiDock. Without them, it would have been impossible to develop the ideas presented in this thesis.

I would also like to thank the other members of the Machine Learning and Statistics (formally known as the Graphics, Colour and Visualisation) laboratory, for providing constant motivation and amusement over the course of the last four years. My thanks also to Russ and Matt in Computing support, for the assistance they have provided throughout this project.

Finally, I would like to thank some important people outside of the University. Firstly, thanks go out to my family, most specifically to my Mother, Judith, for listening (whatever the time of day) and providing support, motivation and encouragement. Thank you also to my friends, old and new, with special mention of Catherine, Tomos, Anna, Bethan, James, Jess, Zoe and Maddie: Without you, I would have lost the plot a long time ago! Thank you!

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Proteins can be considered the building blocks of life. They are complex molecules which perform a critical role in all of the body's functions and they are involved in all of the biochemical reactions taking place in its cells. They are required for structure, function and regulation of every organ, tissue and cell within the body.

Proteins rarely act alone. Molecular processes are often carried out by molecular machines that are built from a large number of protein components which are organized by how they interact with other proteins. Protein interactions also form a key part of the inter-cellular communication[1] process within the body. Receptor (generally the larger biomolecule of an interacting pair) proteins have binding sites which interact with a specific ligand, which can trigger a change of conformation in the receptor protein. This change in conformation can then trigger further changes within a cell.

Since the 1920s[50], microbiologists have studied these interactions in order to gain understanding of the processes involved, and use the information to find novel drugs and determine the causes of diseases.

A recent study by Pradeepkiran and Reddy[115] highlights the benefits of understanding biomolecular interactions. The authors use structure based drug design and molecular docking, both of which are highly dependent on the understanding of biomolecular interactions, to identify five possible ligands that have the potential to help in the battle against Alzheimer's disease; a disease which will effect 1 in 3 born today, and which currently has no cure. Furthermore, understanding molecular interactions has assisted in the discovery of new compounds that provide favourable protection against HIV[68], treatment for prostate cancer[48] and reduce pain[87].

With the rise of drug-resistant bacteria in recent years, the search for new viable drugs is becoming more important. Computing can be used to accelerate the process of discovering new drug candidates by quickly discounting non-viable molecules whilst highlighting likely candidates. Molecular docking is one of the tools used to achieve this.

1.1 Motivations and Research Objectives

Molecular docking is the process of computationally simulating the binding of two molecular structures into a single stable complex. It is a tool commonly used within structure-based drug design, as it can demonstrate how two molecules, the receptor and ligand, fit together (Illustrated in Figure 1.1). This is useful, because many drugs work by fitting into a "target" protein and in doing so, inhibit another molecule from binding, or trigger a separate series of events.

One can classify molecular docking into two types: automated and interactive. For automated docking, pose selection algorithms are employed to search for possible binding poses[6,100,107,158]. This can result in a large number of conformations, which are then scored with a scoring function. Most scoring functions use a physics based force field to estimate the energy of the calculated pose. A low amount of energy indicates a more stable system, and therefore a potential binding interaction. Interactive systems put the user in charge of the docking process, allowing them to use their knowledge and intuition to find a docked pose[52,58].

One way of enhancing interactive docking is to use a haptic device. Haptic-assisted interactive docking systems make use of haptic feedback devices to aid the docking

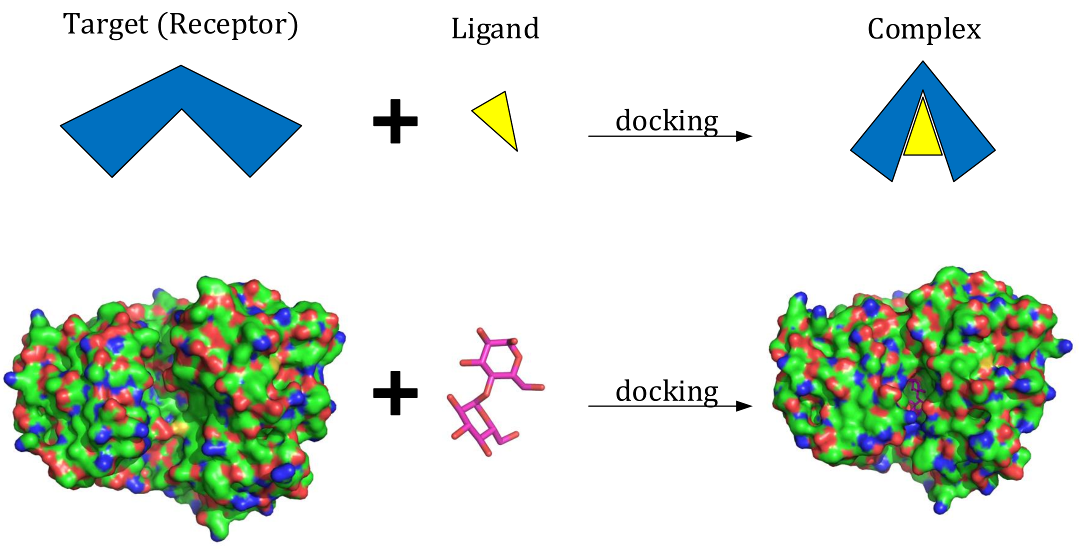


Fig. 1.1 Molecular docking is the practice using computational methods to determine if, and how, two biomolecules bind together. Top row: receptor (Blue) and ligand (yellow) bind together to form a stable complex. In this example, the receptor deforms to better fit the shape of the ligand. Modelling this conformation change within molecular docking remains challenging. The bottom row shows the same process, but with real biomolecules (Maltodextrin binding protein and maltose).

process. These systems allow the operator to use their sense of touch to guide the ligand into a docked pose. The interaction forces that occur during the docking session are transmitted to the user via the haptic device. Besides offering an environment to rapidly test new ideas and hypotheses, haptic-assisted interactive docking systems can be used in conjunction with automated systems to allow experts to test high scoring poses and either improve them or reject them[91,118]. Interactive docking systems have been shown to improve the users' understanding of the process of molecular binding[16,111].

One of the difficulties with using a haptic device with a docking application is the refresh rate demanded by the haptic device: for continuous, smooth and stable kinaesthetic and tactile responses, haptic technology requires the haptic feedback cues to be updated at a refresh rate greater than 500 Hz, ideally at 1 kHz, due to the sensitivity of the human haptic system[33,96]. When this rate is not met device vibrations and force discontinuities can occur limiting practical use.

As a result of this, the majority of current interactive docking systems simplify the simulation by not modelling the conformational change that can occur to a protein upon ligand binding, and is required in order to accurately simulate many molecular interactions.

In a haptic-assisted interactive docking application, the haptic device typically works in tandem with the visual display. The visual display contains a depiction of the ongoing docking scenario, informing the user of the relative position of the ligand and receptor, and their respective topographies. The user is reliant on the visual representation in order to identify possible binding sites, which are often indicated by surface features like pockets and crevices, before steering the ligand toward them with the haptic device.

When the interacting biomolecules are modelled as rigid, it is possible to understand the topography of each of the structures by viewing them from multiple angles during the docking session. If the molecules are continually deforming in response to their current poses, viewing them from different angles becomes difficult. Therefore, a clear visualisation that allows the user to understand the relative depth of parts of the topography of the structures is important.

Current haptic-assisted interactive docking applications prioritise the haptic rendering side of the application, as a result of the strict refresh rate requirements imposed by the haptic device[55], because of this, the visual rendering is often basic, and lacking lighting techniques which can enhance the perception of depth within the scene.

1.2 Research Aims

The objective of this thesis is to address some of the limitations of current haptic-assisted interactive docking applications, by answering the following research questions:

- Is it possible to render, in real-time, on consumer grade hardware, a deforming biomolecular structure with per-pixel lighting effects?
- Is it possible to calculate the deformation of a biomolecule within the time frame imposed by a haptic device, using consumer grade hardware?

The force calculation algorithm developed by Iakovou et al.[56] utilises a regular grid in order to support interactive docking of large biomolecules in haptic time. In order for it to continue to work, algorithms for reconstructing regular grids in real time need to be investigated. Therefore, the additional research question “Is it possible to rebuild or update a regular grid within the 2 ms time frame imposed by the haptic device?” needs to be answered.

1.3 Contributions

The primary research contributions of this thesis are incorporated within the applications Haptimol Protein Trajectory Viewer[89] and Haptimol FlexiDock[90]. Haptimol Protein Trajectory Viewer combines the result of the comparative review of graphics processor (GPU) based grid construction algorithms (Chapter 3) with per-pixel lighting effects that rely heavily on a spatial partitioning structure, in order to render large protein trajectories with high fidelity graphics at an interactive refresh rate, on a desktop PC, for the first time (Chapter 4). ProteinViewer is shown to achieve real-time frame rates whilst rendering the trajectory of a large flagella biomolecule, comprising 314k atoms. The methods used to generate the lighting effects are designed to be compatible with the molecular docking application, Haptimol FlexiDock.

Haptimol FlexiDock is the first haptic-assisted interactive molecular docking application that supports receptor flexibility whilst maintaining a viable haptic-refresh rate. This is achieved by using a novel GPU-based algorithm to compute the receptor’s deformation as a response to the current interaction forces.

The effectiveness of Haptimol FlexiDock is demonstrated by simulating a docking scenario involving the maltodextrin binding protein (MBP) and maltose, and the glutamine binding protein (GlnBP) and glutamine. Results show that the poses generated by Haptimol FlexiDock, when the ligands are placed close to their experimentally determined binding sites, are similar to the experimentally determined ligand-bound poses.

The development of Haptimol FlexiDock resulted in further contributions to the field, including use of an iterative approach to applying the calculated deformation and integration of a colour scheme used to increase understanding of the ongoing docking scenario.

1.4 Publications

The work presented in this thesis is published in the peer reviewed journal articles:

Matthews, N., Easdon, R., Kitao, A., Hayward, S., & Laycock, S. (2017). High quality rendering of protein dynamics in space filling mode. Journal of Molecular Graphics and Modelling, 78, 158–167.

Matthews, N., Kitao, A., Laycock S., & Hayward, S. (2019) Haptic-assisted interactive molecular docking incorporating receptor flexibility. Journal of Chemical Information and Modelling, DOI: 10.1021/acs.jcim.9b00112

1.5 Thesis Outline

The research presented in this thesis comprises three distinct topics: GPU based grid construction, molecular trajectory rendering and haptic-assisted molecular docking with receptor flexibility. As a result, there is a large amount of existing literature that is relevant to various parts of this thesis. Therefore, each content chapter, Chapters 3 - 5, begin by reviewing literature that is relevant to the research presented within it.

Chapter 2 provides a general background of molecular docking, biomolecular interactions and working with haptics. The topic of GPU programming is also necessarily touched upon, due to the dependence of this project on GPU hardware.

Chapter 3 investigates rapid reconstruction of a spatial partitioning structure, a regular grid, on the GPU. Three algorithms are defined, implemented, optimised, tested and compared. The algorithms presented here are pivotal to the rendering system described in Chapter 4.

In Chapter 4, a novel method of rendering Molecular Dynamics (MD) trajectories with ray cast shadows and per pixel ambient occlusion is presented. The methods are implemented in the rendering software 'Haptimol Protein Trajectory Viewer.'

Chapter 5 describes a novel approach to modelling receptor flexibility during molecular docking. The methods discussed are included in the software Haptimol FlexiDock. The approach used is formally presented, tested and discussed, including verification that the rendered motions are close to those that are experimentally derived. At the end of the chapter, the rendering algorithm presented in Chapter 4 is included in FlexiDock.

Chapter 6 discusses the work presented in the thesis and highlights the next steps required in order to achieve the final goal of simulating fully flexible docking.

Chapter 2

Background

2.1 Molecular Docking

The term 'molecular docking' describes the methods used to determine how two structures bind together to form a stable complex *in silico*. The aim of docking is to determine the alignment, conformation and orientation of a molecule when it binds with another to form a complex. Figure 2.1 shows a typical docking scenario.

Complex formation occurs when the free energy, the energy available to do work such as driving a chemical reaction, is at a minimum, and the receptor and ligand conformations complement each other strongly, both geometrically and chemically. Using computational methods to determine these conformations is the aim of molecular docking[84].

Figure 2.1 shows a docking scenario made up of a ligand: maltose, and a receptor: maltodextrin binding protein (MBP). The receptor is usually a protein or other large biomolecule, and the ligand is usually a smaller structure, for example a drug comprising few atoms, although larger molecules could be used[101].

During the docking process, the receptor and ligand deform their structures into conformations that favour binding. The induced conformational change could be limited to minor side chain rotations, or could include major backbone movements,

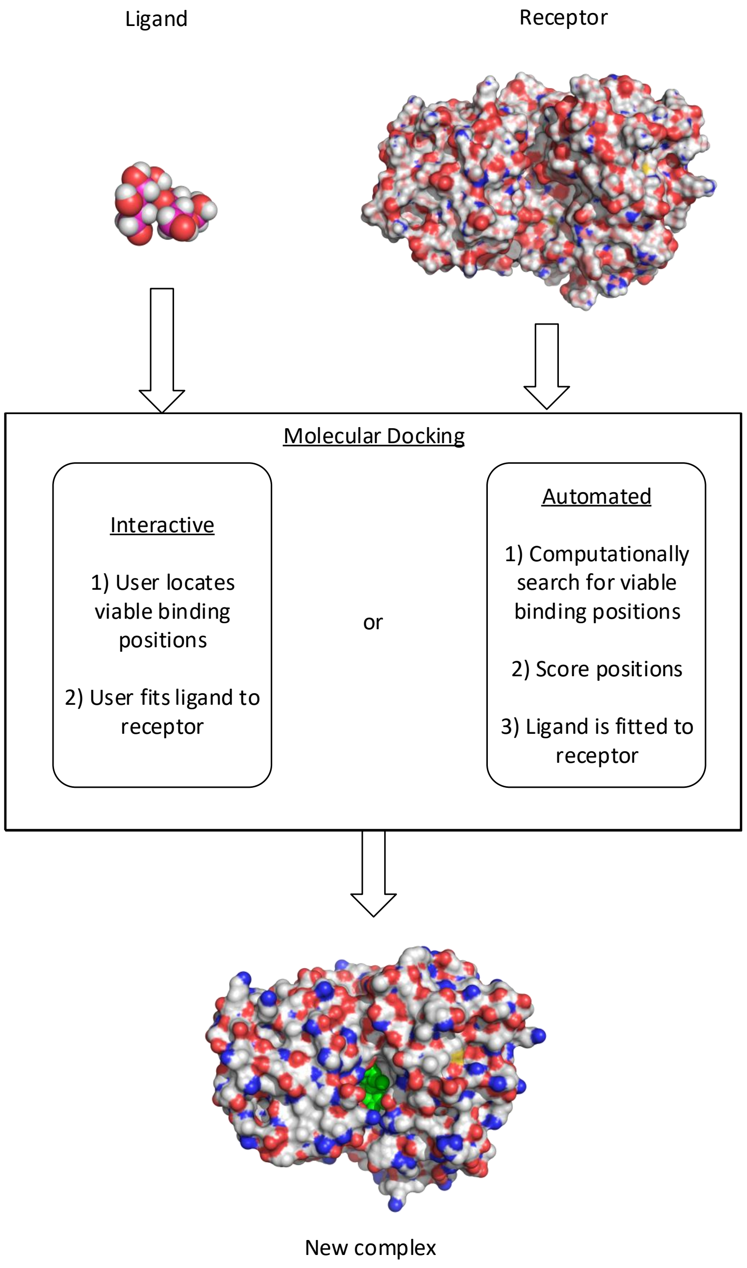

Ligand
Receptor
Molecular Docking
Interactive
1) User locates viable binding positions
2) User fits ligand to receptor
or
Automated
1) Computationally search for viable binding positions
2) Score positions
3) Ligand is fitted to receptor
New complex

Fig. 2.1 Molecular docking. Using computational methods, the binding conformation between a receptor and ligand is found. Within interactive docking the user guides the docking process. In automated docking, computational methods are used to find, and then score, potential binding poses.

depending on the proteins being docked. In the docking scenario depicted in Figure 2.1, for example, the receptor undergoes a domain motion, transforming from an "open" to a "closed" conformation around the ligand.

Identifying how a receptor and ligand bind together can be a key step in developing new methods for treating illnesses. If, for example, an infection was reliant on a foreign molecule binding to a specific biomolecule within the body, researchers could use molecular docking to identify how the pathogen binds to the host's biomolecule, and then scan a large library of ligands in order to identify one that has the potential to disrupt the infections processes'[101].

One can classify molecular docking into two categories: automated and interactive. In automated docking, pose selection algorithms are employed to search for possible binding poses[6,107,158]. Both virtual screening (VS), a tool used to rapidly scan a large library of potential ligands to find those that bind to the target protein[148], and also computer aided drug design, where scientists test how well their synthetic molecules will bind to the target receptor[104], utilise automated molecular docking. Ruling out a large number of potentially viable ligand candidates *in silico*, rather than experimentally, reduces costs and improves the efficiency of novel drug design, making it attractive to pharmaceutical companies. Automated docking has also been used during the process of bioremediation to find enzymes that have the potential of breaking down target pollutants[138].

Automated docking has limitations, primarily that current approaches are not very accurate[82], and the fact that the docking process can take a considerable amount of time even when using a computer cluster[82].

Interactive docking is different in that it places the user in charge of the docking process, allowing them to judge when a biomolecule is docked, rather than solely using a scoring algorithm. Although the overall aim of docking is the same for both interactive and automated docking approaches, they should not be viewed as in competition. Interactive approaches are designed to allow the user to focus the search, and potentially improve the docking pose based on their knowledge and expertise[106,137], rather than testing a large number of potential ligands for docking affinity. Furthermore, in the recent community-wide benchmarks for protein-protein docking[85], the methods which allowed for human intervention/refinement proved the most successful when performing docking on a difficult target.

Therefore, automated and interactive docking approaches should be used in tandem, with interactive docking potentially being used to refine the results of an automated docking session, potentially eliminating some of the false positives *in silico*, rather than in the lab. A common way of enhancing interactive docking is to use a haptic device to both position the ligand relative to the receptor, and also to convey information about the current docking pose to the user, through their sense of touch.

2.2 Working with Haptics

Haptic devices interface with the user through their sense of touch by generating forces and vibrations that they then feel. Haptic devices have been used to enhance interactive docking. In interactive docking, the haptic device is typically "connected" to the ligand, allowing the user to easily position the ligand in 3D space whilst simultaneously providing feedback, through their sense of touch, of the forces occurring between the receptor and ligand.

Haptic devices were first investigated for use in project GROPE[20], which started in 1965. Initial results were good, with the haptic device demonstrably enhancing perception and improving the performance of simulated simple motor tasks. The researchers found however, that the computational power available in the day was not sufficient to calculate the forces present in all but the most simple simulations.

In 1986 the computational power available was found to be sufficient to conduct more in-depth experiments, including the first complex molecular docking task. The molecular docking experiment carried out by Brooks Jr et al.[20] was designed to test whether or not haptic feedback reduced the amount of time it took to successfully dock a ligand. The results showed that providing haptic feedback improved docking performance, when compared to simply using the haptic device as a 3D mouse, although the average docking time was not substantially improved. However, after removing thinking time from the overall docking time, the docking times with haptic assistance were found to be 1.75 times quicker.

Brooks Jr et al.[20] note that the time saving is not the only driving factor of using haptics. Haptic devices result in improved situation awareness. Chemists using GROPE reported getting better comprehension of the force fields in the active sites of the molecules, and gained understanding of exactly why each candidate drug docked well or poorly[20]. Brooks Jr et al.[20] then hypothesised that with the improved understanding of the docking process, researchers will be able to form better ideas for new candidate drugs.

Since project GROPE, a number of haptic assisted interactive docking applications have been published by the research community (See Chapter 5, Section 5.2 for an in depth review of these applications), however, these approaches are limited because molecular flexibility has yet to be adequately addressed.

The primary reason for this is related to the refresh rate demanded by the haptic device. In order in insure that smooth kinaesthetic feedback is felt by the user, the haptic feedback device needs to be "refreshed" at a minimum rate of 500 Hz, ideally 1 kHz[33,96]. This refresh rate places a time limit of 2 ms on any computation required between haptic frames; a very small amount of time in which to compute intermolecular forces, calculate any conformation changes and update relevant models.

A state of the art interactive molecular docking application is HaptimolRD[57]. HaptimolRD utilises the GPU in order to calculate the interaction forces between very large molecules during the docking session. The pre-computation performed in HaptimolRD is limited to the creation of a spatial partitioning structure, either a regular grid or an octree, containing the ligand. This structure is then used to eliminate atom pairs that are far enough apart that they contribute a negligible amount to the total interaction force, reducing the computational cost of computing the net force, when the interacting molecules are large. In order for HaptimolRD to support ligand flexibility whilst maintaining its support for large structures, the spatial partitioning structure used during the force calculation must be rebuilt in minimal time.

Despite the potential benefits of using an interactive docking system, including increased situational awareness of the ongoing docking scenario and the ability for a user to use their own intuition during docking, uptake has been poor. The reason for is likely related to the limitations of interactive systems. Currently, the majority of interactive docking systems do not model flexibility, and those that do are limited in other ways (See Section 5.2), limiting how effective they are at modelling molecular interactions.

Although the haptic device is a key component of the interactive docking system, the visual render of the ongoing interaction is equally important. Current interactive docking systems limit themselves to basic rendering, with little or no lighting effects. This is unfortunate, because directional lighting has been shown to improve the depth perception of three dimensional scenes[150,151]. Figure 2.2 illustrates how directional shadowing can help highlight pockets and crevices within proteins. In Figure 2.2 (A), the protein looks almost flat, whilst in Figure 2.2 (B) deeper parts of the proteins topography are clearly revealed.

It is important that the user is aware of the locations of pockets when performing molecular docking, as they are often sites into which a ligand binds. Presently, HaptimolRD uses a basic rendering algorithm because of the computational expense of adding advanced lighting[55]. When performing rigid docking, this is acceptable, as it is possible to view the structure from multiple angles in order to understand its topography. When the structures can deform however, this becomes more difficult, as parts of the protein will move relative to one another, often in a short space of time. Therefore, to improve the ergonomics of the application, a rendering approach that improves the depth perception of the image is important (See Chapter 4).

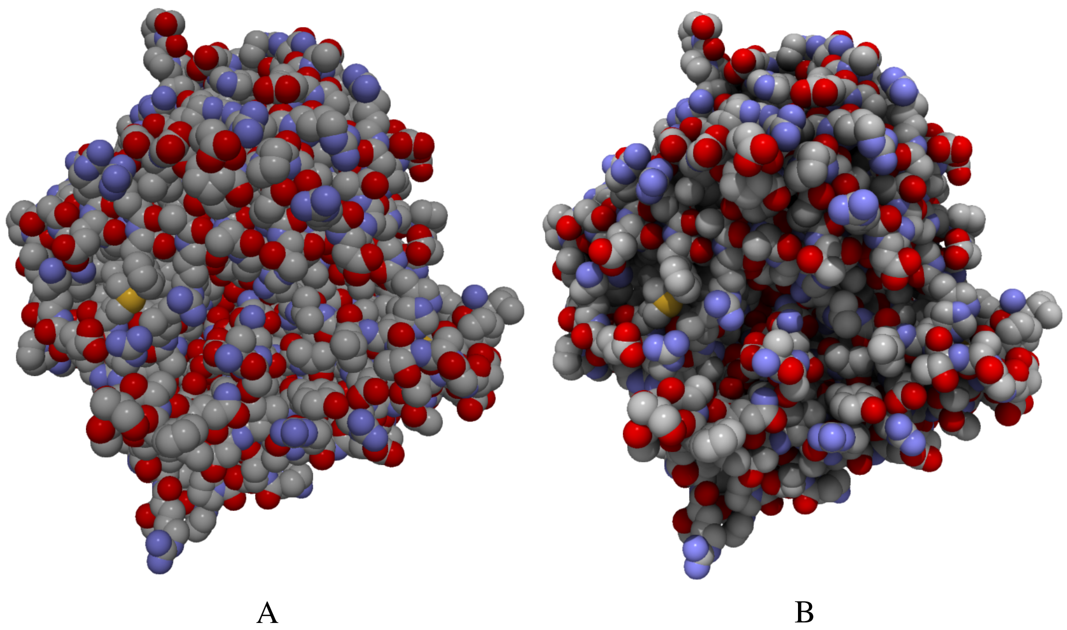


Fig. 2.2 Rendering of a protein, with A) directional lighting only and B) directional lighting, shadows and ambient occlusion. Note how the lighting effects enhance the topography in B), when compared with A).

2.3 Molecular Interactions

Molecular docking aims to determine whether two biomolecules, a receptor and ligand, will successfully bind together when in close proximity to one another. This is determined by calculating the interaction energy and the forces between the two structures. A receptor and ligand are deemed to be bound when the free energy in the overall system is at a minimum.

There are two types of interactions: bonded and non-bonded. Bonded, or covalent, interactions relate to the bond stretching, bond angle and torsion angle potentials that occur when a protein changes conformation. These interactions do not need to be considered when the interacting molecules are modelled as rigid. Non-bonded, or non-covalent, interactions occur between atoms which are not linked by a covalent bond. There are four types of non-bonded interaction, which are listed in Table 2.1. Although individually each of the non-bonded interactions is weak in comparison to a covalent bond, combined they can provide sufficient force to bind the ligand to the receptor.

Electrostatic interactions	These interactions occur between charged atoms. Oppositely charged atoms attract one another, whilst atoms with the same charge repel one another.
Hydrogen bond	A hydrogen bond is a partially electrostatic attractive force between a hydrogen atom that is bound to an electronegative atom (the donor), and another adjacent atom that bears a lone pair of electrons (the acceptor)[8].
Hydrophobic interactions	Hydrophobic interactions are the non-covalent forces that occur where non-polar molecules tend to cluster in water in order to decrease the overall interfacial area between the hydrophobic area and water. They are important when modelling protein folding.
Van der Waals (vdW)	Van der Waals forces' is a general term used to define the attraction of intermolecular forces between molecules. There are two kinds of Van der Waals forces: weak London Dispersion Forces and short-range repulsive forces which prevent the electron clouds of non-bonded atoms from overlapping.

Table 2.1 Non-bonded interactions that occur between biomolecules

In an interactive docking environment, the interaction energy and forces need to be determined in real-time. Most current molecular docking applications simplify the docking problem by modelling both biomolecules as rigid, removing the requirement for incorporating bonded interactions, and then exclusively model van der Waals (vdW) interactions[13,52,77], or vdW and Coulombic interactions[39,83,106,137,155]. The vdW interactions can be modelled using the Lennard-Jones (LJ) potential. The function

$$\vec{\boldsymbol{E}}_{VDW} = \sum_{i=1}^{N}\sum_{j=1}^{M} 4\varepsilon\Big[\Big(\frac{\sigma}{r}\Big)^{12} - \Big(\frac{\sigma}{r}\Big)^{6}\Big] \tag{2.1}$$

can be used to approximate the vdW energy potential between a ligand comprising M atoms, and a receptor comprising N atoms, whilst the function

$$\vec{\boldsymbol{F}}_{VDW} = \sum_{i=1}^{N}\sum_{j=1}^{M} \Big(\Big(24\varepsilon_{ij}\Big[\frac{2\sigma_{ij}^{12}}{r_{ij}^{13}} - \frac{\sigma_{ij}^{6}}{r_{ij}^{7}}\Big]\Big)\vec{\boldsymbol{r}}_{ij}\Big) \tag{2.2}$$

can be used to calculate the vdW interaction force between the same.

In Equations (2.1) and (2.2) ε_{ij} and σ_{ij} are LJ parameters that depend on the characteristics of the interacting atoms, r_{ij} is the distance between the two interacting atoms and $\vec{\hat{\boldsymbol{r}}}_{ij}$ is the unit vector in the direction of atom i to atom j.

To utilise Equations (2.1) and (2.2), the LJ parameters of each of the atoms involved in the docking scenario need to be known. Forcefields can be used to determine these values. The forcefields, including AMBER[31], GROMOS[105], and CHARMM[145], contain experimentally derived values for ε and σ for a large number of atoms. These forcefields are used by molecular dynamics (MD) applications when performing simulations[131], and also by molecular docking applications[58].

The vdW forces comprise an attractive and repulsive component. The first part of the equation, $\frac{2\sigma_{ij}^{12}}{r_{ij}^{13}}$, describes the repulsive component of the force. These forces occur as a result of the overlap of the interacting atom's electron clouds, and are only effective at a very short range. The second component of the equation, $\frac{\sigma_{ij}^{6}}{r_{ij}^{7}}$, equates to the long range attractive vdW forces. Figure 2.3 (B) shows how the forces change rapidly from an attractive force to a repulsive force, as the distance between the interacting atoms reduces to less than $\sigma\sqrt[6]{2}$. Figure 2.3 shows the vdW energy potential and force profile between two oxygen atoms. The force diagram, Figure 2.3 (B) shows how rapidly the LJ force moves from an attractive force to a highly repulsive force as the distance between the interacting atoms becomes very small.

When the ligand is more distant, the electrostatic effect produces a stronger response than the LJ interactions. These electrostatic interactions, as included in some docking applications, are often calculated with Coulomb's law, The electrostatic energy between two biomolecules can be calculated with the function

$$\vec{\boldsymbol{E}}_{ES} = \sum_{i=1}^{N} \sum_{j=1}^{M} \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon r_{ij}} \tag{2.3}$$

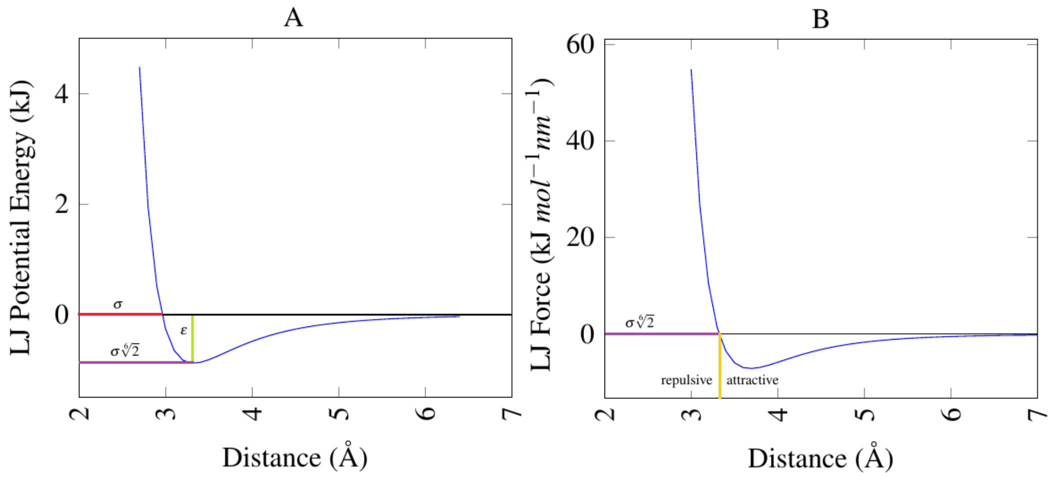


Fig. 2.3 LJ energy potential (A) and interaction force (B) between two oxygen atoms. $\sigma = 0.295992$ $\varepsilon = 0.87864$. Note how the force between the atoms quickly becomes strongly repulsive as the distance between the atoms becomes less than $\varepsilon\sqrt[6]{2}$. σ describes the distance at which the interaction energy becomes greater than 0, whilst ε describes the minimum energy.'

whilst the electrostatic interaction force is calculated with the similar function

$$\vec{\boldsymbol{F}}_{ES} = \sum_{i=1}^{N}\sum_{j=1}^{M} \frac{q_i q_j}{4\pi\varepsilon_0\varepsilon r_{ij}^2}\vec{\boldsymbol{r}}_{ij} \tag{2.4}$$

In these Equations, q_i and q_j are the atomic charges of the two atoms, extracted from a forcefield, ε_0 is the permittivity of free space and ε is the relative permittivity dependent on the dielectric properties of the solvent.

Some molecular docking applications use the function

$$\vec{\boldsymbol{E}}_{VDW+ES} = \vec{\boldsymbol{E}}_{VDW} + \vec{\boldsymbol{E}}_{ES} \tag{2.5}$$

to calculate the total electrostatic and vdW interaction energy between two biomolecules, and

$$\vec{\boldsymbol{F}}_{VDW+ES} = \vec{\boldsymbol{F}}_{VDW} + \vec{\boldsymbol{F}}_{ES} \tag{2.6}$$

to calculate the interaction forces. These equations do not consider the hydrophobic interactions, and although these factors are highly relevant to the strength of the binding, studies suggest that, when addressing rigid-body docking problems[106,137], Equations 2.5 and 2.6 provide a good approximation of the total energy and force involved. Furthermore, there have been no significant studies that model the hydrophobic interactions that occur within biomolecules. Therefore, as with other docking applications, they will not be included in the work within this thesis.

Within an interactive docking application, these forces need to be determined in real-time. A number of techniques have been used to achieve this, including using pre-computed force grids[13,15,19,77,83,136,155] and performing the calculations in real-time between all of the atom pairs[39,52,58,102]. This is discussed further in Chapter 5, Section 5.2. Iakovou et al.[56] presented an GPU based algorithm to calculate the vdW and Electrostatic interactions between two rigid molecules in haptic time. The algorithm presented by Iakovou et al.[56] will be used to calculate the interaction forces within the molecular docking approach used within this thesis.

2.3.1 Flexibility in Proteins

The majority of interactive docking applications model the proteins as rigid, in order to reduce the computational difficulty of the docking problem. However, proteins are flexible: in order to function, their conformation often has to change. It has been observed that a conformational change is often prompted when a ligand bonds to a receptor[41]. There are two main theories relating to how this conformational shift occurs: the "induced fit" and "selected fit" hypotheses. Figure 2.4 shows how these theories differ.

The induced fit hypothesis states that the ligand binding induces conformational change in the receptor, whilst selected fit states that the ligand selects and stabilises the receptor into a complementary fit, and then binds to it.

A biomolecule, restrained so it cannot translate or rotate globally, has 3N-6 degrees of freedom (DoF), where N is the number of atoms in the biomolecule. It is the large dimensionality of molecular flexibility that makes it the most challenging aspect

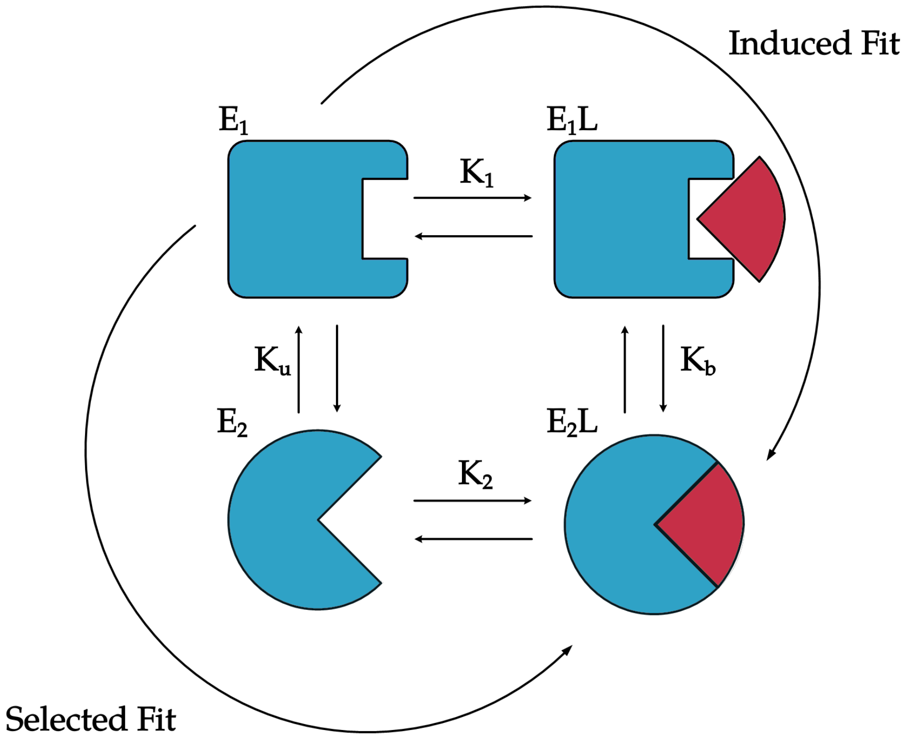


Fig. 2.4 Four state diagram showing the two main theories as to how conformation change occurs during binding. E_1 is the starting conformation of the protein, E_2 is the protein in its binding conformation, E_1L and E_2L are the same poses, but with a ligand bound. On the induced-fit route, the ligand binds to the receptor in its starting pose, then the receptor changes conformation to match. Within the selected-fit pose, the receptor deforms to a conformation that corresponds to the shape of the binding ligand, then the ligand binds. Based on diagram presented by Weikl and Deuster[152]

of modelling molecular interactions. Nevertheless, it is widely acknowledged that molecular flexibility improves the quality of poses found via molecular docking, and so, creating a tool that can model it in real time could be widely beneficial. In Chapter 5, a method to model receptor flexibility that works in haptic time is presented.

2.4 GPU Programming

The key to the goal of interactively modelling protein interactions with full flexibility at an interactive refresh rate is the GPU. GPUs provide a way to perform a large number of calculations in parallel, allowing a huge amount of computation to be performed in very little time. Iakovou et al.[57] demonstrated that by performing protein interaction calculations on the GPU instead of the CPU, they achieved a 90x speed-up in performance.

General-purpose computing on graphics processing units (GPGPU) became popular in 2001, when programmable shader support was implemented for graphics cards. These early applications were limited however, because the shader APIs were designed for graphical rendering, not general purpose computation. Consequently, in order to implement general algorithms on the GPU, they had to be transcoded for use with graphics primitives.

In 2006 Nvidia corporation launched CUDA, which allowed programmes to be written in C and then compiled for execution on the GPU. CUDA added general programming features, like the ability to perform scattered read and write memory operations to the GPU for the first time. This resulted in a more diverse range of algorithms being implemented.

The earliest implementations of CUDA were somewhat limited, notably lacking features that can be pivotal when performing multi-threaded computation; atomic functions being an example of this[103]. In later versions these features were added, and combined with advancements in graphics hardware, have made the GPU a very versatile and powerful computing device. In 2009, OpenCL, an open source alternative to CUDA was released. OpenCL is advantageous in that it can be executed on any mainstream manufacturers GPU and also on the majority of CPUs, however, some of the features incorporated in CUDA are unavailable in OpenCL at present.

Despite the increase in available features, writing an application that utilises a GPU requires a different approach to algorithm design, compared to programming for execution on a CPU. These differences stem from the hardware design: the GPU is generally a Single Instruction Multiple Data (SIMD) processor: best performance is achieved when all of the threads are performing the same instruction. The code to be executed on a GPU is contained within individual kernels. The kernel describes the code to be executed by a single thread. When a kernel is started a large number of threads are launched, which execute the code in parallel. The number of threads required is specified when the kernel is launched. The threads are divided up into work groups, Nvidia GPUs sub-divide the work groups into blocks of 32 threads, called warps. Communication between threads in work groups is possible, although it comes with a performance penalty. Communication between threads that are in different work groups is more challenging, and should be avoided. Best performance is achieved when all the threads are performing the same operation, therefore code branches should be avoided, because if even a single thread in a work group branches, all of the threads in that group "stall" until that one thread has finished its branch. This can be detrimental to application performance.

The memory model on GPUs is also different to that of the CPU. Firstly, GPU memory is currently separate from system memory (although there has been progress toward having a unified memory model), therefore, GPU memory *buffers* have to be allocated separately to system memory. Communication between *host memory* and *device memory* is achieved with API calls. The *device* is the GPU, the *host* is the CPU and accompanying system memory. Copying memory between the host and device is a time consuming operation, and therefore should also be avoided during time critical sections of code.

There are six different memory spaces on the GPU. Generally, the more space available in any specific memory bank, the slower it is. Figure 2.5 shows the memory hierarchy of the GPU.

- Registers: Registers are the fastest memory space on the GPU. Each thread has a limited number of registers which are typically used to store frequently accessed thread-private variables. Registers are typically thread private, although it is possible to pass their contents directly to another thread within the block. The fewer registers utilised by a thread, the more warps can execute concurrently.

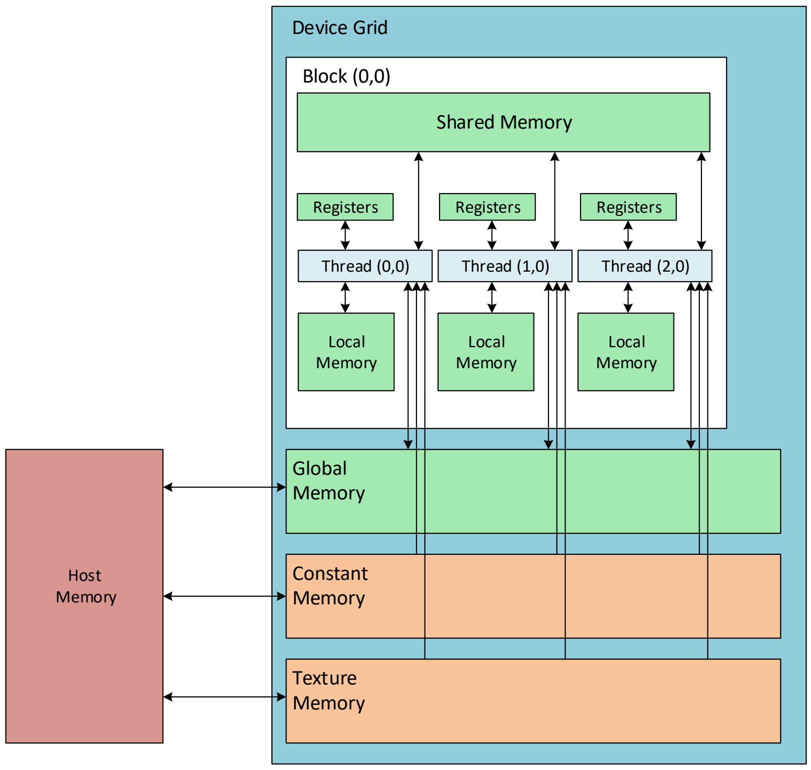


Fig. 2.5 Diagram showing the memory hierarchy of the GPU. Memory areas that threads can read and write to are coloured green, read only memory is orange and memory inaccessible from the GPU kernels is coloured red. Within the diagram, one thread block and three threads are shown. In reality, each block will have many threads, and the GPU grid will comprise many blocks. Figure adapted from Cheng et al.[23]

- Local Memory: Similarly to registers, local memory is thread private. However, local memory offers much lower performance than the registers, on a par-with global memory. It is primarily used for data that will not fit in the registers.
- Shared Memory: A limited amount of shared memory is available per work-block. All of the threads within a work group can can access the data within it. As shared memory is on chip, it offers low latency access, and a large amount of bandwidth. As shared memory is accessed by many threads, writes to it must be synchronised.
- Constant Memory: constant memory is a cache of dedicated device memory, which can offer a high amount of bandwidth. It cannot be modified by any

kernel, and must be declared by the host. Constant memory performs best when all threads in a warp read from the same memory address.

- Texture Memory: Texture memory is a type of cached read-only global device memory. Texture memory is optimised for 2D spatial locality, and supports floating-point interpolation as part of the read process. In some applications, it offers superior performance to global memory.

- Global Memory: Global memory is the largest and slowest memory bank on the GPU. It can be accessed from any thread at any time throughout the lifetime of the application. Coalescing memory accesses to global memory is important in order to achieve the best bandwidth.

The fastest way to load data from global memory is to coalesce memory accesses within a thread group. Memory coalescing is a technique used to achieve the best usage of global memory bandwidth. That is, when parallel threads running the same instruction access consecutive locations in global memory, the best memory access pattern is achieved.

In summary, to achieve the best performance possible with a GPU, the problem to be solved needs to be split between a large number of threads, that ideally utilise few registers. Global memory accesses should be coalesced, and any data used by multiple threads within a work group should be copied to shared memory where possible. Furthermore, any data required by the GPU kernels should be loaded into the GPU's memory before any time-critical code begins.

2.5 Summary

Molecular docking has proven useful to the pharmaceutical industry, contributing to the development of novel drugs. Automated docking approaches are useful for screening a large number of potential ligands for compatibility, but are limited in that they often produce many false positives. Interactive systems are obviously not suited to screening large libraries as automated approaches do, but rather should be used in conjunction with them, for example they could be used to eliminate false positives *in silico*. Augmenting an interactive docking application with haptics speeds up the docking process and helps the user comprehend why each candidate drug docked acceptably or less well. Since project GROPE, there have been a number of haptic-assisted interactive docking applications released, with a few made freely available for use.

Despite the potential benefits from using an interactive docking application, uptake has been poor, compared to automated approaches. This could be a result of comparative scarcity of haptic devices, required in order to achieve the best usage of the software, or the cost of proprietary docking software. However, one of the limitations of current interactive docking applications is that the majority of applications model interacting biomolecules as rigid, when in reality they are flexible structures. This could be a contributing factor for the poor uptake of interactive applications.

The next few chapters discuss the development of a haptic-assisted interactive docking application that supports molecular flexibility. The development is two fold, initially the visual representation of the biomolecules within the application is developed, then an approach to calculating molecular flexibility in haptic time is developed.

Chapter 3

GPU Grid Construction

3.1 Introduction

The main contributions of this thesis, high quality molecular rendering of deforming proteins and haptic-assisted interactive molecular docking with flexibility, require a spatial acceleration structure to achieve real-time performance. The acceleration structure, a regular grid, divides the Cartesian space of a 3D scene into cells which can be accessed directly. Each cell of the grid stores references to the objects that occupy that area of space; information that is accessible in near constant time. This information can then be used to reduce the number of queries performed by algorithms that are dependent on spatial information, which can result in reducing their runtime.

The main drawback of using a regular grid is that they can be costly to construct. This is often done as a precomputation step prior to use. However, pre-constructing the grid prevents it from being used in conjunction with a dynamic scene, because as the objects being rendered move, the contents of each cell change.

In order to use the regular grid in conjunction with a deforming molecular structure, the grid has to be updated, or rebuilt, every time the scene changes. When utilising the regular grid to accelerate the calculation of the intermolecular forces between molecules, whilst working with haptics, this could be as often as once every millisecond, with an upper time limit of 2 ms. Therefore, for the grid to be used for such a purpose, the grid construction must be fast enough to complete within this time frame, whilst leaving time for any other computation required.

Within this Chapter, previous attempts to rapidly construct regular grids are investigated. The fastest construction methods are then taken, optimised for molecular data, and then benchmarked. The benchmarks show that reconstructing the grid at a haptic-frame is possible, even for comparatively large proteins.

3.1.1 Contributions

This chapter's main contributions are:

- A review of current methods for constructing a regular grid, using a graphics processing unit.
- In depth analysis of three regular grid construction algorithms, including how long they take to construct and their performance in use.
- Testing in this chapter demonstrates, for the first time, that it is possible to reconstruct a regular grid in less than 2 ms, using consumer grade hardware.

3.2 Background

The regular grid was proposed as an acceleration structure in 1986[38], specifically for ray tracing. Since then, the structure has been used in other projects including: fluid dynamics[122], collision detection[144] and molecular docking[57]. A regular grid can be used to accelerate almost any N-body problem because it can be used to aggregate many intersection (or proximity) tests into a single operation. Other spatial data structures can be used to achieve the same thing. Tree-based structures are especially useful for reducing the number of tests performed during an N-body simulation, as multiple cells (or nodes) can be excluded with a single intersection test.

A Bounding Volume Hierarchy (BVH) is a tree-based data structure commonly used within ray-tracing applications, to reduce the number of ray-object intersection tests performed. In a BVH, all geometric objects are wrapped in bounding volumes, which form the leaf nodes of the tree. These nodes are then collected into sets and placed into larger bounding boxes. This continues in a recursive fashion, until the entire scene is in the same volume; this is the root node of the tree. During use, intersection tests are only performed on nodes whose parents were found to intersect with the ray, effectively reducing the time complexity of the algorithm from O(N) to O(log(N)).

In comparison to tree-based spatial partitioning structures, the regular grid has two distinct advantages. Firstly, accessing each cell of the grid is an O(1) operation. Secondly, regular grids are also more straightforward to construct, requiring only basic spatial decomposition, which results in fast build times[65]. These advantages come at the price of a larger memory footprint: Because grid structures divide the scene into equally sized cells, empty space is split up at the same resolution, leading to empty cells. Furthermore, the uniform cell size affects the traversal speed of the grid, because rather than just crossing empty regions in one step, as you can with a tree, many steps are required[60].

Regular grids are ideally suited for use with static scenes because they are straightforward to construct but difficult to update efficiently. Regular grid construction and reorganisation has been widely researched, however only a small number of discrete construction algorithms have been published and none of them have been shown to run in haptic time.

Two approaches to filling the grid can be taken: the grid can be “tight” or “loose”[9]. In the tight version of the grid each of the elements being inserted are placed into every cell that they touch. If you are placing large objects into a fine grid, this can lead to many references pointing at the same object, resulting in many more insertions into the grid. This can be detrimental to the grid’s build performance, depending on the intended use[9]. However, taking a tight approach can provide substantial benefits when using the grid later on, especially if it is being used for a data comparison application, like collision detection. The alternative placement method is known as the "loose" approach. In a loose grid, only one reference per object is placed in the grid. Which cell it is placed in depends upon the implementation, but the centre of the object or a specified vertex could be used[9].

Despite the difficulties involved with modifying a grid structure after construction, especially if an object moves beyond the bounds of the existing grid, dynamic grid structures have been presented in the literature. Two algorithms were presented by Reinhard et al.[116]. The first system utilised a system of two bounding boxes, a *logical* box and a *physical* box. When objects in the scene moved out of the physical box, rather than rebuild the physical grid structure, the logical box is extended to encompass them. The object would then be placed in the physical grid, as if the grid wrapped around. Therefore, the grid can be expanded without performing any costly memory reallocations. Objects were tagged if they were in *physical* or *logical* space, so as to accelerate ray traversal. Figure 3.1 shows how the logical box expands the grid.

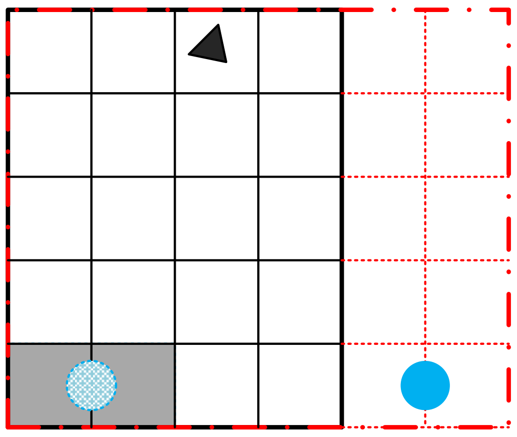


Fig. 3.1 Expanding a regular grid by using a logical grid. When an object moves (in this case a blue circle) out of the grid (indicated by the black mesh), the logical grid (indicated by red dots) is expanded to incorporate the object. Rather than reconstruct the entire grid, the object is then placed into the original grid as if it wrapped around. In this case, the blue circle is placed into the shaded cells.

The second structure Reinhard et al.[116] proposed is described as a hierarchical grid, designed to ensure that large objects placed within the grid did not occupy a large number of cells (Reinhard's structures were both presented as tight grids), leading to a large number of cells needing to be updated if the object moved. The concept is like an octree in that each cell of the grid is a leaf node of the data structure, however, Reinhard's implementation allows objects to be placed in non-leaf nodes if they are resident in all of the cells below that node, effectively reducing the number of insertions/deletions into the grid.

When compared with a regular non-dynamic grid, both of Reinhard et al.[116] systems were shown to suffer a performance penalty whilst rendering a static scene, owing to a more complicated grid traversal algorithm. The dynamic grids were shown to render an animation in real time, but the frame rate was adversely effected after a moderate number of grid modifications took place.

Other researchers[25,65,147] found that because of simplicity and speed of rebuilding the grid, using a dynamic structure is unnecessary. This has led to research into the best way to quickly rebuild a grid.

3.2.1 Parallel Construction

To achieve the fastest construction time possible, the grid will be built in parallel. One of the main difficulties with parallelising grid construction algorithms is dividing the work load evenly amongst a large number of threads - potentially thousands on a modern graphics card. Different approaches have been taken to accomplish this.

An early parallel grid construction algorithm, designed for the CPU, was presented by Ize et al.[60]. Ize et al.[60] describe several methods of parallelising regular grid construction designed for use with a conventional multiprocessing architecture, rather than the SIMD architecture used within most GPUs. Ize et al.[60] focused on the insertion of triangles into the grid, as they used the structure for ray tracing. Three approaches were described:

- Sort-First: Each cell is assigned a thread, which locates the triangles that are resident within it and stores them. The primary advantage of this approach is that no write-conflicts occur, because each cell will only be written to by one thread. The disadvantage is that every triangle within the scene will be tested by every thread, leading to a large number of memory accesses that are avoided using the other approaches.

- Sort-Last: Each thread is assigned a set of triangles, and calculates which cells they are resident in. This approach counters the disadvantage of the first approach - each triangle is only accessed once, however write-conflicts are now possible; if many threads try to write to the same cell, the memory will bottle-neck, reducing the algorithms performance.

- Sort-Middle: The middle ground between the sort-first and sort-last approaches. Each thread is responsible for a set of triangles, which it divides into parts. The triangles are then passed to whichever thread is responsible for the cells to which they are assigned, which performs the insertion. This approach combines the advantages of the sort-first and sort-last methods - each triangle is only accessed once and each cell will only ever be accessed by one thread. The drawback of this approach is the buffering fragments between the threads, which can be costly.

Of the three methods presented by Ize et al.[60], the best performing algorithm was the sort-middle approach, because load balancing is straight forward, and no scattered read/writes occur, which is ideal for the architecture they used. It is unlikely that a GPU implementation of the same algorithm will yield the same results due to architectural differences.

The other approaches will port to the GPU rather better, however sort-last is likely to give the best overall performance on a GPU because it provides the finest granularity of parallelism; it is unlikely that there will be more cells than objects to be placed within the grid, although this depends on the grid resolution.

Sorting has been explored as a method of creating regular grids. The broad idea presented by Kalojanov and Slusallek[65] and Lagae and Dutré[76], with a GPU implementation demonstrated by Green[44] is simple - have an array of objects, work out which cell of the grid they should be in and specify that as the object's "cell ID." All that is then required is to sort the list of objects by cell ID and sweep along the array storing the indices at which each cell starts and ends.

Green[44] presented a second construction algorithm, that utilises atomic operations to construct the grid. Within the second algorithm, a fixed amount of memory is allocated per cell, and a second "cell-counter" array, equal in length to the number of cells within the grid, keeps track of how many objects are in each cell. Each object to be inserted in the grid atomically increments the cell counter, giving it a position within the respective cells buffer. The object's index is then recorded in the correct position in the grid (This algorithm is discussed in detail in Section 3.4.1). Green's implementation produced a loose grid that utilised features of the dataset: a number of equally sized spheres which never overlapped, to keep the memory consumption low. By setting each side of the cell to be equal in size to the diameter of the spheres used within the grid, each cell will only contain the centre points of four spheres at most.

Barbieri et al.[9] presented an algorithm to construct a regular grid on the GPU. The grid is implemented as a matrix of linked lists. The authors declare an array of length equal to the number of cells in the grid they are creating. This array is called the *head*. To place an object in the grid, an atomic exchange function is used on the relevant cell's head. The old cell head is stored with the point itself, and the index of the point performing the operation replaces it. In this way, each cell will be pointing to a linked list which can then be traversed to access the points within that cell. Barbieri et al.[9] showed that the linked list construction algorithm outperformed both of the approaches presented by Green[44] during a particle simulation. Note, however, that the contents within the grid cells are not contiguous in memory, which could adversely effect runtime performance in some applications.

3.2.2 Hierarchical Data Structures

As mentioned previously, hierarchical data structures can also be used to divide 3D space. These structures organise data into a tree-like layout. The most primitive tree structure can be defined as a collection of nodes linked together in such a way that no node is duplicated in the tree, each node has only one parent, and no node references its parent or any of its grandparents. Tree structures have been used to accelerate a variety of applications, including interactive molecular docking[57], collision detection[156] and ray tracing[157]. The main advantage trees have over conventional lists is an average case time complexity of log(n) for search, insertion and deletion. This is a result of only having to follow one path from the root node to the required leaf node, rather than scan all of the elements in an array.

When used to partition 3D objects, the tree is used to divide up the spatial area into smaller areas. Generally, this allows empty space to be traversed more quickly than it would be in a regular grid, as the grid resolution of the empty space is lower than the resolution of space with an object in it. Trees do suffer a poorer access time than the regular grid (worst case O(N) vs O(1)) and are also more complicated to build.

The use of hierarchical data structures is common in graphics applications, therefore there has been some research into using them with moving 3D scenes.

BVH

The first GPU-based construction algorithm for creating a BVH was presented by Lauterbach et al.[80]. They coined their data structure the “Linear Bounding Volume Hierarchy.” First they compute a Morton index code from each objects position in space, then they sort the geometry based on the bits of their Morton codes. Next, they calculate the dividing lines between the nodes, in parallel, and order them such that they have a list of splits at each level in the tree. Finally, they convert this list into a form that is usable as a top-down hierarchy, during which they compute the correct bounding boxes for each node. An investigation by Karras and Aila[67] demonstrates that this LBVH construction method can complete within 2 ms, however they also highlight that LBVHs suffer a performance penalty when used, compared to true BVHs.

Other GPU-based BVH construction algorithms have been presented, including the work by Garanzha et al.[40] and Karras and Aila[67]. The algorithm presented by Karras and Aila[67] was fast: their approach involved constructing a low quality LBVH, and then optimising it in parallel. Despite being fast to construct, and providing a significant speed up in their ray-tracing project, none of their construction tests demonstrated construction completing in under 1 ms, even using the powerful Nvidia Titan.

Octree

Another tree based spatial partitioning structure which has been demonstrated as effective in the field of molecular docking[57], is the octree. An octree is a tree in which each non-leaf node must have eight children nodes. It is used in 3D space to subdivide an area into eight equally sized smaller areas. Figure 3.2 shows how an octree divides Cartesian space up.

Karras[66]' method for creating an octree on the GPU is similar to the BVH construction algorithm presented by Lauterbach et al.[80], Karras[66]' algorithm requires the Morton keys, assigned to each triangle, to be sorted. They they construct a binary radix tree in parallel. Once this tree has been created, a parallel prefix sum is performed in order to allocate tree nodes, and then the parent of each node is found, resulting in an octree. This method has proven to be exceptionally quick on modern graphics hardware, as a result of very little synchronisation between threads. Despite this, however, Karras[66]' results show that the construction time is still an order of magnitude

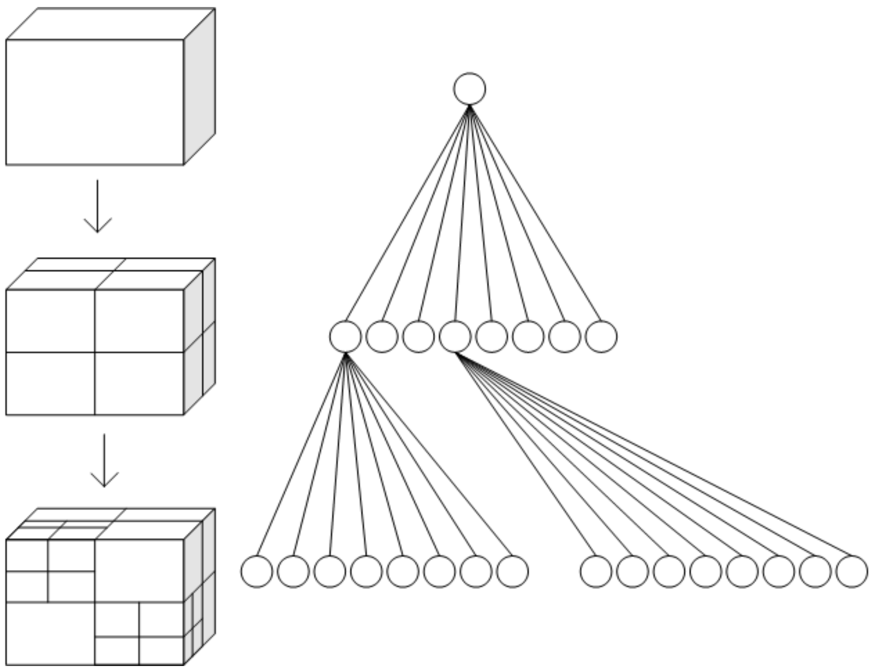


Fig. 3.2 How an octree divides up space. Each non-leaf node has 8 children.

slower than this projects target of 2 ms. Because of this, and also as a result of Iakovou et al.[57], demonstrating that the intermolecular force calculation that will utilise the data structure executes more quickly when used with a regular grid, rather than with a tree based structure, development of a fast construction algorithm for a hierarchical data structure is not undertaken here.

In the following section, three different ways a regular grid can be stored in memory are presented, and construction algorithms for each method are formally described. The linked list system presented by Barbieri et al.[9] and the atomic grids described by Green[44] are included. Following this, each of the grid construction methods will be benchmarked with molecular data in order to determine which will offer the best performance for the use intended in this thesis.

3.3 Methods

One of the major differences between each of the grid construction algorithms is the memory layout of the final structure. The simplest memory layout is pictured in Figure

3.3. This memory representation will be referred to as the *fixed grid* throughout the rest of this thesis. Within a fixed grid, the capacity of each of the cells is the same. This allows each cell in the grid to be accessed directly, through use of Equation 3.1.

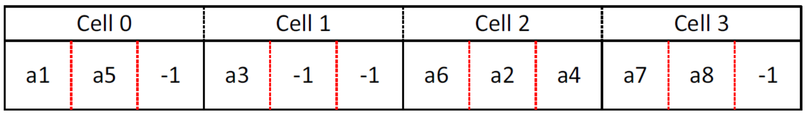


Fig. 3.3 A representation of a one dimensional fixed grid containing eight elements (a1...a8). Note how each cell is the same size. -1 denotes unused allocated memory within the grid.

The function

$$c_i = (z \times x_{cc} \times *y_{cc}) + (y \times x_{cc}) + x \tag{3.1}$$

can be used to calculate the memory index of an object in a one dimensional array from its three dimensional grid cell coordinates: c_i denotes the memory address of cell xyz. x_{cc} and y_{cc} denote the number of cells in the x and y planes of the grid. This equation allows each cell to be accessed with one operation; useful in both the build and use stages of the application. The drawback to the fixed grid is memory inefficiency; because all the cells have to be the same size, they all have to be the size of the cell containing the most elements. This results in a large amount of allocated memory being unused. Green[44] utilised this method within a particle system (See Section 3.2.1)).

A more memory efficient solution was presented by Lagae and Dutré[76]. The compact grid, shown in Figure 3.4, is a structure consisting of two arrays. The first array acts as a lookup table for the cell start indices within the second array. To access any individual cell of the grid, first the cell start location has to be read from the lookup array and then the data array accessed to retrieve the information. This data representation is more memory efficient and less expensive to enlarge if necessary, however during construction and use the structure is less operation efficient. Green[44] suggested a method for creating a loose compact grid on the GPU, again utilising atomics. This compact grid algorithm requires two passes of the data; the first pass counts how many atoms are in each cell, the second pass uses this data to construct the grid. Compact grids can also be created by sorting the data and then finding the points in the sorted list at which the cells change.

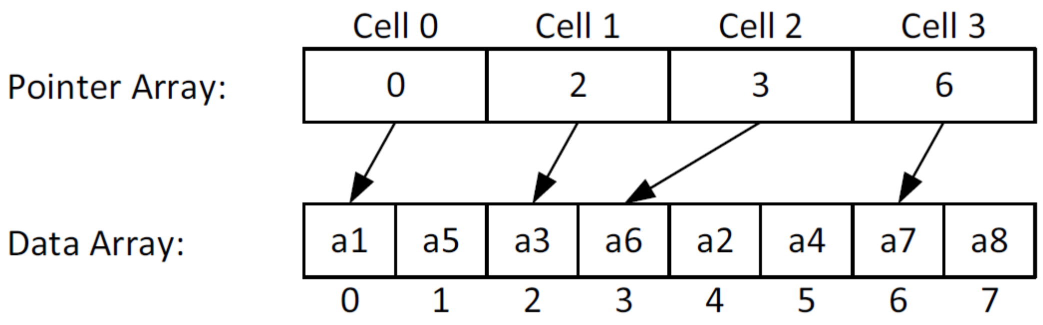


Fig. 3.4 A representation of a compact grid containing eight elements (a1...a8). Note how the grid is closely packed, leaving no wasted space. Also the figure highlights the two step lookup method, demonstrating the potential inefficiency of the structure.

Barbieri et al.[9] presented an algorithm in which the regular grid is represented as linked lists, and demonstrated how to construct it on the GPU. The linked list grid (pictured in Figure 3.5) utilises two arrays. The first array contains a pointer to the first object within that cell. The second array contains all the data contained within the grid. Each element in the data array also contains both the data within the grid and a pointer to the next element in the linked list.

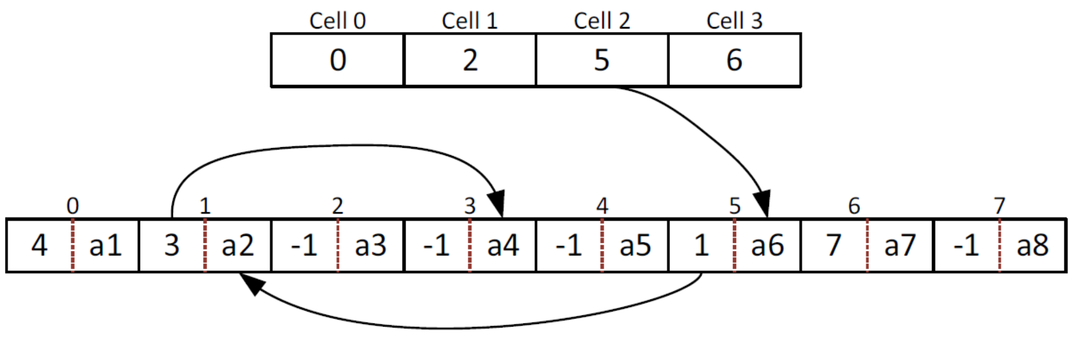


Fig. 3.5 A visualisation of a linked grid containing eight elements (a1...a8). Note how, like the compact grid, the linked grid is closely packed, leaving no wasted space.

To achieve the best possible overall performance, a balance between grid construction time and how quickly the grid can be queried must be found. Video memory is also a scarce resource in all but the most expensive modern graphics cards. For this reason memory efficiency should also be considered when analysing each of the regular grids. Each of the structures presented has limitations: the fixed grid is not memory efficient, the compact grid requires multiple passes of the data to construct and the information in the linked grid is not stored sequentially in memory.

In this project, the regular grid will be used with molecular data. In order to limit the number of insertions into the grid, a design decision was made to set the minimum cell size to $2S_r$, where S_r is the radius of a Sulphur atom, which is 1.8 Å. Sulphur was chosen as it has the largest radius of the most commonly found atoms within PDB files, therefore, setting the minimum cell size to $2S_r$ insures that the majority of atoms will be resident in at most 8 grid cells.

3.4 Implementation

Three implementations of the grid were tested, one for each type of grid. All the different algorithms begin in the same way. These steps are formally described in Algorithm 1.

Algorithm 1 prepareForGridConstruction(**atoms**,*n*,*cs*) **return** *mins*, *slen*

```
Require: atoms {list containing XYZ coordinates of n atoms, in protein.}
Require: cs {Desired cell size of constructed grid.}
1: $[mins, maxs]$ ←parallelMinMax(atoms) {minimum and maximum in x,y and z planes}
2: $slen \leftarrow maxs - mins$ {Length of bounding box sides}
3: for $i \leftarrow 1$ to 3 do
4:     $slen(i) \leftarrow ceil(slen(i)/cs)*cs$ {Round side lengths to next multiple of cs }
5: end for
6: $numCells \leftarrow product(slen)$
7: return [$mins$, $slen$]
```

The function `parallelMinMax` is described in Appendix A.

3.4.1 Fixed Grid

Both the tight and loose fixed grid construction algorithms are broadly similar. The only difference is how the atoms are placed within the grid structure.

Algorithm 2 createFixedGrid(**atoms**,n,cs,$alloc$) **return** $fixedGrid$, $cellAtomCounts$

```
Require: atoms {list containing XYZ coordinates, and radius of each atom in protein.}
Require: $cs$ {Desired cell size of constructed grid.}
Require: $n$ {length of list atoms.}
Require: $alloc$ {Capacity of each cell.}
1: $[mins, slen] \leftarrow$ prepareForGridConstruction(atoms,$n$,$cs$)
2: for $i \leftarrow 1$ to $numCells$ in parallel
3:    cellAtomCounts($i$) $\leftarrow 0$
4: end for
5: for all Atoms $a$ in atoms in parallel
6:    $s_r \leftarrow 1 + (\text{floor}(a.radius/cs))$
7:    $Cell_{mid} \leftarrow (a.xyz - min.xyz)/cs$
8:    for $i \leftarrow -s_r$ to $s_r$ do
9:       for $j \leftarrow -s_r$ to $s_r$ do
10:         for $k \leftarrow -s_r$ to $s_r$ do
11:            $cellX \leftarrow Cell_{mid}.x + i$
12:            $cellY \leftarrow Cell_{mid}.y + j$
13:            $cellZ \leftarrow Cell_{mid}.z + k$
14:            if isValidCell($cellX$, $cellY$, $cellZ$, $cs$, $mins$) then
15:               $cMin \leftarrow$ getCellMinPoint($cellX, cellY, cellZ, cs, mins$)
16:               $cMax \leftarrow$ getCellMaxPoint($cellX, cellY, cellZ, cs, mins$)
17:               if sphereWithinCell($a, cMin, cMax$) then
18:                  $cellId \leftarrow (cellZ * slen.x * slen.y) + (cellY * slen.x) + cellX;$
19:                  $gridIndex \leftarrow$ atomicAdd( $cellAtomCounts[cellId], 1$)
20:                  $gridIndex \leftarrow$ getGridPosition( $cellX, cellY, cellZ, alloc, gridIndex,$ slen)
21:                  $fixedGrid(gridIndex) \leftarrow a_{index}$
22:               end if
23:            end if
24:         end for
25:      end for
26:    end for
27: end for
28: return $fixedGrid$, $cellAtomCounts$
```

Algorithm 2 describes how to create a tight fixed grid, (shown pictorially in Figure 3.3). The loose grid algorithm would be identical, but without the `for` loops on lines 8, 9 and 10. Memory is allocated in order to allow aspects of the scene to move beyond the initial bounds of the regular grid. Allocating sufficient memory for the fixed grid is difficult, because all the cells have to be the same size as the largest and the number of atoms within the largest cell will not be known until runtime. The decision was taken to allocate a block of memory (how much memory being dependant on the number of atoms in the loaded structure), and divide it by the required number of cells. Within our implementation, the amount of memory allocated was enough to ensure that no cell will ever become full. In its worst case, this algorithm has a run time complexity of 9N, making it an O(N) algorithm, where N is the number of objects to be placed into the grid.

```
Algorithm 3 getGridPosition( $cellX, cellY, cellZ, mem, cellPos,$ slen) return $gridIndex$
Require: $cellX$ {Cell index on X axis}
Require: $cellY$ {Cell index on Y axis}
Require: $cellZ$ {Cell index on Z axis}
Require: $mem$ {amount of memory allocated per cell}
Require: $cellPos$ {Position the atom is to be placed inside the cell}
Require: slen {Number of cells in each side of the grid}
1: $gi \leftarrow (cellZ * mem * slen.x * slen.y) + (cellY * mem * slen.x) + (cellX * mem) + cellPos$
2: return $gi$
```

`GetGridPosition` (Algorithm 3) is a formal description of Equation 3.1, as used within the grid construction algorithms.

3.4.2 Compact Grid

The compact grid construction algorithm consists of multiple kernels, with global synchronisation points between them. Each step is listed in Algorithm 4.

Within Algorithm 4, `parallelPrefixSum` is called, this algorithm is defined in Appendix A. The other algorithms are presented within this section. Again, all memory is allocated before execution and sufficient memory is allocated to allow for atom movement beyond the initial bounding box.

Algorithm 4 createCompactGrid(**atoms**,*n*,*cs*) **return** *cellStartPoints*, *compactGrid*

```
Require: atoms {list containing XYZ coordinates of each atom in protein.}
Require: cs {Desired cell size of constructed grid.}
Require: n {length of list atoms.}
1: [cellAtomCounts, mins, slen] ← prepareForGridConstruction(atoms,n,cs)
2: [atomsInEachCell, atomCellPairs] ← calcCellContents() {Algorithm 5}
3: cellStartPoints ← parallelPrefixSum(atomsInEachCell)
4: compactGrid ← copyIntoCells(cellStartPoints, atomCellPairs, n) {Algorithm 6}
```

Algorithm 5 describes how the number of atoms resident in each cell is calculated. After it has been determined that an atom is within a cell, an atomic add operation is performed on that cells “count”. This value, and the `cellId` are then stored as a pair, to accelerate filling the grid structure later, after the cell start points have been calculated. This part of the compact grid's construction algorithm is, in its worst case O(N), where N is the number of atoms within the scene.

The final part of the compact grid's construction is copying the atoms into the grid. This is achieved with Algorithm 6. Algorithm 6 uses the cell ID and location pair, stored within Algorithm 5, to rapidly copy the atom information into the grid. Storing the information from the earlier method uses more memory, but removes the need to perform any further atomic operations. In its worst case, Algorithm 6 is also an O(N) algorithm. This, combined with the O($Mlog(M)$) operations, where M is the number of cells in the grid, from the prefix scan stage makes, the compact grid a worst case O(N+$Mlog(M)$) algorithm, with a complete run-time complexity of $17N+Mlog(M)$, a higher order algorithm than both the fixed and linked grids.

3.4.3 Linked Grid

The final algorithm implemented is the linked grid algorithm, defined originally by Barbieri et al.[9]. Like the fixed grid, a linked grid can be constructed with a single kernel, described in Algorithm 7. As with the other two algorithms, memory is allocated beforehand, including enough to allow for an increase in the number of cells.

Algorithm 5 calcCellContents(**atoms**,n,cs,$alloc$) **return cellAtomCounts** , **atomCellPairs**

```
Require: atoms {list containing XYZ coordinates of each atom in protein.}
Require: cs {Desired cell size of constructed grid.}
Require: n {length of list atoms.}
Require: alloc {Capacity of each cell.}
 1: for $i \leftarrow 1$ to $numCells$ in parallel
 2:    cellAtomCounts($i$) $\leftarrow 0$
 3: end for
 4: [cellAtomCounts, $mins$, $slen$] $\leftarrow$ prepareForGridConstruction(atoms,$n$,$cs$)
 5: for all Atoms $a$ in M in parallel
 6:    $Cell_{mid} \leftarrow (a.xyz - min.xyz)/cs$
 7:    $s_r \leftarrow 1 +$ (floor($a.radius/cs$))
 8:    $acPairWid \leftarrow a_{index} * 8$
 9:    for $i \leftarrow -s_r$ to $s_r$ do
10:       for $j \leftarrow -s_r$ to $s_r$ do
11:          for $k \leftarrow -s_r$ to $s_r$ do
12:             $cellX \leftarrow Cell_{mid}.x + i$
13:             $cellY \leftarrow Cell_{mid}.y + j$
14:             $cellZ \leftarrow Cell_{mid}.z + k$
15:             if isValidCell($cellX$, $cellY$, $cellZ$, $cs$, $mins$) then
16:                $cMin \leftarrow$ getCellMinPoint($cellX, cellY, cellZ, cs, mins$)
17:                $cMax \leftarrow$ getCellMaxPoint($cellX, cellY, cellZ, cs, mins$)
18:                if sphereWithinCell($a, cMin, cMax$) then
19:                   $cellId \leftarrow (cellZ * slen.x * slen.y) + (cellY * slen.x) + cellX$;
20:                   $gridIndex \leftarrow$ atomicAdd( $cellAtomCounts[cellId], 1$)
21:                   $atomCellPairs(acPairWid)$ = [$cellId$, $gridIndex$]
22:                   $acPairWid = acPairWid + 1$
23:                end if
24:             end if
25:          end for
26:       end for
27:    end for
28: end for
29: return $cellAtomCounts$, $atomCellPairs$
```

The `atomicExch` function used within Algorithm 7 is an inbuilt function in the CUDA library. The function works by taking an item in an array and swapping it with the value specified in a single operation. The original value is then returned. As with the fixed grid, the linked grid construction algorithm has a run-time complexity of 9N, making it an order O(N) algorithm in its worst case. This means that in the build stage,

Algorithm 6 copyIntoCells(**atoms**, **cellStartPoints**, **atomCellPairs**) **return** *compactGrid*

```
Require: atoms {list containing XYZ coordinates of each atom in protein.}
 1: for all Atoms $a$ in M in parallel
 2:    $acPairWid \leftarrow a_{index} * 8$
 3:    for $i \leftarrow 0$ to 8 do
 4:       $pair \leftarrow = atomCellPairs(acPairWid + i)$;
 5:       if $pair.x$ != -1 then
 6:          $index \leftarrow$ cellStartPoints[$pair.x$] + $pair.y$
 7:          $compactGrid[index] \leftarrow a_{index}$
 8:       else
 9:          break
10:       end if
11:    end for
12: end for
```

the fixed grid and linked grid should perform similarly, with the compact grid taking longer.

3.5 Method Analysis

There are three main areas that need to be assessed in order to determine which of the regular grid construction algorithms will offer the best performance for the uses intended in this thesis. They are: How long each structure takes to construct, the performance of each structure when it is used and the memory consumption of each of the structures.

Construction time To determine the execution time of each of the construction algorithms, the run time of each algorithm will be recorded ten thousand times, and the average determined.

As the cell size used within the grids is likely to have an effect on the performance of the structure, multiple cell sizes will be tested. The cell sizes used within the tests will be a multiple of the radius of a Sulphur atom (1.8 Å), starting at two. This ensures that the majority of atoms are placed in at most eight cells. For all of the structures tested within this section, memory is preallocated where possible, however if buffers

Algorithm 7 createLinkedGrid(**atoms**,n,cs) **return** $listStartAddresses, linkedGrid$

```
Require: atoms {list containing XYZ coordinates of each atom in protein.}
Require: $cs$ {Desired cell size of constructed grid.}
Require: $n$ {length of list atoms.}
Require: $alloc$ {Capacity of each cell.}
1: [cellAtomCounts, $mins$, $slen$] $\leftarrow$ prepareForGridConstruction(atoms,$n$,$cs$)
2: for all Atoms $a$ in M in parallel
3:    $s_r \leftarrow 1 + (\text{floor}(a.radius/cs))$
4:    $Cell_{mid} = (a.xyz - min.xyz)/cs$
5:    $acPairWid = a_{index} * 8$
6:    for $i \leftarrow -s_r$ to $s_r$ do
7:       for $j \leftarrow -s_r$ to $s_r$ do
8:          for $k \leftarrow -s_r$ to $s_r$ do
9:             $cellX = Cell_{mid}.x + i$
10:            $cellY = Cell_{mid}.y + j$
11:            $cellZ = Cell_{mid}.z + k$
12:            if isValidCell($cellX$, $cellY$, $cellZ$, $cs$, $mins$) then
13:               $cMin \leftarrow$ getCellMinPoint($cellX, cellY, cellZ, cs, mins$)
14:               $cMax \leftarrow$ getCellMaxPoint($cellX, cellY, cellZ, cs, mins$)
15:               if sphereWithinCell($a, cMin, cMax$) then
16:                  $cellId \leftarrow (cellZ * slen.x * slen.y) + (cellY * slen.x) + cellX$;
17:                  $gridBody(2 * acPairWid) \leftarrow a_{index}$
18:                  $gridBody(2 * acPairWid + 1) \leftarrow$ atomicExch(($listStartPointers[cellID]$), $acPairWid$);
19:                  $acPairWid = acPairWid + 1$
20:               end if
21:            end if
22:         end for
23:      end for
24:   end for
25: end for
26: return $fixedGrid$, $cellAtomCounts$
```

need resetting to zero at the start of every build, the time taken to do this is included within the total runtime.

The execution time of each algorithm will be measured using the high-precision timer provided within the windows library, to insure any overheads resulting from launching the GPU kernels are included in the runtime.

Overall performance The construction time of the grid only tells half the story. The grids must also be efficient in use, in order to be useful for accelerating algorithms.

Therefore, to establish which of the grid structures provides the best overall performance, a simple locality search test was performed. The test involved counting how many atoms were within a set cut-off distance of each of the atoms within the scene. This test will be repeated multiple times, in order to determine how searching a larger number of smaller cells compares with searching a small number of large cells.

Using a locality search as a performance test will provide insight into how well each structure will perform if used in conjunction with the interaction force algorithm presented by Iakovou et al.[56], as the access patterns will be very similar.

Within these tests a tight grid is constructed because it is the more computationally expensive structure to build; if a tight grid can be constructed in haptic time, a loose grid can be as well.

Memory consumption As GPU memory is a scarce resource, the final aspect that will be assessed is the memory consumption of each grid. This can be determined using the NVIDIA profiling tools.

3.6 Results

Biomolecules taken from RCSB Protein Data Bank (PDB) will be used to perform the tests described in Section 3.5. PDB data is used as it will provide a realistic idea of the performance of the grids when used for the purposes intended in this thesis.

Table 3.1 lists the eleven proteins were taken from the RCSB protein data bank[14] for use within testing. These biomolecules were chosen because they vary widely in size and the number of atoms they contain per unit area; some contain hydrogen atoms which effectively increase the density of the structure, conversely, some only contain $C\alpha$ atoms, which reduces the number of atoms resident in each grid cell. Together they should provide a reasonable idea of how well each grid will perform in the majority of use cases.

PDB Code	Number of Atoms
1ANF	2860
1OMP	5737
5E0T	5804
1AF6	10050
1XI4	15300
4A97	25020
3E76	53970
1AON	58674
3JCU	59354
1FFK	64268
1HTQ	90672

Table 3.1 Information relating to the proteins used within the grid performance testing.

A desktop workstation equipped with an Intel i7 processor, 16 GB of RAM and an Nvidia GTX 980 GPU was used to perform the benchmarks presented in this Chapter.

3.6.1 Construction Time

For an initial, broad understanding of each algorithm's performance, the results of each of the cell sizes from the construction time test were then averaged together. The results can be seen in Figure 3.6.

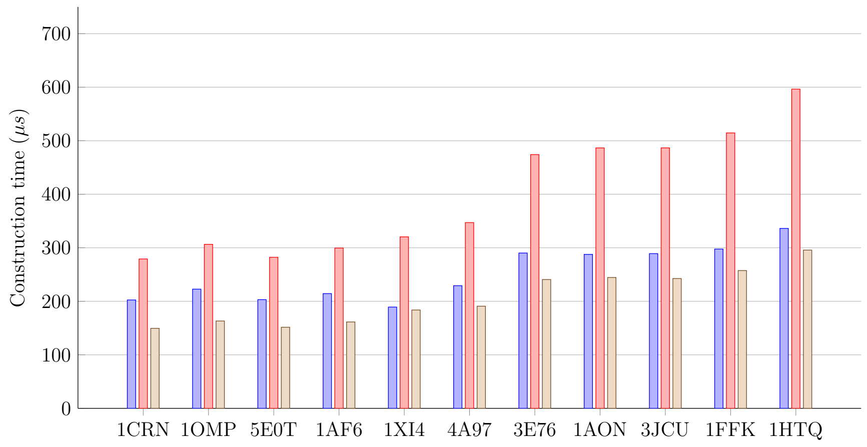


Fig. 3.6 Average grid construction times for each structure (blue: fixed, red: compact, brown: linked) when constructed over 11 different proteins.

Initial results show that, on average, the linked grid is the quickest to construct, closely followed by the fixed grid, with the compact grid taking the longest of all:

	$2S_r$ (3.6 Å)		$8S_r$ (14.4 Å)	
	Read (MB)	Write (MB)	Read (MB)	Write (MB)
Fixed	0.95	27.95	0.95	7.72
Compact	19.43	48.17	2.46	19.38
Linked	0.96	37.75	0.95	9.70

Table 3.2 Table showing total kernel global memory accesses in MB, for the different approaches, when constructing a grid for 3JCU. Two cell sizes are shown, $2S_r$ (3.6 Å) and $8S_r$ (14.4 Å).

twice as long as the linked grid in some of the tests. All of the grid construction algorithms complete in well under 1 ms, for all of the proteins tested, leaving at least 1 ms in which to perform computation with the grid, when using it with a haptic environment.

Part of the cause of the compact grid's comparatively poor performance is highlighted in Table 3.2. Because the algorithm requires two passes of the data to complete, rather than the single pass required by the other two algorithms, far more memory accesses are performed during grid construction. As highlighted in Section 2.4, memory operations, especially non-coalesced scattered read and writes, are comparatively slow operations. This, compounded with the fact that the compact grid construction algorithm does comparatively more work, results in a longer construction time.

To determine if the grid cell sized used during grid construction has a significant effect on performance, the build times of the largest cell sized used (14.4 Å) were subtracted from those of the smallest (3.6 Å), giving the difference. The results can be seen in Figure 3.7.

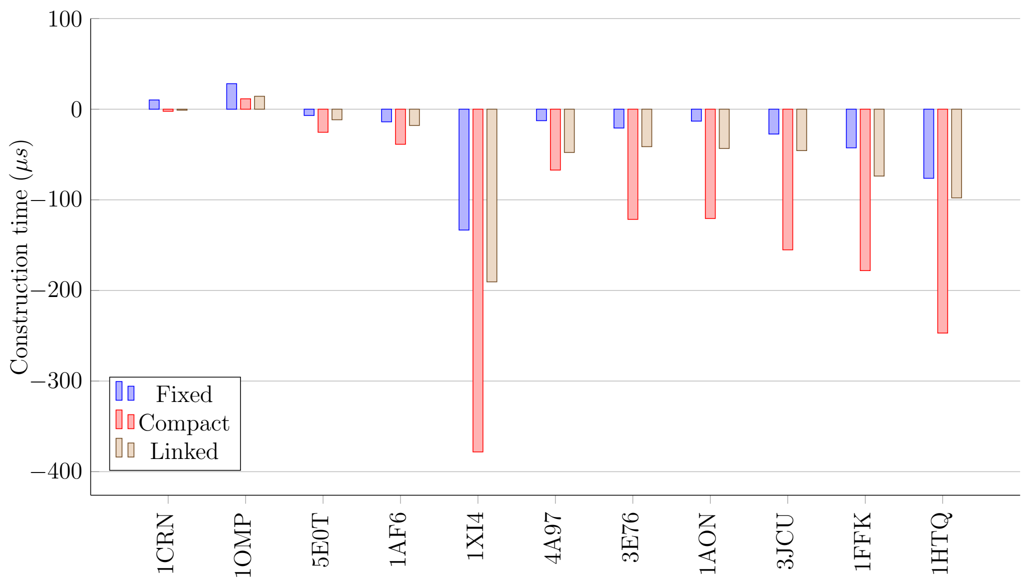


Fig. 3.7 Graph shows the time difference between constructing a grid with a cell size of 3.6 Å ($2S_r$) and one with cells 14.4 Å ($8S_r$) big, measured with the NVIDIA profiling tool.

Figure 3.7 shows that when the protein structure is small, the time difference between using a small and large cell size is minimal. When considering the larger structures, the compact structure does show a performance difference - using a larger cell size results in a shorter build time. The cause of this is visible in Table 3.2. The larger cell size results in fewer insertions into the grid, reducing the number of memory operations, leading to a better overall runtime. However, the inverse is likely to be true when using the structure, as each cell will contain more references. The larger cell size will result in more atoms being queried per cell traversed, potentially increasing the execution time of the algorithm using the grid.

Figure 3.7 shows a large construction time delta for the protein 1XI4. As Table 3.1 shows, 1XI4 only contains 15300 Atoms, far fewer than 1HTQ and 1FFK. Table 3.3 shows the dimensions of each protein's bounding boxes.

Table 3.3 shows that 1XI4 is a spatially larger structure than all of the others tested; the file used during testing only contains the Cα atoms of the structure. The result of this is a grid containing a large number of cells, each cell containing few atoms. The compact grid's performance suffers the most because of the second part of its runtime

PDB Code (Atoms)	X	Y	Z
1ANF (2860)	63.36	52.00	61.05
1OMP (5737)	53.41	58.85	74.40
5E0T (5804)	90.25	85.56	80.51
1AF6 (10050)	84.99	88.39	72.51
1XI4 (15300)	387.95	350.85	432.21
4A97 (25020)	246.24	256.47	121.06
3E76 (539708)	151.94	152.71	155.53
1AON (58674)	145.66	227.9	224.02
3JCU (59354)	204.21	236.99	114.75
1FFK (64268)	178.76	223.65	215.85
1HTQ (90672)	223.11	152.17	215.19

Table 3.3 Table shows the lengths of each side of each protein's bounding box.

complexity function - there is Mlog(M) additional work performed in comparison to the other structures.

A note on a sorting based approach: Within Section 3.2, it was mentioned that the compact grid can be created by calculating which cells each object is a resident of, and then sorting the list by the cell ID. This approach is not implemented, as an early investigation showed that the time taken to place and sort even a modest list of atoms was too long for use with a haptic device. Figure 3.8 shows the time taken to perform the placement and sorting stage, using the GPU based thrust library, of a sort based tight compact grid construction algorithm for each of the test proteins. The atomic based compact grid discussed in Section 3.4.2 results are included for comparison. A cell size of $2S_r$ was used in both cases.

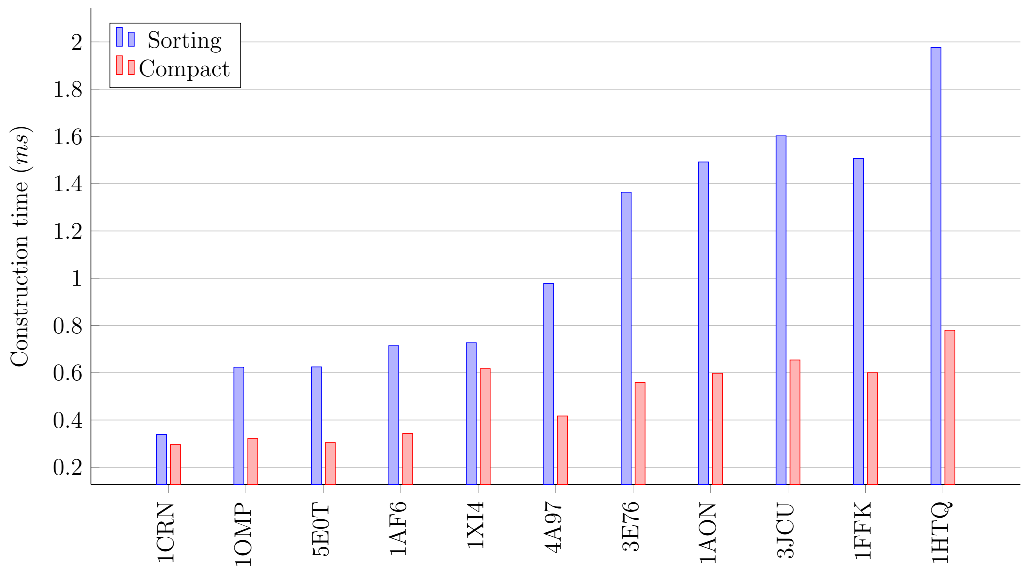


Fig. 3.8 Graph compares the first stage of the sorting based compact grid construction with the total runtime of the atomic based compact grid approach

The results in Figure 3.8 show that the sorting approach is comparatively slow, even when excluding the final stage of the build algorithm. All of the larger proteins failed to complete the sort stage in 2 ms, and all of the results show a longer run time than the entire atomic based compact grid construction algorithm. For this reason, continual development of the sort-based approach was dropped.

3.6.2 Overall Performance

The locality search test was run with three different cut off distances: 1.5 Å, 6.0 Å and 12.0 Å. Using 1.5 Å as a cut-off distance means that for all grids, a maximum of one cell in each direction needed to be searched. The larger cut-off distance of 12 Å was selected because it is often used as a cut-off distance in MD[114]. A 6 Å test was also run, as it falls nicely between the two other test sizes. Figure 3.9 shows the average results of all the tests. These results show that the compact grid offers the best performance in use. Therefore, it is likely that, should the grid be used more than once per reconstruction, the compact grid will offer the best all round performance, despite

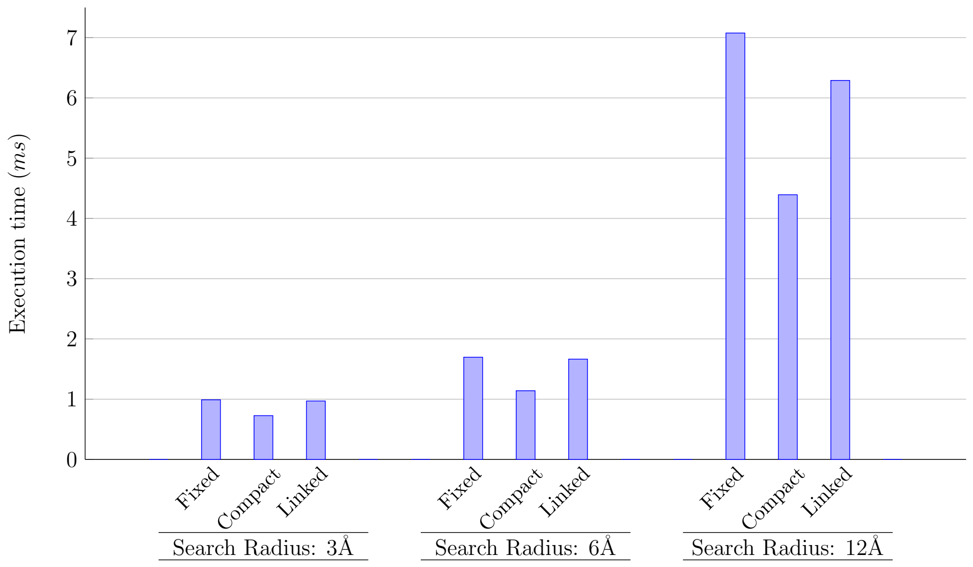


Fig. 3.9 The average run runtime of the nearest neighbour search time across all structures, for each of the different search radii.

being the slowest to construct. Figures 3.10 to 3.20 show the test run times added to the respective grid build time, for each of the tested proteins.

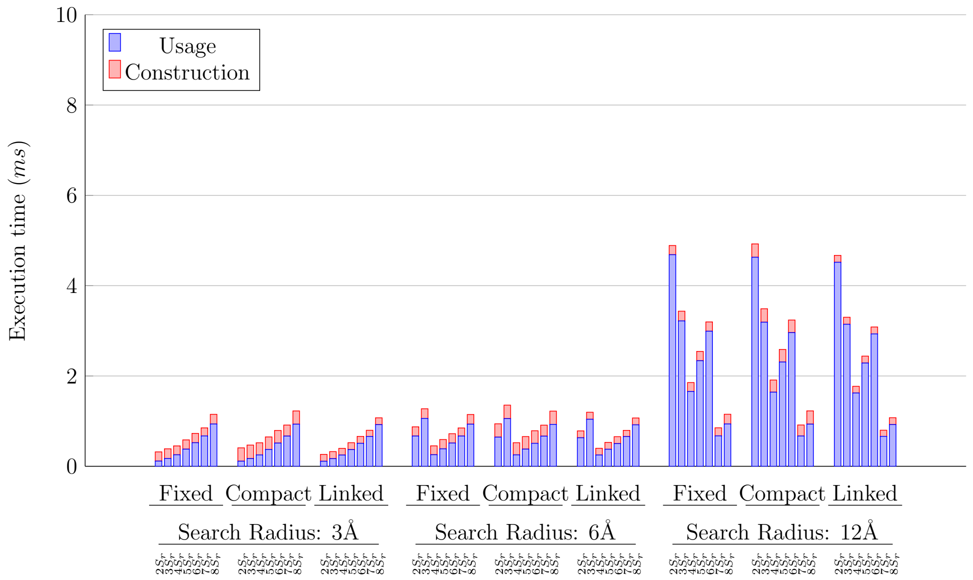


Fig. 3.10 Results for protein 1ANF - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

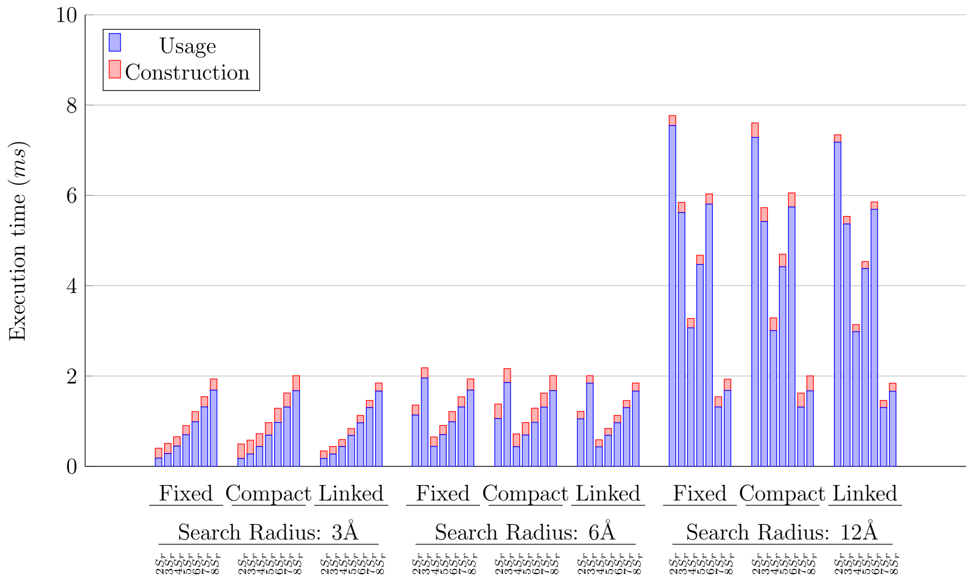


Fig. 3.11 Results for protein 1OMP - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

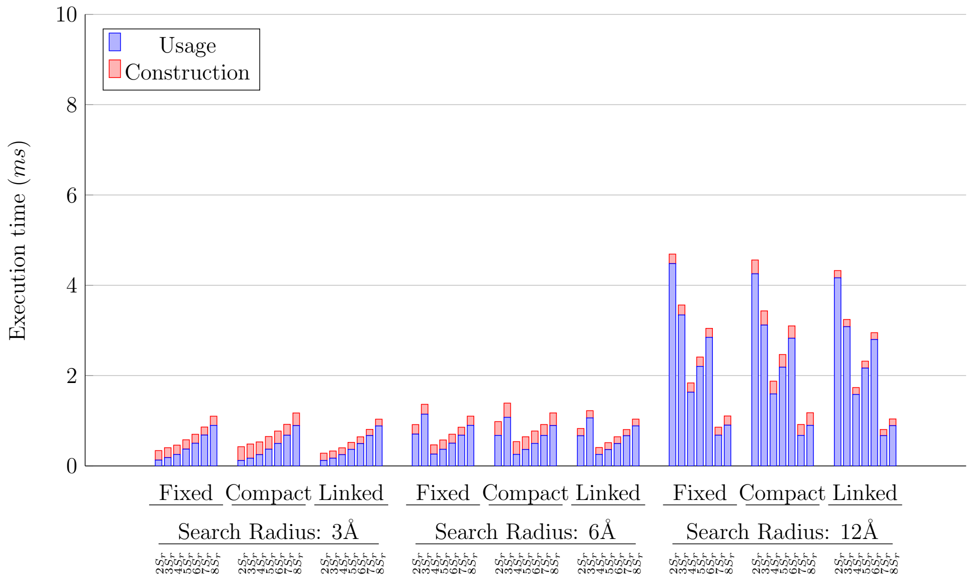


Fig. 3.12 Results for protein 5E0T - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

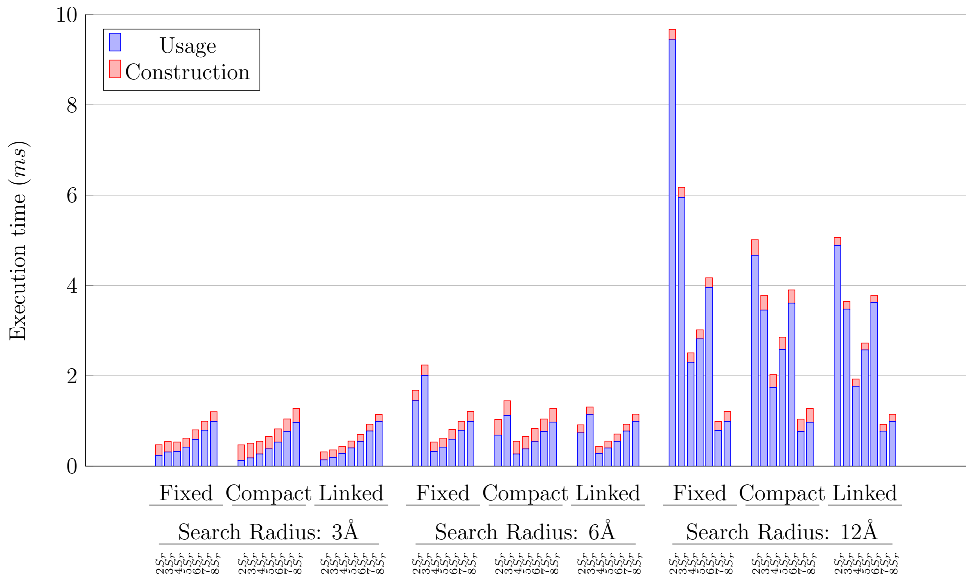


Fig. 3.13 Results for protein 1AF6 - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

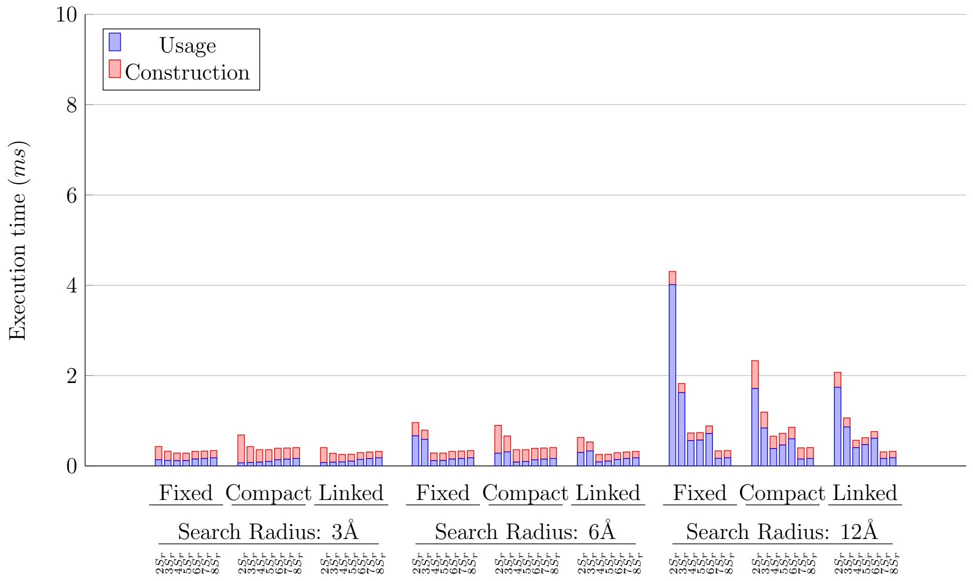


Fig. 3.14 Results for protein 1XI4 - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

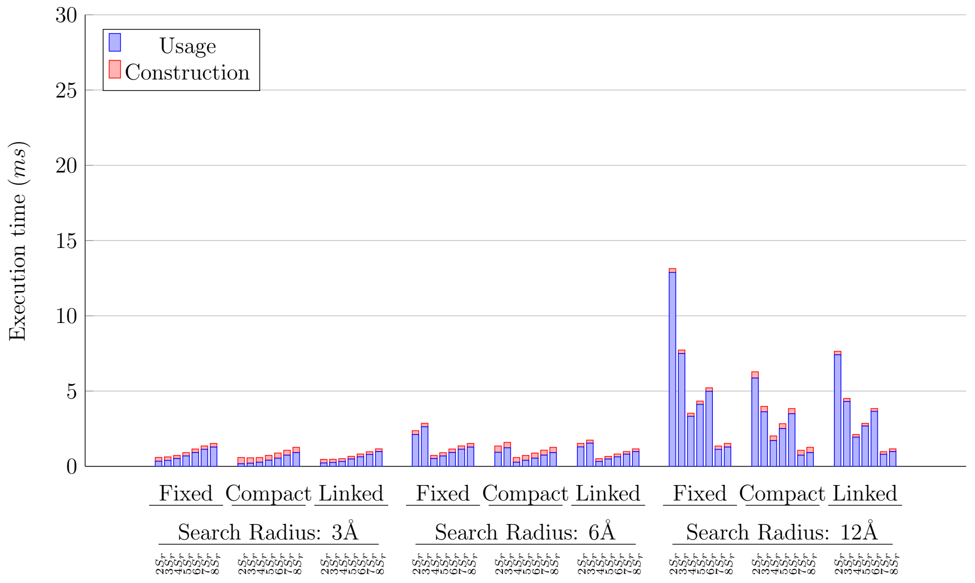


Fig. 3.15 Results for protein 4A97 - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

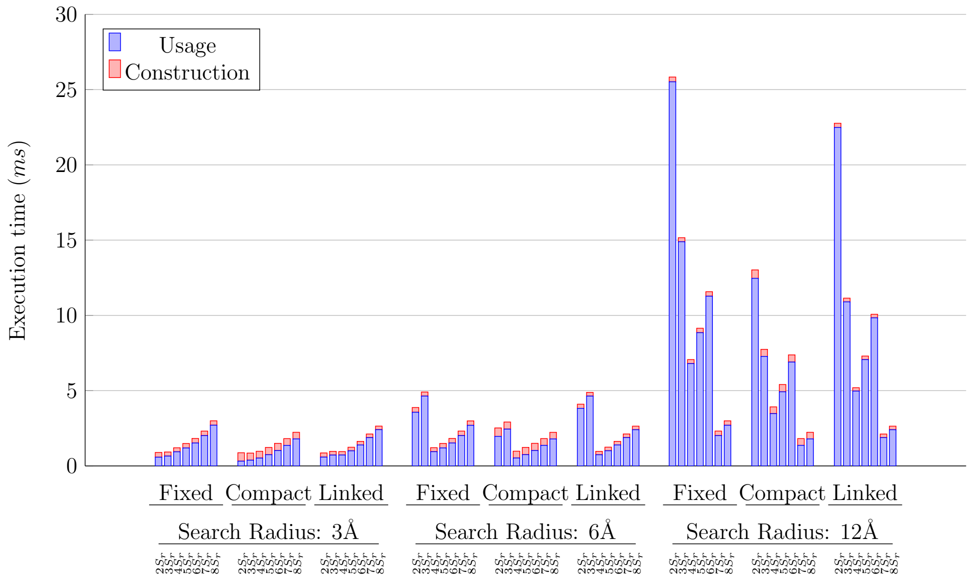


Fig. 3.16 Results for protein 3E76 - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

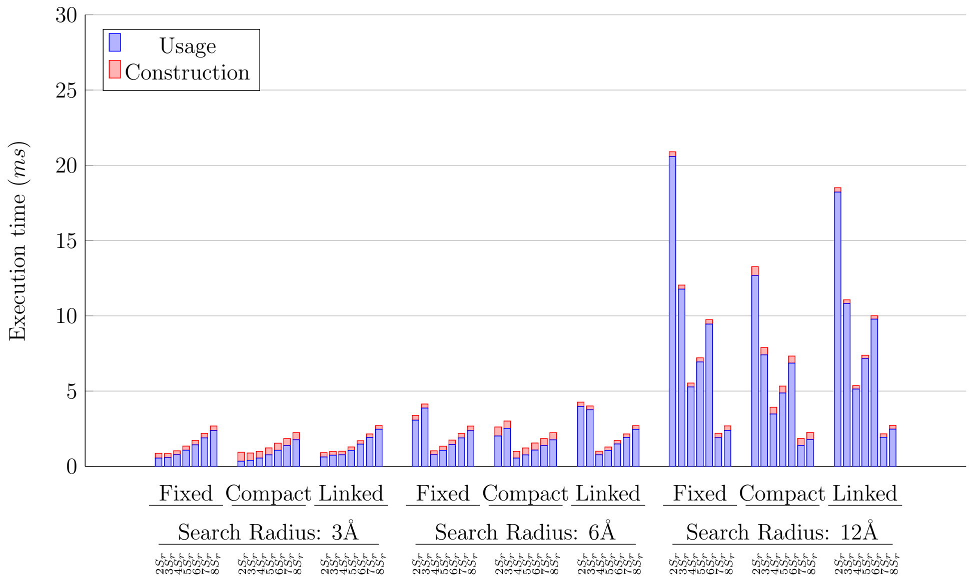


Fig. 3.17 Results for protein 1AON - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

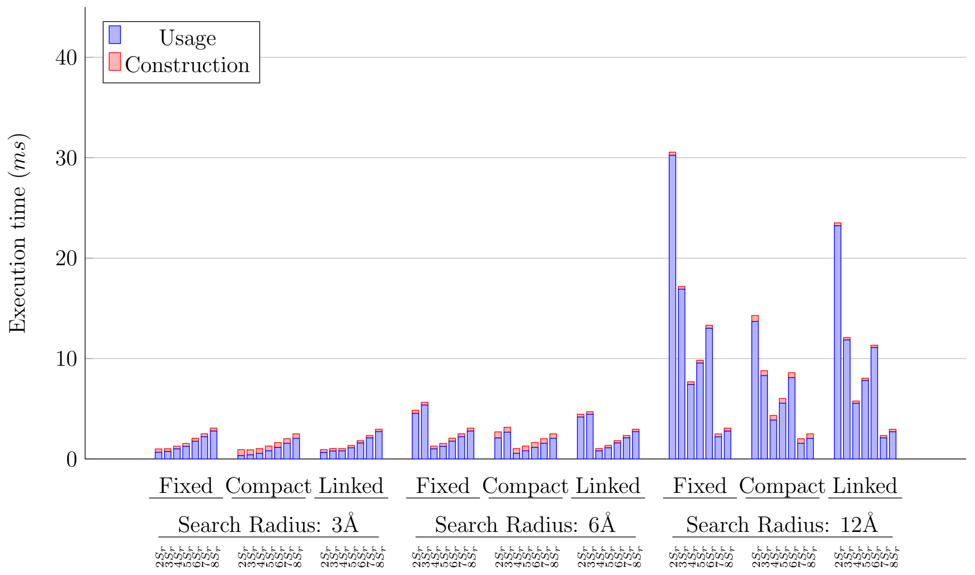


Fig. 3.18 Results for protein 3JCU - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

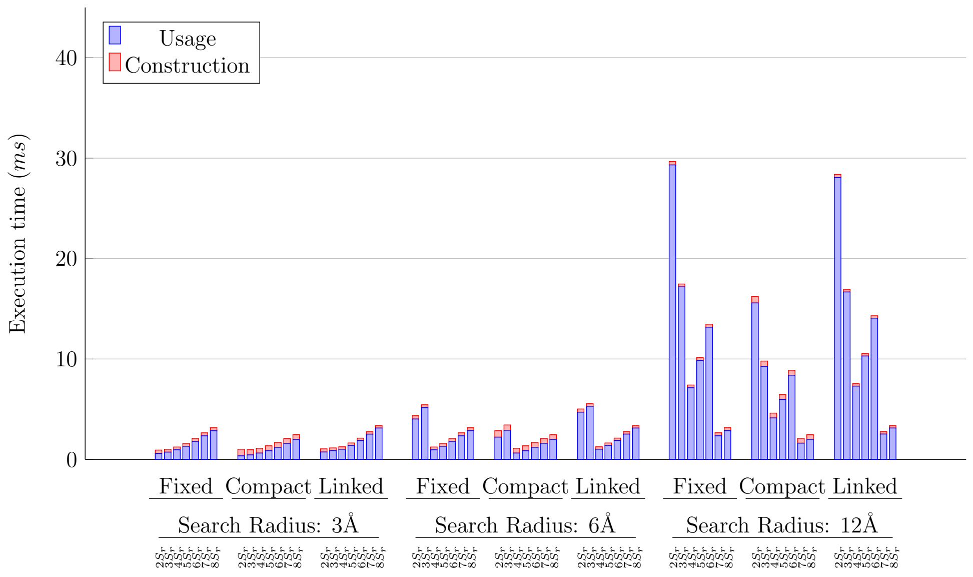


Fig. 3.19 Results for protein 1FFK - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

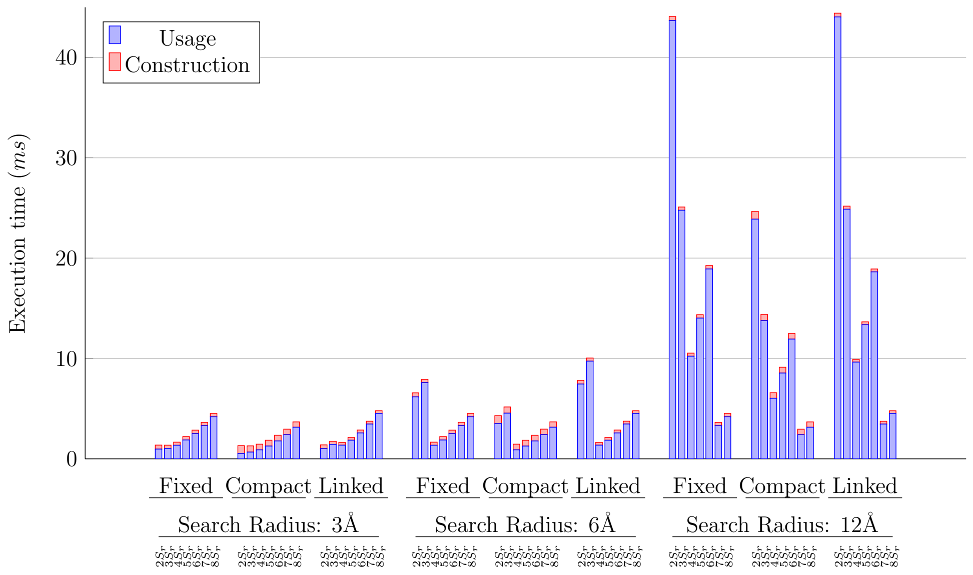


Fig. 3.20 Results for protein 1HTQ - Graph shows the build and use time for all tested combinations of grid type and construction cell sizes. Three search radii are shown: 1.5 Å, 6.0 Å and 12.0 Å.

The results of all the proteins reveal the same pattern: The smallest cell size that is greater than the search radius is the fastest to build and run, regardless of the grid construction algorithm used or the protein that is being placed into the grid. This fact is most clearly evident in the results of the 6 Å test - there is a clear reduction of time between the cell size of $3S_r$ (5.4 Å) and $4S_r$ (7.2 Å). The same behaviour is evident in the 12 Å test: a clear reduction in run time is visible between the cell sizes of $6S_r$ (10.8 Å) and $7S_r$ (12.6 Å). From these tests, it is suggested that the cell size used has a larger impact on the overall performance of the structure than the grid construction algorithm - tuning the cell size to the application, in order to limit the number of cells searched during use is therefore important.

Comparing the performance of each of the grids in the local atom count experiment shows that, when the protein is small, the linked grid marginally outperforms the other two algorithms, although the overall performance of the three structure is similar. When the protein and search radius are larger, the compact grid yields the better result by a fair margin. This demonstrates that the compact grid is more efficient to use than the linked and fixed grids. As a "one size fits all" structure, the compact grid is the most appropriate, as the time penalty for using with smaller structures is less than the increase in run time resulting from use the linked grid with larger structures.

The overall performance of each of the grids is very similar, however of the three, the performance of the fixed grid is most unexpected; as each cell can be accessed in O(1) time, it should be the fastest. Instead, it is the slowest of the three grids. Further analysis suggests that the memory layout of the fixed grid is to blame. Owing to the fact the grid is comprised a large block of memory, with the data interwoven with empty space, the limited on chip cache is used ineffectively because unused cell memory cannot be distinguished from utilised cell memory. The effect of this can be seen in Table 3.4.

Table 3.4 shows the total amount of data transferred between the video memory and the GPUs L2 cache and the efficiency of the transfer, as a percentage of the theoretical maximum, for the 12 Å querying test with a grid cell size of $2S_r$, as reported by the NVIDIA GPU profiler. The efficiency column indicates how efficiently the memory bus was utilised. When the structures are smaller, the efficiency of the transfers is low because there isn't enough simultaneous data access transactions to maximise the

PDB Code (Atoms)	Fixed		Compact		Linked	
	Transfers (MB)	Efficiency (%)	Transfers (MB)	Efficiency (%)	Transfers (MB)	Efficiency (%)
1ANF (2987)	2.9	0.26	6.5	0.59	6.7	0.63
1OMP (5737)	8.4	0.47	3.8	0.22	8.2	0.48
5E0T (6043)	2.9	0.29	6.2	0.67	2.5	0.28
1AF6 (10520)	6.1	0.30	3.6	0.35	7.0	0.68
1XI4 (15317)	69.0	3.21	12.0	3.31	41.0	10.83
4A97 (25300)	74.0	2.40	13.0	0.94	32.9	1.89
3E76 (54478)	260.0	4.76	14.2	0.52	778.0	17.75
1AON (58891)	249.0	5.17	18.0	0.68	716.0	17.00
3JCU (59354)	275.0	4.01	23.0	0.80	792.0	18.60
1FFK (64296)	361.0	5.45	123.0	3.60	1550.0	23.00
1HTQ (90696)	478.0	4.93	65.0	1.30	1850.0	21.20

Table 3.4 Table shows the amount of data moved from the GPUs main memory to its cache, in MB, during the 12 Å querying test. Grids with a cell size of $2S_r$ (3.6 Å) were used. The efficiency columns show the percentage utilisation of the GPUs memory bus, according to the NVIDIA profiling tool.

data bus. When the structures are larger, more data is transferred, and the efficiency is slightly better, however still low. This is a result of the access patterns used during the proximity queering test; the random access patterns that are required to search the cells of the grid do not fully utilise the cards memory bus.

The table shows that although the linked grid transfers far more memory between the RAM and the cache, these transfers are performed more efficiently, utilising more of the available bandwidth than the fixed grid. Both the fixed and linked grids transfer considerably more data than the compact grid, demonstrating an area where the compact grid has a performance edge over the other structures.

3.6.3 Memory Consumption

Each of the different grid structures utilises video memory differently. Table 3.5 shows how much memory is allocated for each structure, for each of the test proteins when the grid cell size is $2S_r$. All of the structures have additional memory allocated to them, in order to allow an increase in the number of cells within the grid.

Two main points can be drawn from the data in Table 3.5: 1) The fixed grid uses the most memory, by a considerable margin and 2) The Linked and Compact grids memory consumption is more dependent on the distribution of atoms, rather than the number of atoms in the grid.

PDB Code (Number of Atoms)	**Fixed (MB)**	**Compact (MB)**	**Linked (MB)**
1ANF (2860)	150.1	0.8	0.4
1OMP (5737)	150.2	1.5	0.6
5E0T (5804)	150.2	1.8	0.8
1AF6 (10050)	300.4	2.8	1.2
1XI4 (15300)	300.6	53.3	26.4
4A97 (25020)	300.9	12.1	5.6
3E76 (53970)	301.9	15.3	6.7
1AON (58674)	302.1	19.6	8.8
3JCU (59354)	302.1	18.0	8.0
1FFK (64268)	302.3	22.1	9.9
1HTQ (90672)	303.3	26.8	11.8

Table 3.5 Table showing the amount of memory allocated for each grid type, when the grid cell size is $2S_r$ (3.6 Å), for each of the different proteins. Values are in megabytes.

The fixed grids memory consumption is a result of each cell within the structure being of equal size. As stated in Section 3.4.1 To ensure that no cell runs out of memory, the original allocated memory pool is large, leading to a high overall memory consumption. This approach, as opposed to allocating a small amount of memory and then increasing the size of the buffers if required, was taken because of the performance penalty of increasing the size of GPU memory buffers during execution.

The linked and compact grids also allocate all required memory before execution, with room for expansion. Owing to the nature of the structures, doing this is far less memory intensive than with the fixed grid. By utilising the fact that each atom will be in at most 8 cells, the grid buffer will never be longer than eight times the number of atoms within the structure. The only arrays that are of an indeterminate length are the arrays that contain the cell start pointers, which, at 4 bytes per cell, are comparatively cheap to over allocate. The linked grid uses less memory than the compact grid because of its single stage build algorithm; the compact grid's additional memory consumption is a result of having to calculate cell start points and then copy the atoms into the grid in two separate stages, requiring buffers to store the intermediate data.

Table 3.5 shows the memory consumption of each grid when a cell size of $2S_r$, the smallest tested cell size, is used. If a bigger cell size was used, there would fewer be cells in the grid, leading to a drop in memory consumption for all three grid types, as the arrays used to point to, and keep track of the contents of each cell, would reduce in size.

3.6.4 Discussion

This chapter set out to prove that regular grids could be constructed in less than 2 ms, with time left over for some computation. For most of the smaller structures: 1ANF, 5E0T, 1AF6 and 1XI4, the results show that, in all tested cases, this is achieved by each of the different grid algorithms, when the grid cell size is properly tuned. When considering the larger structures: 3E76, 1AON, 3JCU, 1FFK and 1HTQ, they only meet the time requirement when the neighbour search radius is small and the cell size is optimum. However, if we interpolate the forces on the haptic device, allowing a reduction in refresh rate to 500 Hz, or an update every 2 ms, only the largest three proteins fail to meet this goal target and only when using performing the nearest neighbour search with a 12 Å search radius.

The linked grid could be argued to be the best construction algorithm, because it is the fastest to construct and has the smallest memory footprint of all of the tested structures. However, the grid querying testing showed that there is a performance penalty when using the linked grid, when a large number of cells are searched.

The fixed grid construction time took slightly longer than the linked grid, and the overall performance was shown to be the slowest of the three. This was unexpected, as the fixed grid should be much faster in the querying test than the linked grid, and faster than the compact grid, owing to the O(1) access time of each of the cells. The slow down is a result of the pockets of empty space between the cells leading to sub-optimal memory access patterns during execution.

The compact grid took the longest to construct, but proved the fastest to query, by a large margin, in the case of the biggest proteins. If the grid is used multiple times per-rebuild, the compact grid will provide the best overall performance, despite being slower to construct. If the structure is small, marginally better performance may be achieved with the linked grid.

Tests demonstrated that setting an appropriate cell size had a larger effect on overall grid performance than changing the grid construction algorithm. It is therefore important that the grid cell size is tuned for its proposed application.

The focus of this chapter has been on reconstructing a spatial data structure from scratch. Alternatively, one could attempt to update an existing data structure with the new positions of its constituents. Reinhard et al.[116] demonstrated this was possible with regular grids, although the performance was not brilliant, and after a few alterations, real-time performance was lost.

It would be trivial to create a modifiable fixed grid, as there is free memory within each cell of the structure. One would simply have to identify those atoms which have moved, test them to see if they've moved out of their original cells, and if so determine which cell they've moved into, and add them to those cells, whilst removing them from any cells that they're no longer in. With the compact grid this process is more complicated as there is no spare memory between cells, however, if there were only a few changes to the structure during usage, it could be worth doing.

The problem that arises is, during molecular docking, the vast majority, if not all, of the atoms move between frames; if the cell size is small, many of these atoms will have moved into, and out of cells they were not touching previously. Therefore, attempting to modify the structure may well lead to spending time identifying those atoms which have moved out of a cell, or into another cell, before effectively reconstructing the data structure anyway.

3.7 Conclusion

To conclude, it has been shown that it is possible to create and use a regular grid in less than 2 ms. All of the grid algorithms presented complete their build stages in under

1 ms for all of the proteins tested, and when the cell size is tuned to the application correctly, all of the grid structures complete the build and the 1.5 Å and 6.0 Å query tests within the 2 ms/500 Hz time limit.

When the protein structure is large, the compact grid provides the best overall performance, owing to its more efficient memory layout. For the smaller proteins, there is very little time between the slowest and fastest structures; the linked grid may give the best performance, but it will be marginal. For this reason, the compact grid will be utilised within both the rendering and the molecular docking stages of this project.

In the next chapter, the compact grid is utilised within a molecular trajectory rendering system, to allow per-pixel shadows and ambient occlusion to be calculated in real time, on a deforming protein.

Chapter 4

Rendering Deforming Proteins with Advanced Lighting

4.1 Introduction

In an interactive molecular docking environment, the “haptic” render is three dimensional, as haptic devices allow exploration with the X, Y and Z axes. The “visual” render however, is a two dimensional perspective projection of 3D geometry, as the majority of Visual Display Units (VDUs) are limited to rendering in two dimensions (although head-mounted displays and 3D VDUs are becoming more common). The problem with viewing a protein in 2D is understanding the relative depth of different parts of the topography; surface features may not be obvious to the user. When the proteins involved in the docking scenario are static, this can be mitigated by viewing the structures from multiple angles, both before and during docking. Such an approach may not be practical whilst modelling flexibility, as during the docking process, the topography of the proteins changes, often in a short space of time.

Lighting effects, including ambient occlusion[95] and shadowing, can be used to enhance depth perception within a three dimensional scene. Ambient occlusion is a technique which calculates how exposed a point is to ambient lighting; more enclosed portions of the scene are rendered with less illumination than more open areas. Ambient occlusion has been shown to be an effective method of highlighting crevices within a protein's topography[46]. Shadows have been shown as an effective method of showing how an object moves, relative to another object, in 3D space[151]. Therefore, the combined effect of both should nicely enhance the three dimensionality of the biomolecule being rendered.

The problem is that these effects are computationally expensive to generate and render. A common approach to reducing the computational cost is to calculate the lighting on a per-vertex level shading (often known as Gouraud shading[43]), and then interpolate the results. However, the resulting effect can be of low quality with little detail, especially when the number of vertices is low, or the distance between them is large. The alternative to this is calculating the lighting of each pixel independently (known as Phong shading[113]), which, although more computationally expensive than per-vertex, ensures that the lighting is as detailed and consistent as possible. Computing per-pixel shading on a deforming scene, in real time, presents a significant challenge, owing to the amount of computation required per-frame.

Nevertheless, to ensure the accuracy and consistency of the lighting effects within the interactive molecular docking environment, per-pixel lighting effects will be used. In this chapter, a novel approach to adding shadows cast as a result of directional lighting and ambient occlusion to a deforming protein rendered in space-filling mode is presented. As a proof of concept, the algorithm is implemented in the software "Haptimol Protein Trajectory Viewer," before being included in Haptimol FlexiDock at the end of the next Chapter.

4.1.1 Contributions

In this chapter, a novel method of rendering a deforming biomolecule with per-pixel ambient occlusion and ray-cast shadows is presented. The discussed method does not perform any pre-computation, and can be used to render a freely deforming structure, as required for interactive molecular docking software. It is shown that real-time performance is maintained whilst rendering these effects on a deforming biomolecule up to 75 thousand atoms in size on consumer grade hardware; a first in the field of biomolecular rendering.

4.2 Background

Research into graphical rendering has produced a number of methods that can be used to add advanced lighting effects to a computer generated scene. Rendering can be performed before the video is required (pre-rendering) or in real time (real-time rendering). This chapter is concerned with real-time rendering, as the developed methods are intended for use in an interactive environment.

4.2.1 Shadows

Shadowing occurs when an object blocks, or partially blocks, illumination from a light source from falling on another surface. The light rays that cause shadows are generally considered to have been cast from a defined origin, and travel in a set direction. Ray tracing is the most elegant way to produce realistic shadows within a three dimensional scene, however it is computationally expensive. Alternative algorithms have been the subject of extensive research since the 1970s. The algorithms developed primarily fall into two broad categories: shadow volumes and shadow mapping.

Shadow mapping[153] has proven to be the dominant method used to generate real-time shadows in interactive applications, including the molecular graphics applications QuteMol[141] and RasMol[121]. Rendering with a shadow map is a two stage process. Firstly, the scene is rendered from the light's point of view, with the depth buffer (or depth map), saved to a texture. The scene is then re-rendered from the camera's viewpoint. After the scene has been projected into the correct coordinates, each pixel is tested against the depth map. If the z-value is greater than the stored value in the depth map, the object can be considered to be obscured from the light source, and therefore is in shadow.

The accuracy of shadows generated with shadow maps is limited to the resolution of the shadow map used, which is limited by the size of the texture. This inaccuracy can often be seen as aliasing at the edges of the shadows. Furthermore shadows generated with shadow maps have to be rendered once per light, which is time consuming, especially when a scene is illuminated by many lights. Often, when scenes are lit with many lights, the shadow mapping is limited to the main light source, for example the sun. Shadow mapping has been the subject of a large amount of research, with algorithms developed to tackle some of these limitations. Some approaches[88,133,154] increase the resolution of the shadow map when rendering parts of the scene near the camera, and reduce it for more distant objects. This reduces aliasing, but does not eliminate it. Another area of shadow map research also built on the idea that points at different distances from the camera need different shadow map densities. Developed by Zhang et al.[161] and later improved by Dimitrov[29], the cascaded shadow map method works by splitting the viewing frustum, allowing each shadow map to focus on a smaller area and therefore provide a better match between sampling frequencies in view space and texture space. These algorithms can produce an alias free shadow, however the shadow effect produced is hard.

A further development in shadow mapping, called ‘Variance Shadow Maps’, was presented by Donnelly and Lauritzen[30]. Variance Shadow Mapping aimed to allow shadow maps to be efficiently filtered, allowing them to take advantage of features included in modern graphics hardware, namely mip-mapping and anisotropic filtering. The method worked acceptably, but suffered from light bleeding. Further research has been undertaken to fix this issue by Lauritzen and McCool[79] and Annen et al.[3]. The results are real-time, all-frequency soft shadows, however the issues have not been completely eliminated.

Although the shading effects produced by shadow mapping can be of very high quality, aliasing often occurs, which detracts from the visual acuity of the scene. An alternative method to generating shadows, developed by Crow[27] is to create a shadow volume instead of a shadow map. The method calculates the geometry of the area occluded from a light source, defining these as shadowed areas. The basic process to calculate the shadow volume is to first find all the silhouette edges, the edges that separate front and back oriented faces, and extend these edges away from the light source toward infinity. The volume is then capped, at either the front or the back, depending on the implementation. If a point lies within this generated volume, then it is in shadow, otherwise it is illuminated.

Shadow volumes have proven to be less computationally expensive than pure ray-tracing and more visually pleasing than shadow mapping. This is largely because shadow volumes are per-pixel accurate, whereas shadow maps tend to suffer from aliasing. This method of shadow generation does have its drawbacks however, mainly that the additional geometry needed for the shadow volumes is costly to generate and then render. Also, there are issues when the shadow caster does not have a mesh that accurately represents an objects shape - for example, when billboards are used. Finally, shadow volumes are not natively compatible with soft shadows, which are often seen in the real world. Improvements have been made to the original algorithm including work by Heidmann[49] and Everitt and Kilgard[35] but these limitations are still present.

Although shadow volumes offer per-pixel accuracy, as desired for our molecular docking tool, the representation of the protein will be constructed using billboarding (Section 4.3.1), therefore, they cannot easily be incorporated in our application.

Real-time ray tracing has been included in a molecular renderer before, with excellent visual results, however the frame rate achieved was low[99]. BnsView, presented by Knoll et al.[72] is a CPU based renderer designed for running on multi-core CPU architectures, the visual results displayed are impressive, demonstrating interactive performance on a data set with 15 million atoms, but the hardware used during testing, a computer equipped with two 8-core Intel Xeons, a co-processor with 61 cores as well as an NVIDIA K20, cannot be considered consumer grade. Similarly, Stone et al.[135] presents an immersive molecular visualisation tool that utilises remote GPU clusters to present a high definition render of a trajectory to a head mounted display.

Ray casting can also be used to generate a shadowed effect. Demonstrated by Easdon[32] in his static protein viewer, shadows can be added to a render by casting a ray toward the light source from all of the pixels in the scene that contain geometry. As the ray traverses the scene, it performs intersection tests with other geometry. A spatial partitioning structure is used to eliminate distant geometry from testing. If the ray reaches the light source, it is illuminated, otherwise, it is in shadow. The limitation of the algorithm used by Easdon[32] is that it requires pre-computation: the shadow computation is heavily reliant on a regular grid, which is computed before rendering begins.

In Chapter 3, it was shown that it is possible to reconstruct a regular grid in less than 1 ms; fast enough for it to be suitable for use with a haptic device. Therefore, it follows that it is also suitable for use with a visual display, as the required frame rate is much lower. In this chapter, it is utilised in order to render ray-cast shadows on a deforming protein.

4.2.2 Ambient Occlusion

Ambient occlusion is a technique used for calculating how exposed each surface is to ambient light from the environment. For example, on an overcast day, the inside of a tunnel will be darker than the outside because it is more *occluded* from the diffuse light of the environment, than its exterior. Ambient occlusion attempts to simulate this darkening effect.

Ambient occlusion shading is a popular way of enhancing the depth of a three dimensional scene. The most straightforward and accurate way of calculating ambient occlusion is with ray tracing. Rays are cast from a point, and intersection tests are performed with other objects in the scene. After many rays have intersected the hemisphere of that point, a high quality approximation of the ambient occlusion is produced. The main limitation of this approach is the amount of computation required to produce a smooth model, although recent developments in GPU hardware are helping to overcome this[24].

Ambient occlusion can be calculated in object-space or screen-space. The first approach, object-space ambient occlusion, computes an occlusion value from the geometry within the scene. The later method, screen-space ambient occlusion, calculates ambient occlusion on a per-fragment basis.

An early technique to render ambient occlusion in object space was presented by Pharr and Green[112]. The algorithm described uses a preprocessing step to calculate how much of the external environment can be seen from each point on a model, then uses this information to compute the visible lighting of that point. The preprocessing step is too expensive to consider integrating into the display loop.

Sattler et al.[120] presented a hardware accelerated ambient occlusion algorithm that worked by constructing a visibility matrix. Although the authors highlight that the method can be applied to dynamic scenes, it is limited in that it is applied per vertex rather than per pixel. This results in artefacts caused by under sampling, which can be seen as aliasing at the shadow edges. A similar approach was used by Tarini et al.[141] within QuteMol: An offscreen rendering pass computes a shadow map, then this shadow map is used to calculate the light for each fragment within the scene. The performance of these approaches is dependent on the number of vertices within the scene, meaning performance will suffer with large, dynamic data sets.

An alternate idea, first presented by Bunnell[21] and later improved by Hoberock and Yuntao[51], works by approximating an accessibility value of each element in the scene. This is achieved by calculating the solid angle of an oriented disk which is then used to calculate the amount an object shadows another object. The algorithm presented utilises a hierarchical data-structure to reduce the resolution of the occlusion as the object being rendered becomes more distant. As noted by Hoberock and Yuntao[51], the presented method does have limitations, namely, discontinuities may be visible within the ambient occlusion and often the resulting occlusion is biased toward darkness.

Another method, published by Kontkanen and Laine[74], computes inter-object AO in real time. For each occluder, a grid surrounding the object is computed. The grid contains an approximation of the occlusion caused by that object. This information is then used within the fragment shader at run-time to determine an occlusion value on a receiving object. Although real-time performance is achieved, this method suffers from a long pre-computation time and has the potential to use a large amount of memory, especially if the number of objects within the scene is large.

Density volumes have also been used to achieve ambient occlusion in real time[46,108]. The methods work by storing occlusion information inside a volume. Grottel et al.[46] presented a volume based ambient occlusion algorithm for use when visualising particle-based data, molecular dynamics simulations for example. Their algorithm, implemented in MegaMol, uses a volume composed of cells to determine the intensity of light at different areas within the scene. During the rendering, the volume of each sphere resident within a voxel is added to that voxel's value. Tri-linear interpolation of the volume is then used to determine the "occlusion factor" at each fragment within the scene. This approach is computationally inexpensive, and can be used to approximate ambient occlusion in very large particle datasets, however it also has some fundamental limitations. Firstly, the quality of the ambient occlusion is highly dependent on the resolution of the volume. If the resolution is low, the calculated ambient occlusion is global: large pockets are highlighted, whilst smaller crevices are not. When the resolution is high, the inverse is true. Therefore, the resolution of the volume needs to be carefully tuned according to the dimensions of the molecule being rendered; this is done by the user within MegaMol. A further limitation of MegaMol's rendering ambient occlusion method is that when planes of spheres are not aligned with the volume, artefacts can occur in the occlusion effect. These limitations make the algorithm unattractive for use within an interactive docking system.

A further ambient occlusion algorithm, presented by Staib et al.[132], used cone tracing to create a realistic ambient occlusion effect, considering even distant geometry. Similarly to the algorithm presented Grottel et al.[46], the algorithm is dependent on a three-dimensional density texture, and suffers the same drawback: the lower the texture resolution, the more interpolation is required, reducing the accuracy of the lighting estimation. The authors note that artefacts occur when the rendered spheres overlap, owing to the use of point-based rendered spheres.

At a similar time Favera and Celes[36] presented another cone-tracing based algorithm. The algorithm they presented follows a similar process to the others described above: the scene is voxelised into a regular grid, then during rendering the hemisphere around each visible point is sampled by a set of cones. The algorithm approximates the volume of the cones using spheres. The amount of occlusion applied to a point is dependent on how obstructed each sphere within the cone is. An earlier cone-tracing algorithm was presented by Crassin et al.[26]. The earlier algorithm generates a low resolution sparse voxel octree rather than a regular density texture.

An analytical object-space ambient occlusion algorithm was presented by Skånberg et al.[129]. The algorithm described exploits the fact that all the geometry in the scene is spherical, and uses the solid angle formula to determine the fraction of any neighbouring spheres that are visible from a point. This is then used to compute the ambient occlusion contribution of the neighbouring sphere to the point in question. The technique used produces a realistic-looking ambient occlusion effect, however it is highlighted within the paper that there are limitations to this approach, mainly relating to the fact that a large search neighbourhood will result in poor performance, and a smaller one will result in a poor estimation of ambient occlusion.

Screen space ambient occlusion (SSAO) is generally less expensive than object space occlusion because the amount of work completed by the algorithm is dependent on the resolution of the render, rather than the number of objects in the scene. However, compared with object space occlusion, it suffers from some disadvantages, namely, the technique is view dependent, and is consequently unable to take into account geometry that is not rendered to the screen.

The first major development in screen space ambient occlusion was developed by Mittring[97]. The technique described calculates a per-fragment occlusion value using the depth buffer. Each pixel calculates its own occlusion value by randomly sampling different points around itself and computes the amount of occlusion applied to it based on differences in depth. The resulting approximation is reasonably successful in adding an ambient occlusion effect to a scene in real time, and has proven to be a popular choice for interactive applications. However, this early presentation of the technique suffered from incorrect shadowing caused by self-occlusion. This was resolved by Filion and McNaughton[37], who, instead of generating random sampling vectors in a sphere around the test point, generated vectors around a hemisphere which is centred around the normal of that point in the screen.

Horizon Based Ambient Occlusion (HBAO) was introduced by Bavoil et al.[12] and then improved by Bavoil and Sainz[11]. HBAO uses the depth buffer as a height map to find the horizon, defined as the average slope in the height field around a point. Heavily occluded points were shown to have a steep horizon angle. The resulting occlusion effect is shown to be higher quality than standard SSAO, but the effect is more expensive to compute.

Many other screen space ambient occlusion algorithms have been presented, including work by Shanmugam and Arikan[124] who presented an approximation to ambient occlusion that works in real time by separating near and distant ambient occlusion and rendering the near occlusion with a high level of detail, and Kajalin[64] who built on Mittering's work. Despite these improvements in SSAO, object space ambient occlusion produces a higher quality image because it considers the entire scene, as opposed to only considering what is visible through the view frustum.

Therefore, an object-space Ambient Occlusion algorithm will be investigated for rendering deforming proteins. Of the methods discussed, an adaptation of the analytical method used by Skånberg et al.[129] is likely to offer high quality ambient occlusion whilst maintaining a real time refresh rate. Although only suitable for local occlusion, the effect produced is of high quality and calculated per-pixel, therefore meeting the criteria set out in the introduction.

4.3 Methods

As stated in the introduction, the objective of this chapter is to develop an approach to rendering per-pixel lighting on deforming molecular structures in real-time. The shadowing portion of the lighting algorithm used in this chapter builds on the work done by Easdon[32], adapting it for use on a dynamic scene. The ambient occlusion algorithm used is similar to that presented by Skånberg et al.[129] although it is modified for effective use alongside the ray-cast shadows.

The end result is a novel approach to rendering shadows and ambient occlusion on a deforming scene, with methods efficient enough to be used to render large deforming molecular structures.

4.3.1 Sphere rendering

A common way of rendering a sphere is to generate a mesh of triangles, as an approximation of a sphere. When the number of triangles is high, the approximation can look reasonably accurate (Figure 4.1 (C)), however when rendering a protein comprising many atoms, the total number of triangles within the scene becomes large. This results in high memory usage and rendering time, and so is not a viable approach for the proposed application, in which keeping overall computation time to a minimum is important. Using fewer triangles reduces the computational footprint, at the expense of image quality (Figure 4.1 (B)), making it undesirable for use with per-pixel lighting.

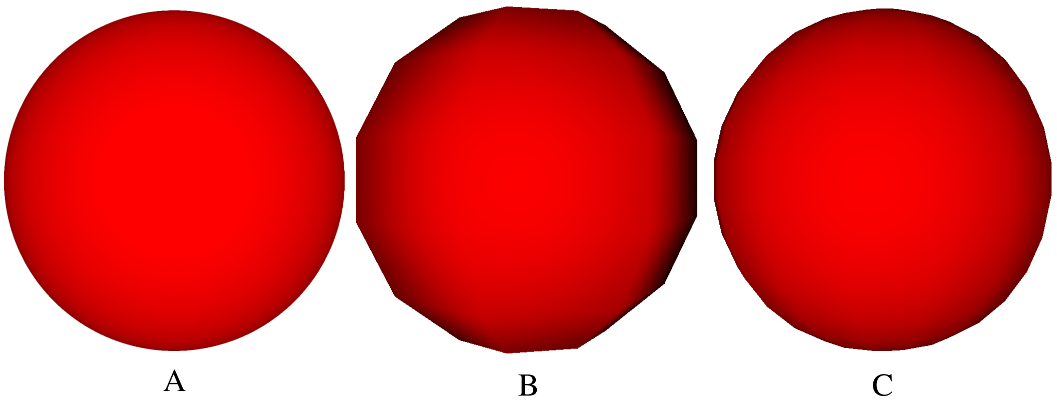


Fig. 4.1 An oxygen atom rendered using (A) ray-casting and a triangle mesh made up of (B) 84 and (C) 420 triangles

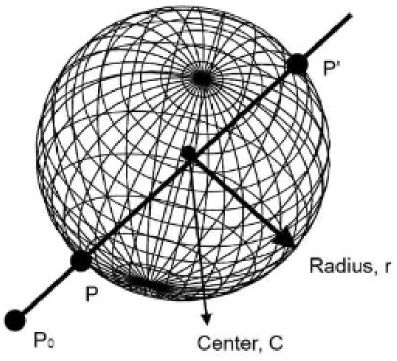


Fig. 4.2 A ray with position P_0 intersecting a sphere with centre C and radius r at points P and P'. (Easdon[32])

Smooth spheres can be created using ray casting. The algorithm, originally presented by Gumhold[47], then improved by Sigg et al.[127], works by generating a billboard equal in size to the bounding box of the desired sphere, and then culling all of the pixels from the billboard that fall outside of the sphere. In order to determine whether or not a pixel is within the sphere, whilst ensuring the sphere remains perspectively correct, ray-casting is used (Figure 4.2).

When the billboard has been positioned in screen space, a ray is cast from the camera through each pixel, or fragment, within it. This is achieved by substituting the parametric equation of a ray: $\mathbf{P} = \mathbf{P}_0 + t\hat{\mathbf{v}}$, into the equation for a sphere: $|\mathbf{P} - \mathbf{C}|^2 - r^2 = 0$, which results in $|(\mathbf{P}_0 + t\hat{\mathbf{v}}) - \mathbf{C}|^2 - r^2$. This is equivalent to the quadratic equation:

$$\hat{\mathbf{v}}^2 t^2 + 2\hat{\mathbf{v}} \cdot (\mathbf{P}_0 - \mathbf{C})t + |\mathbf{P}_0 - \mathbf{C}|^2 - r^2 = 0, \tag{4.1}$$

which is solved for each pixel using the quadratic formula. If, for any specific pixel, there is no real solution to the equation, the pixel is discarded, as it lies outside of the sphere.

The resultant sphere, pictured in Figure 4.1 (A), is not only accurate to the pixel, but has also been shown to offer better performance than the conventional triangle mesh comprising many triangles[32]. A similar approach to generating spheres is used in the software MegaMol[45] and VMD[54]. This approach to generating spheres will be used in the rendering algorithms presented in this chapter.

4.3.2 Shadow casting

Given a rendered scene, shaded areas can be determined by casting a ray from each fragment toward the light source. If the ray intersects with other geometry in the scene, the pixel it was cast from is in shadow (Figure 4.3). Although ray-casting produces shadows that are accurate to the pixel, it is a computationally intense approach to use, as each ray in the scene has to perform intersection tests with all of the geometry pictured in the render.

In order to reduce the number of intersection tests performed by each ray, a spatial partitioning structure can be used to quickly eliminate geometry that is well remote of the path of each ray. However, use of this acceleration structure ordinarily limits the use of ray casting to static scenes. In Chapter 3, GPU based regular grid construction algorithms were demonstrated to be fast enough for use with a haptic device. Therefore, it follows that they are also suitable for use with visual rendering, as the refresh rate requirements for smooth video are much lower than those demanded by a haptic device.

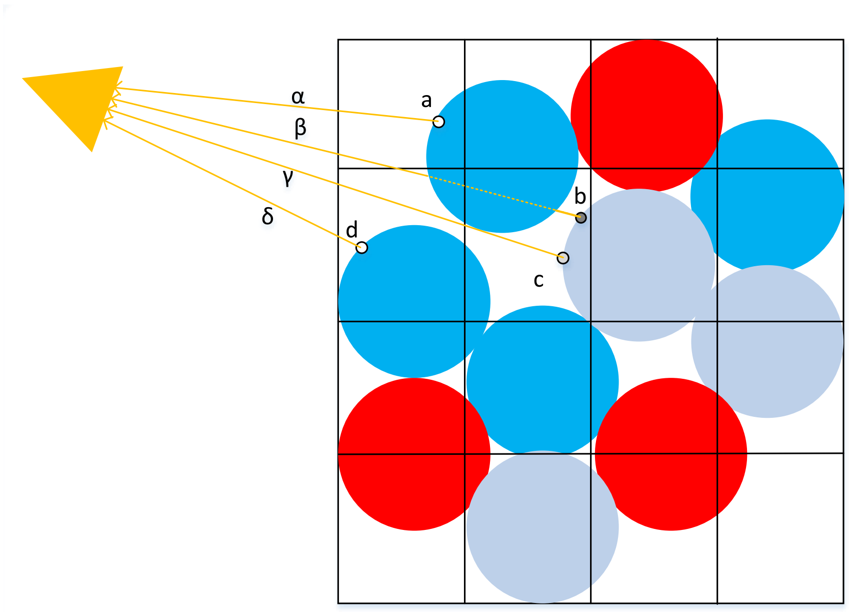


Fig. 4.3 A two dimensional model of a protein inside a grid, together with examples of light-path detection rays used to determine the illumination for four points on the model. Rays α and δ are uninterrupted, so the pixel will be fully illuminated. Ray β shows a completely interrupted ray so pixel b will be completely in shadow. Ray γ shows a partially obscured ray. This results in a slightly darkened, but not completely shaded pixel, resulting in a soft shadow effect.

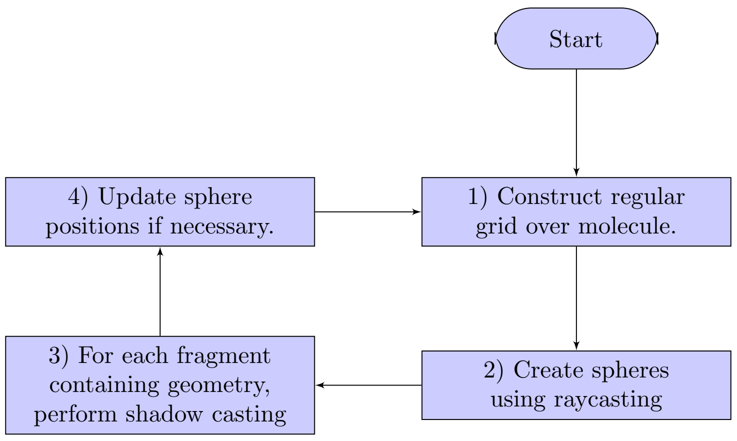

Start
1) Construct regular grid over molecule.
2) Create spheres using raycasting
3) For each fragment containing geometry, perform shadow casting
4) Update sphere positions if necessary.

Fig. 4.4 Flowchart describing the algorithm used to render ray-cast shadows on a deforming protein.

Figure 4.4 shows the algorithm used in order to generate the shadow effect. To achieve a high frame rate, it is important that each of the steps is completed in the minimal amount of time. Therefore, wherever possible, the computation will be performed on the GPU, thus avoiding data transfer backward and forward between the GPU and CPU. Memory buffers can be shared between CUDA and OpenGL, so transferring the regular grid and updated atom positions between the two is straightforward, however, there is a performance penalty when doing so. The shadow calculation portion of the algorithm, Step 3 in Figure 4.4, is described in Algorithm 8.

Algorithm 8 Shadow calculation algorithm

```
 1: for all pixels containing geometry in parallel
 2:     $R \leftarrow N \cdot L$
 3:     SphereID←AtomID pixel is part of
 4:     if $R > 0$ then
 5:         if SphereID$>$ 0 then
 6:             Calculate grid index of ray origin
 7:             while inside grid do
 8:                 for spheres in grid cell do
 9:                     if ray intersects sphere then
10:                         Pixel is in shadow
11:                         break
12:                     end if
13:                     Advance to next cell
14:                 end for
15:             end while
16:         end if
17:     end if
18: end for
```

Algorithm 8 shows the method each fragment in the scene uses to determine whether or not it is in shadow. Again, it is crucial that each step completes in the smallest amount of time possible.

As shown in Chapter 3, the cell size of the regular grid can have a large impact on the performance of the algorithm using it. Therefore, a balance needs to be found between the size of each cell and the number of cells that need to be traversed by each ray. During ray-casting, it is likely that the smallest cell size will offer the highest performance. This is investigated in the results section of this chapter.

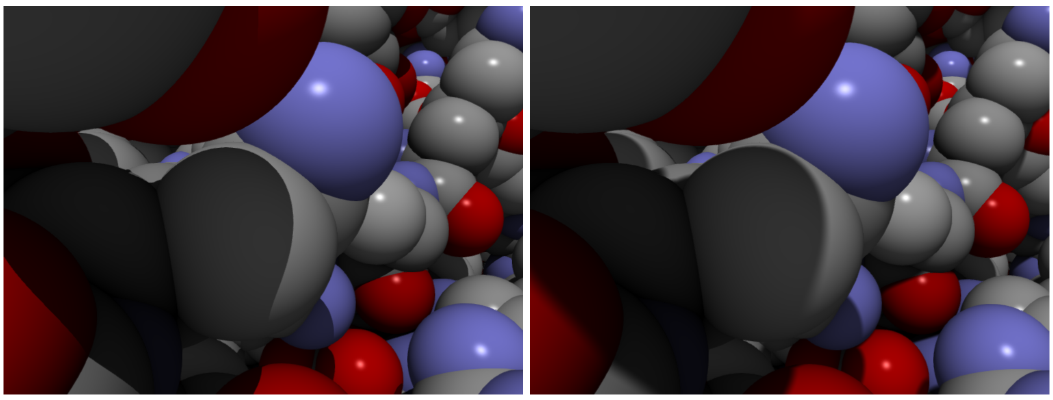


Fig. 4.5 Rendering with (A) hard and (B) soft shadows

When used with a point light source, the described ray casting technique produces hard shadows (Figure 4.5 (A)). In the real world, the boundary between the shaded and illuminated portions of the scene is less well defined (Figure 4.5 (B)). To achieve this with ray casting, an area light source could be used. However, if the size of the light source is increased, more than a single ray per-pixel will be required to generate the desired effect, which will increase the amount work required per-frame, resulting in a reduced frame rate.

A straightforward method to approximate the soft shadow effect whilst using a point light source was presented by Parker et al.[109]. The algorithm calculates approximate soft shadows from a point light source by using an enlarged sphere. Figure 4.6 depicts how the method by Parker et al.[109] works.

Each atom which potentially shades the fragment being tested is enlarged. If the fragments' ray collides with the original atom, the pixel is in full shadow. If the ray collides with the enlarged atom, it is partially in shadow. The level of shadow applied to the pixel is dependent on how close the ray is to intersecting the original sphere. The closer the ray passes to the sphere, the darker the pixel is. Any subsequent spheres that the ray intersects will contribute to the shadow intensity of the pixel. The resulting soft shadow effect can be seen in Figure 4.6 (B). Approximating shadows in this way is unlikely to significantly extend the computation time of the lighting.

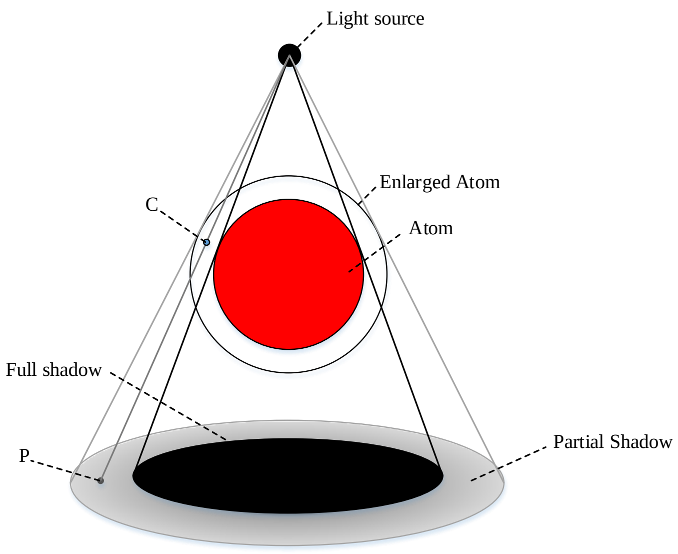


Fig. 4.6 A diagram demonstrating how an enlarged sphere is placed over an atom to create a soft shadow effect. The shadow intensity at point P is half of the full shadow intensity, because its ray passes the atom halfway between the atom and enlarged atom.

The algorithm described in Figure 4.4 describes an approach to producing quality, per-pixel shadows in real time. Provided the algorithm is fast enough, it could be used in an interactive molecular docking application to help improve the perception of depth of the rendered protein. The capabilities of the algorithm are assessed in Section 4.6.1 of this chapter.

4.3.3 Ambient Occlusion

The protein representation used in this thesis comprises spheres. This can be exploited in order to calculate how occluded any individual fragment of the scene is. An approach similar to the one described here is used by Skånberg et al.[129] in their molecular interaction visualisation tool. Figure 4.7 pictorially describes the algorithm used to determine how shaded any individual fragment is. The algorithm works by computing the solid angle (ω) of the visible proportion of the neighbouring spheres. The solid angle (Ω) of sphere B from point f is $\Omega = 2\pi(1 - cos(\theta))$, however only the fraction of the solid angle visible above the horizon is required. Therefore, t is

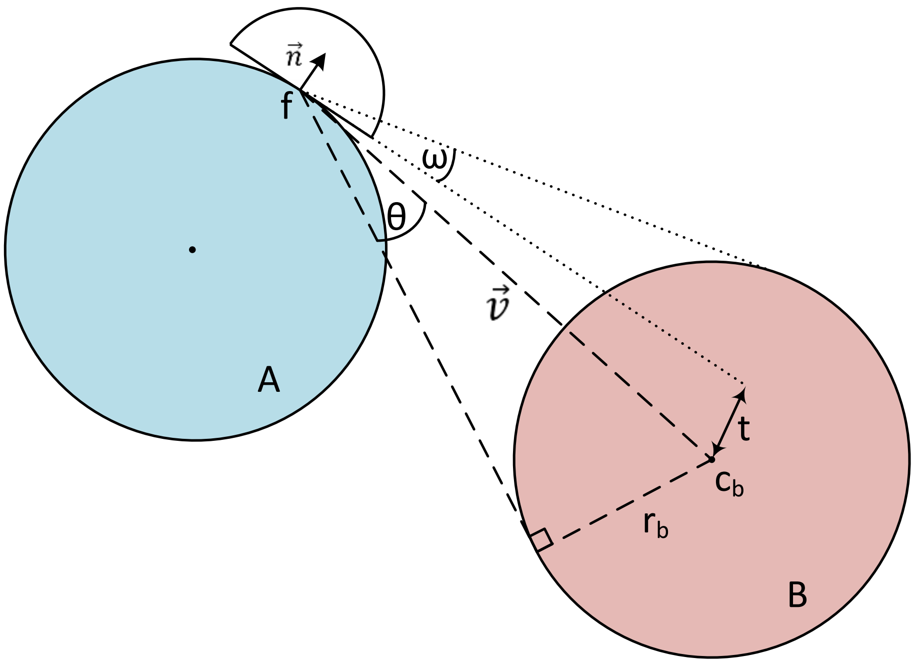


Fig. 4.7 Diagram showing how the AO contribution to point f on sphere A, by sphere B is calculated.

calculated as $t = v \cdot n$ where n is the unit normal of the fragment f, and v is the distance between C_b and f. The equation $p = t + r_b$ is then used to give the proportion p of sphere B that is above the horizon of fragment f. To calculate the AO contribution of that fragment, p is clamped to $[0, 2r_b]$ and then normalised to give p_{norm}. Therefore, the amount of atmosphere that is not occluded is given by $AO = 1 - p_{norm} \cdot (1 - cos(\theta))$. The final intensity of the lighting at fragment f is calculated by summing the ambient occlusion (AO) contribution of all of the atoms local to the fragment being tested, clamping it to between 1 and 0, and taking that value from 1[129].

The resulting illumination is a good approximation of local ambient occlusion that has a smooth fall off. However, it is an approximation, as it does not take into account the fact that the solid angle does not change in a linear fashion as the occluder grazes the horizon[129]. Furthermore, in closely packed portions of the scene, it is possible that atoms that provide no occlusion to a fragment are included in the illumination (Figure 4.8), as no visibility check is performed. This can lead to over occlusion within the scene, detracting from the quality of the render.

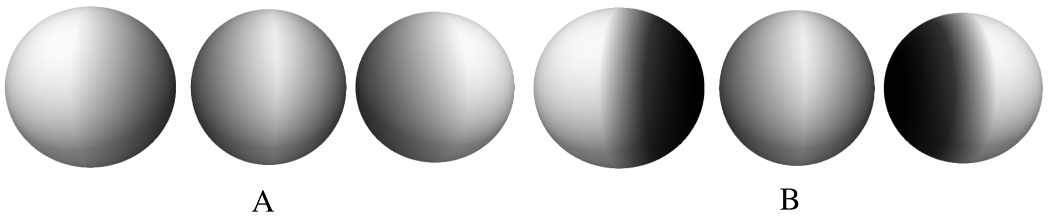


Fig. 4.8 Demonstrating over-occlusion within the analytical ambient occlusion method. (A) shows the ambient occlusion calculated when a small cut-off is used. The occlusion between each of the spheres is roughly the same. In (B), an area of influence large enough to incorporate the left hand sphere within the occlusion calculation for the right hand one, and vice versa. This results in far more extreme occlusion on the respective spheres, than on the central one.

The algorithm as presented by Skånberg et al.[129] utilises a regular grid, with a cell size set to the desired area of influence, the area around each sphere within which neighbours are considered within the occlusion calculation, in order to determine local occluders. The result of this is that each fragment performs a neighbourhood search to determine potential occluders, which is expensive. Also, should a larger area of influence distance be desired, the grid will comprise very large grid cells, reducing the effectiveness of the grid structure.

In the approach used within PTV, to allow the grid created for the shadow-casting algorithm to also be used for the ambient occlusion, a list containing all of the atoms within the set area of influence is generated for each of the atoms in the protein. These lists are generated using CUDA.

By performing proximity querying once per cell, rather than once per fragment, a larger cut-off distance can be used, potentially offering a more global AO effect. In order to counter the over occlusion effect that occurs when using a large area of influence, an AO scaling factor is used. The scaling factor can be used to reduce the effect each occluder has on a fragment.

4.4 Implementation

The described rendering algorithms are implemented using a deferred rendering approach, written in OpenGL 4.3. The rendering pipeline can be seen in Figure 4.9. Up until Step 5, a standard GLSL rendering pipeline is followed. Step 5 uses a ray-tracing approach to compute the points of intersection between the camera and billboard to determine if the point of intersection is on the sphere. These points of intersection are then stored in an off-screen buffer, and are used later in Steps 6 and 7 to render the lighting.

```
//calculate Ambient Occlusion
float calcAO(vec3 pos, float sphereID, vec3 norm)
{
  //if sphere does not exsist, return default value.
  if (sphereID < -0.5)
    return 0.7f;

  //determine lookup index of occluder list
  int id = int(sphereID) * maxOccluders;
  float ao = 0.0;

  for (int i = 0; i < maxOccluders; i++)
  {
    //load occluder sphere ID.
    int occluderSphereID = int(aoDATA[id + i]);

    //if ID < 1, all occluders tested, break loop.
    if (occluderSphereID < -0.5){
      if (i == 0){ //If no occluders in list, return constant
  .
        return 0.7f;
      }
      break;
    }

    //load occluding sphere from SSBO.
    vec4 sphere;
    int lookupId = int(occluderSphereID) * 4;
    sphere.x = spherePositions_ssbo[lookupId];
    sphere.y = spherePositions_ssbo[lookupId + 1];
    sphere.z = spherePositions_ssbo[lookupId + 2];
    sphere.w = spherePositions_ssbo[lookupId + 3];

    //position sphere in space.
    vec4 sphereEye = mvMatrix * vec4(sphere.xyz, 1.0);

    //calculate vector between occluding and original sphere.
    vec3 dir = pos - sphereEye.xyz;
    float len = length(dir);
    float sphereLen = sphere.w / len;
    float sqrtV = 1.0 - (sphereLen * sphereLen);
    sqrtV = (sqrtV < 0) ? 0 : (sqrtV > 1) ? 1 : sqrtV;

    //Determine occlusion contribution (Section 4.3.3)
    float t = dot(norm, dir);
    float AOProportion = (clamp((t + sphere.w), 0.0, 2 *
   sphere.w)) / (2 * sphere.w);
    ao += 1.0 - (AOProportion * sqrt(sqrtV));
  }
  //calculate final occlusion
  return clamp(1.0 - (ao * aoIntensity), 0.0, 1.0);
}
```

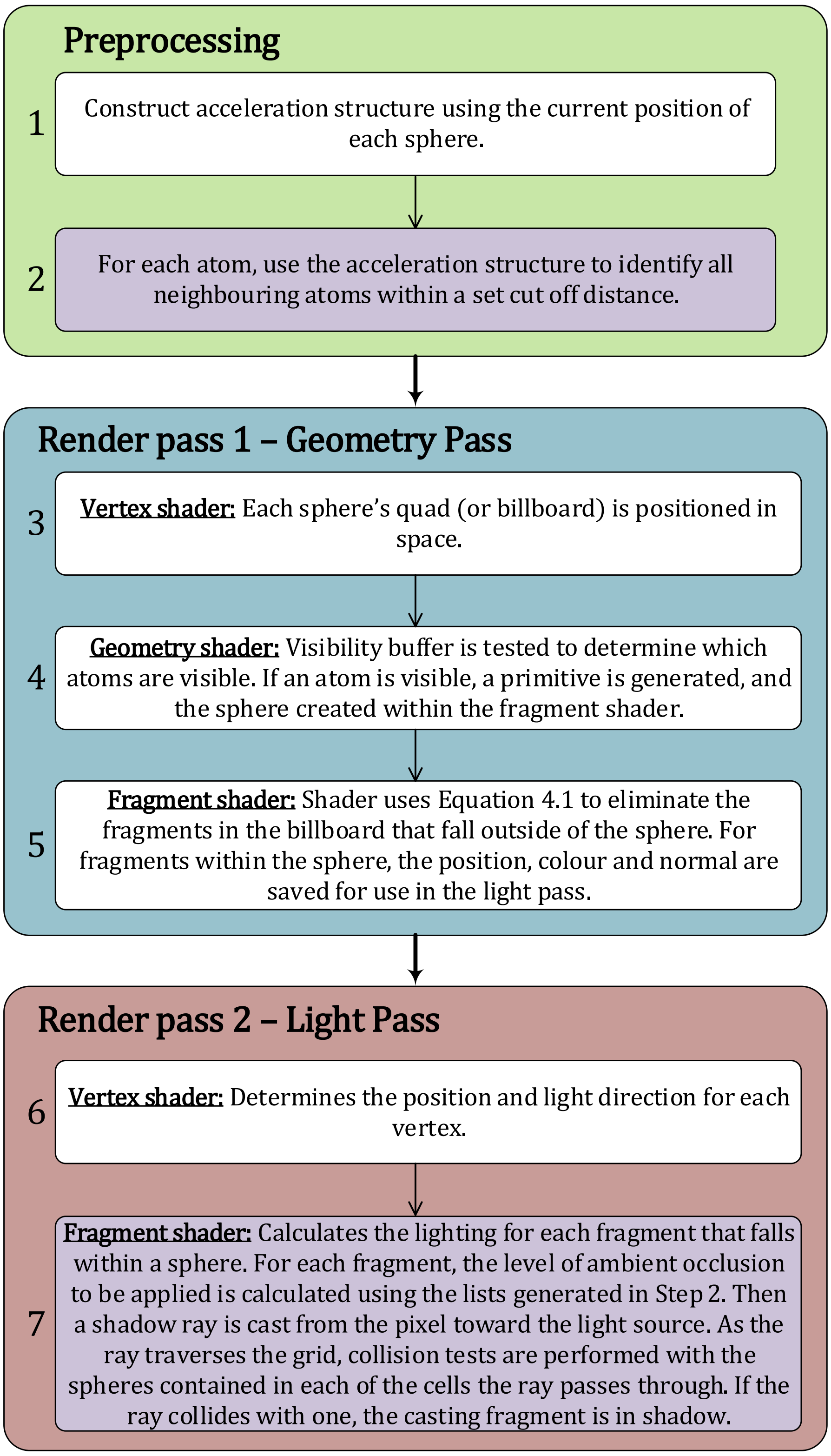

Preprocessing
1
Construct acceleration structure using the current position of each sphere.
2
For each atom, use the acceleration structure to identify all neighbouring atoms within a set cut off distance.
Render pass 1 – Geometry Pass
3
Vertex shader: Each sphere's quad (or billboard) is positioned in space.
4
Geometry shader: Visibility buffer is tested to determine which atoms are visible. If an atom is visible, a primitive is generated, and the sphere created within the fragment shader.
5
Fragment shader: Shader uses Equation 4.1 to eliminate the fragments in the billboard that fall outside of the sphere. For fragments within the sphere, the position, colour and normal are saved for use in the light pass.
Render pass 2 – Light Pass
6
Vertex shader: Determines the position and light direction for each vertex.
7
Fragment shader: Calculates the lighting for each fragment that falls within a sphere. For each fragment, the level of ambient occlusion to be applied is calculated using the lists generated in Step 2. Then a shadow ray is cast from the pixel toward the light source. As the ray traverses the grid, collision tests are performed with the spheres contained in each of the cells the ray passes through. If the ray collides with one, the casting fragment is in shadow.

Fig. 4.9 Diagram shows the rendering pipeline as used by Haptimol Protein Trajectory Viewer. A purple background indicates the regular grid is used at that stage of the rendering algorithm.

Listing 4.1 Method used to determine ambient occlusion of each fragment in scene.

Step 2, the generation of the occluder lists uses a locality search identical to the one tested in Chapter 3 to find all of the neighbouring atoms within the nearest n. Rather than simply count these neighbours, as the search does in Chapter 3, the ID of each is saved to memory for use within Step 7.

The lighting for each fragment is calculated in Step 7 of Figure 4.9. The ambient occlusion factor is calculated first, then the shadow calculation is performed. Listing 4.1 shows the method used to determine the amount of darkening applied to each visible fragment. Occluding atoms for each atom are arranged in the global array `aoData`, which can hold a set number of occluders for each atom. This list is iterated over, with the ao contribution of each sphere added together into the variable `ao`. `ao` is then multiplied by the customisable intensity multiplier `aoIntentsity` and taken from 1.0. The result is then clamped between 0 and 1, with 1 equal to no occlusion, and 0 equal to full occlusion. As each sphere will have a different number of occluders, an early break clause is specified on line 18. At the end of the list of occluding sphereIDs, '-1' is inserted, indicating the end of the list. The array `aoData` is large enough in size that no atom will ever run out of space whilst interactive performance is maintained.

The shadow casting algorithm is also performed on every visible fragment of the scene. Listing 4.2 shows the collision testing process performed for each cell the cast ray traverses. Lines 6 - 12 are for determining the start and end point of the specified cell. The final cell in the grid needs a different method to determine this than the rest of the grid, as it has no subsequent `cellStartPoint` to use. Therefore, the uniform `totalAtomCount`, which contains the total number of items in the array `cellId_ssbo`, is used to determine the length of the final cell.

The `for` loop on line 14 then iterates over the cell, testing each sphere within it. Within the for loop, lines 33-49 describe performing the ray-sphere intersection

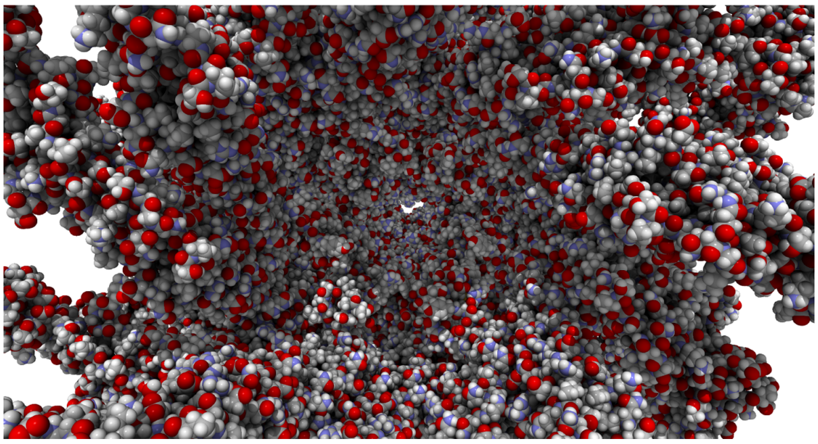


Fig. 4.10 Screen capture of FLAG taken during performance testing.

with the loaded sphere. Here also, the soft cell effect described in Figure 4.6 is produced, if the ray grazes the sphere's surface. Listings 4.1 and 4.2 describe the most computationally expensive parts of the light pass fragment shader (Step 7 of Figure 4.9). The complete shader is shown in Appendix B. In the subsequent sections, the performance of these methods is tested, and the quality of the effect they produce analysed.

```
//calculates if the ray intersects with any spheres located in the grid cell specified by gridCellIndexID
void softShadowGrid(vec3 ro, vec3 rd, float sphereID, int gridCellIndex1D, inout float s)
{


  int accessId, loopLimit, readId;
  accessId = cellStartPoint[gridCellIndex1D];
  if (gridCellIndex1D + 1 == finalCell){
    loopLimit = totalAtomCount - accessId;
  } else {
    loopLimit = cellStartPoint[gridCellIndex1D + 1] - accessId;
  }

  for (int i = 0; i < loopLimit; i++)
  {
    float shadowSphereID = cellId_ssbo[accessId];
    accessId++;

    //cellId_ssbo includes values of -1 when the index list ends
    if(shadowSphereID < -0.5)
      break; //break the for loop if no more atoms in grid cell

    int lookupId = int(shadowSphereID) * 4;

    vec4 sphere;
    sphere.x = spherePositions_ssbo[lookupId];
    sphere.y = spherePositions_ssbo[lookupId + 1];
    sphere.z = spherePositions_ssbo[lookupId + 2];
    sphere.w = spherePositions_ssbo[lookupId + 3];

    vec4 sphereEye = mvMatrix * vec4(sphere.xyz, 1.0);

    float t0 = dot((sphereEye.xyz - ro), rd);
    float b = 0.020;
    float R = sphere.w - b;
    float d = length((t0 * rd) - sphereEye.xyz + ro);

    if(t0 < 0.0){ //no collision, continue.
      s = min(s, 1.0);
    } else {
      if(d < R){ //collision, break loop
        s = 0.0;
        break;
      } else if(d > sphere.w){ //no collision, continue
        s = min(s, 1.0);
      } else { //close, simulate soft shadow.
        s = min(s, smoothstep(0.0, 1.0, (d - R) / b));
      }
    }
  }
}
```

Listing 4.2 Testing the contents of each cell for a collision.

4.5 Method Analysis

There are two main aspects of the render algorithms described above that need to be assessed: how well they perform in a real-time rendering environment, and the quality of the final effects. To assess the performance of the methods, and determine whether they are fast enough for use in an interactive molecular docking environment, a variety of biomolecules will be rendered, and the achieved frame rate will be recorded. A frame rate of 24 frames per second (FPS) or high, constitutes real time rendering.

To determine whether these algorithms achieve this refresh rate, each biomolecule will be rendered with diffuse lighting, ambient occlusion, shadows and then all three together. During the rendering tests, the biomolecules will be positioned so that they fill the screen, to ensure the majority of pixels are filled (See Figure 4.10), and rotated on the spot. Each test will involve rendering the biomolecule as it rotates upon the spot, for at least 5 minutes. These tests will be repeated at least twice. The compact grid (Chapter 3, Section 3.4.2) was used as the spatial partitioning structure in these tests, because of its low memory consumption and fast query time.

The most computationally expensive part of the shadow casting algorithm will be performing the shadow-casting on a per-fragment level. Therefore, it makes sense to tune the regular grid cell-size to that stage of the algorithm. Consequently, 5 different grid-cell sizes will be tested, in order to determine which cell size offers the best performance. As in Chapter 3, to ensure the majority of atoms are placed in at most eight cells, the formula nS_r was used to calculate each cell size, S_r is equal to the radius of a sulphur atom (1.8 Å), and n a multiplier, starting at 2. Furthermore, the frame rate will be measured whilst reconstructing the regular grid and whilst rendering a static biomolecule.

The performance of the described analytical AO algorithm is dependent on the area of influence: the area within which neighbouring atoms are considered to contribute to lighting of the pixel. A smaller cut-off distance will result in better performance but will produce a more local approximation of AO, when compared to a larger cut-off

PDB Code	Number of Atoms
1CRN	327
1OMP	5737
5E0T	6,040
3E76	54,464
3JCU	75,994
1HTQ	97,872
FLAG	316,404

Table 4.1 The number of atoms that make up each of the proteins tested.

distance (See Section 4.3.3). Therefore, testing will be performed using three different cut-off distances: 3Å, 6Å and 10Å. Again, the test will be performed whilst rendering both a static and a dynamic biomolecule.

In addition to testing the performance of the algorithms, the visual quality of each method needs to be assessed. To do this, renders generated using the described techniques will be compared to other molecular rendering applications which support similar effects.

4.6 Results

Table 4.1 contains information relating to the biomolecules used in testing. All of the structures except for FLAG were sourced from the protein data bank[14]. FLAG was provided by another research group. All of the tests were run on a desktop computer equipped with an Intel i7 processor, 16GB of RAM and an Nvidia GTX 980 graphics card. The rendering tests were performed at resolution of 1080p.

4.6.1 Method Performance

The benchmark test was rendering the biomolecule with diffuse lighting (See Figure 4.11). The results can be seen in Figure 4.12. The results show that without any advanced lighting effects, real-time rendering is easily achieved for all of the test proteins.

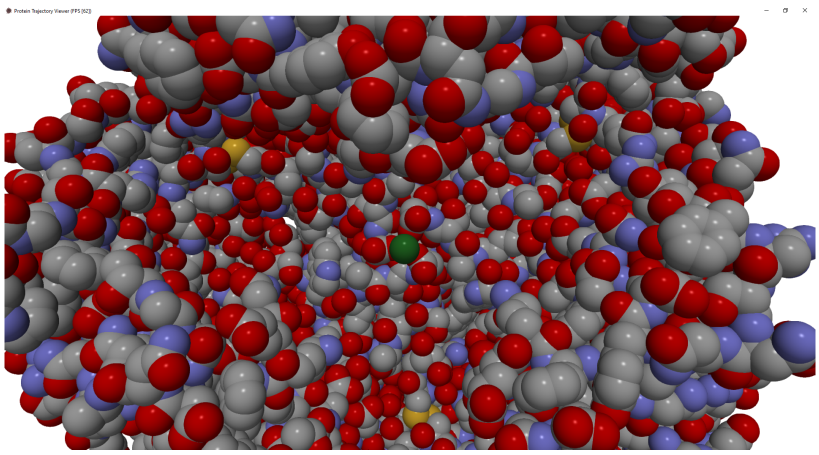


Fig. 4.11 Biomolecule rendered without any lighting effects.

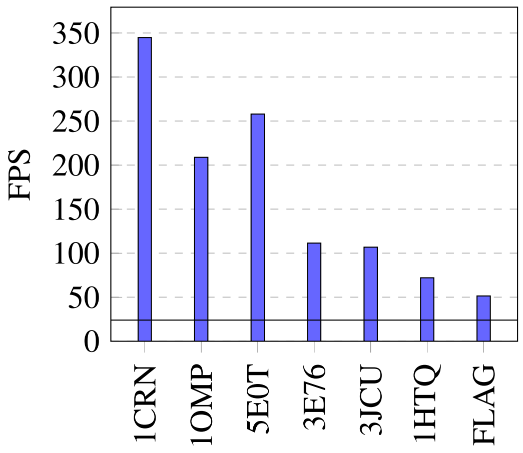


Fig. 4.12 Graph showing the average frame achieved whilst rendering each of the test proteins with no shadows or ambient occlusion.

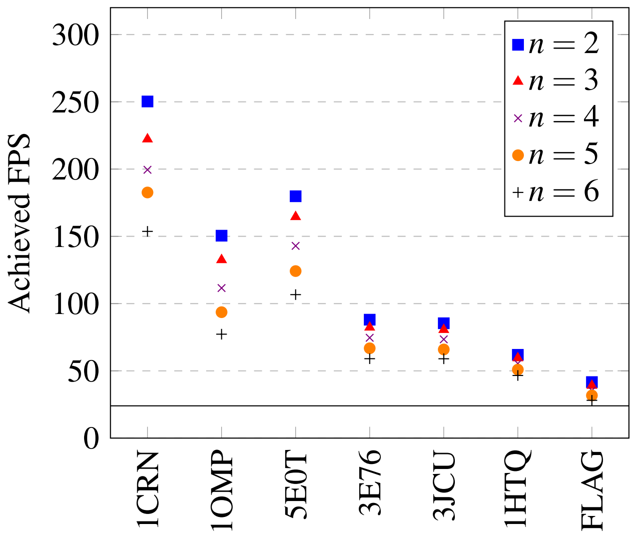


Fig. 4.13 A graph showing the achieved frame rate when rendering seven static proteins with ray cast shadows. Five different values of n were tested.

Shadows

Figure 4.13 shows the frame rate achieved when rendering each of the tested biomolecules with diffuse lighting and shadows, whilst using an acceleration structure comprised 5 different grid cell sizes.

Figure 4.13 shows that although real-time performance is achieved for all of the tested biomolecules and grid-sizes, the performance penalty for using a large grid cell-size is massive; in the case of 1OMP, using a grid cell size of $6S_r$ instead of $2S_r$ halves the frame rate. This makes sense, because smaller cells will, on average, contain fewer atoms than larger ones, reducing the number of intersection tests performed by each ray. Therefore, a cell size of $2S_r$ will be used for the subsequent tests.

A test was also run to determine the performance penalty of using soft shadows rather than hard shadows. The test found that over three tests, whilst rendering FLAG, the flagella filament, the frame rate achieved whilst rendering soft shadows was 0.98 FPS lower on average than when rendering hard shadows. As this penalty is small, soft shadows will be used in all the subsequent tests.

The results in Figure 4.13 show the average frame rate achieved without reconstructing the regular grid used to accelerate the shadow casting. Figure 4.14 shows the

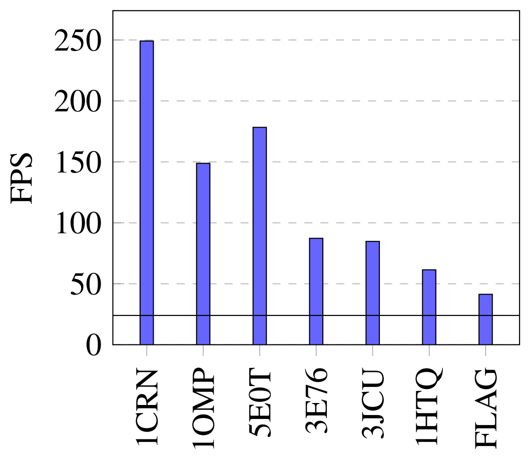


Fig. 4.14 Graph showing the frame rate achieved whilst rendering ray cast shadows and reconstructing the regular grid, as would be required when rendering a deforming protein. A grid cell size of $n = 2$ was used.

frame rates achieved whilst performing all of the additional work needed to render a deforming protein, primarily reconstructing the regular grid every frame.

The results in Figure 4.14 show that real-time performance is achieved for all of our test proteins, whilst rendering shadows and reconstructing the regular grid every frame. This demonstrates that the proposed shadow algorithm is fast enough for use in dynamic scenes, therefore, it could be used in an interactive molecular docking application. The increase in per-frame rendering time is negligible, because the grid construction algorithm consumes so little time compared to the shadow casting portion of the algorithm.

Ambient Occlusion

Figure 4.15 shows the achieved frame rate when rendering each of the test proteins with ambient occlusion, whilst using four different cut-off distances. In this first test, the data structures required to produce the ambient lighting effect were not refreshed, in order to determine the effect a larger area of influence has on the frame rate.

The results in Figure 4.15 show that although, for all tests performed, an interactive frame rate was maintained, increasing the cut off has a drastic effect on the frame

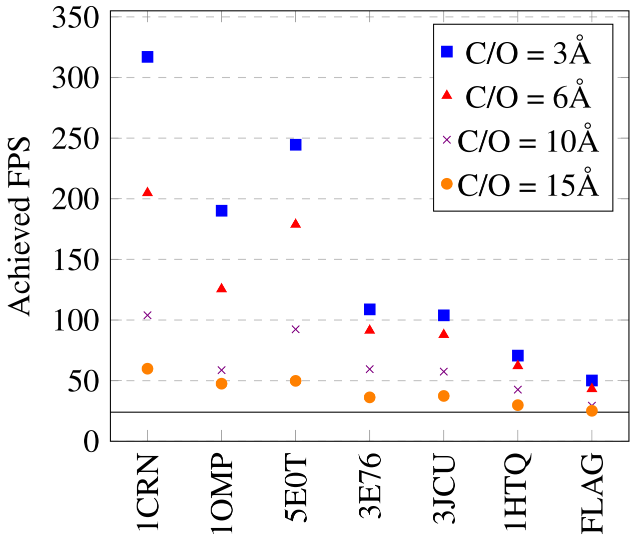


Fig. 4.15 Graph showing the achieved frame rate when rendering seven static proteins with analytical ambient occlusion. C/O, the cut off distance, equates to the search radius of potential occluders.

rate. This effect is likely to be exacerbated when reconstructing the data-structures per frame, as required by a dynamic scene.

Figure 4.16 shows the frame rate achieved whilst rendering ambient occlusion and reconstructing both the regular grid and occluder lists, as required by a dynamic scene. When the cut-off distance is very small, 3Å, interactivity is maintained for all of the tested structures, however, when using a larger cut-off, the frame rate achieved when rendering the larger structures drops below the required 24 FPS threshold. Interactivity is maintained for all of the structures apart from the very largest, FLAG, when using a cut-off of 6Å, which is enough to produce a good approximation of local ambient occlusion (See Section 4.6.2).

The results suggest that the AO approximation algorithm is fast enough to use whilst rendering a deforming protein, provided it has fewer than 100,000 atoms. This is encouraging, as there are a huge number of structures that fall into that category, and, as GPUs become more powerful, that atom limit will rise.

A pertinent question is "What effect would changing the cell size have on the performance of the ambient occlusion algorithm?" Whilst rendering the static structure,

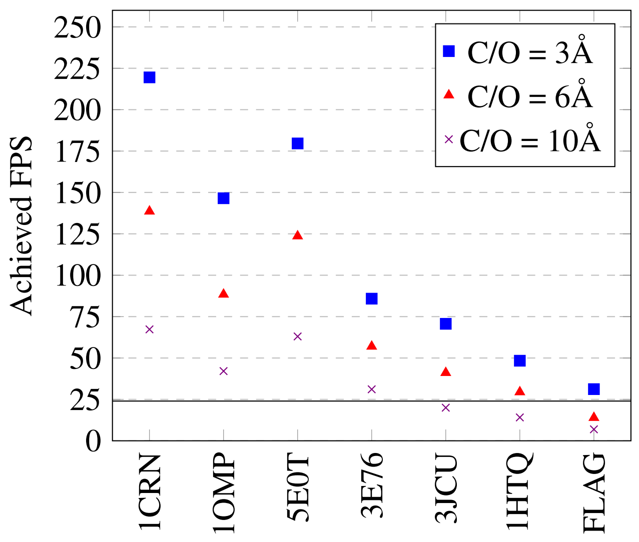


Fig. 4.16 Graph showing the average frame rate achieved when rendering five test proteins whilst rebuilding the regular grid and occluder lists in order to simulate rendering a deforming scene. C/O equates to the area of influence used.

changing the grid cell size would have no effect on performance at all, as the list of local occluders is only constructed once, before rendering begins. However, when rendering a deforming biomolecule, it would have an effect. As the process of building the occluder lists uses the regular grid in the same way the locality search performed in Chapter 3 does, the locality search test results can be used to give some idea of the affect changing the grid cell size would have. In Chapter 3, it was shown that setting the grid cell size to be just larger than the area being searched yielded the best performance, thus, we can assume that the same would be true here.

If using a 6Å area of influence for the ambient occlusion calculation, the optimum tested grid cell size would be $4S_r$ (7.2Å), according to the results presented in Chapter 3. Figure 4.17 shows the increase in time taken to render a frame with shadows, with a grid cell size of $4S_r$ rather than $2S_r$, compared to the increase in time taken to perform a locality search with a cell size of $2S_r$, rather than $4S_r$ for each of the proteins tested in both Chapters.

Figure 4.17 shows that for the four smaller proteins, the increase in shadow calculation time is greater than the reduction in time resulting from using a regular

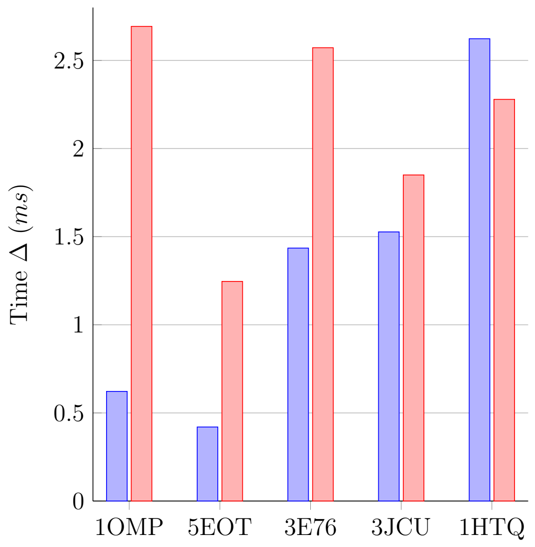


Fig. 4.17 Comparing the decrease in the locality search runtime (blue) with the increase in shadow casting rendering time (red), when using a cell size of $4S_r$, rather than $2S_r$.

grid tuned for the occluder list construction stage. Therefore, whilst $2S_r$ may not be optimum for this stage of the rendering pipeline, in most scenarios that use a small area of influence, it makes sense to optimise the grid for the shadow casting stage, and suffer a slight time penalty when calculating the ambient occlusion.

Combined performance

The performance when rendering both ambient occlusion and shadows, whilst rebuilding the data structures required was also measured. The results can be see in Figure 4.18.

Figure 4.18 shows that for the all of the proteins except the largest two, real-time performance is maintained whilst rendering per-pixel ambient occlusion and shadows. These performance figures suggest that the described rendering algorithms are theoretically suitable for use within our interactive docking system, provided the interacting molecules are not massive.

To demonstrate the effectiveness of the described algorithms, they were implemented in the software Haptimol Protein Trajectory Viewer (PTV). PTV allows the playback of molecular dynamics trajectories with per-pixel shading and lighting. PTV

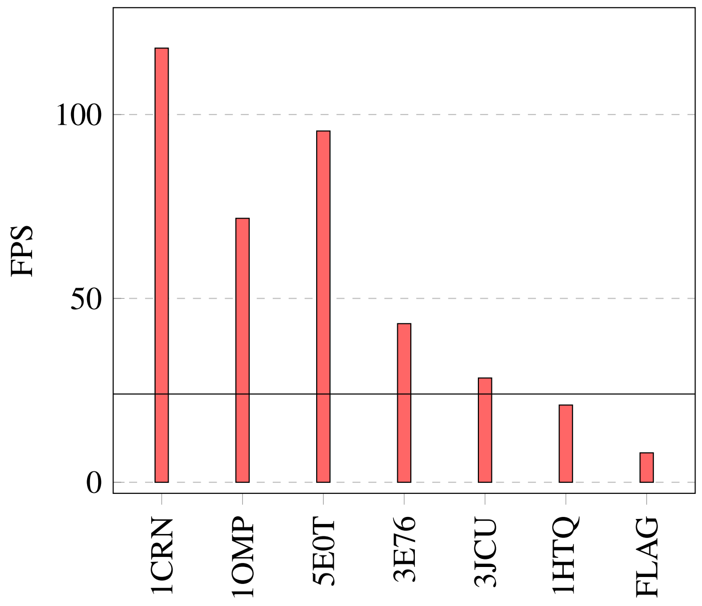


Fig. 4.18 Frame rates whilst rendering each of the proteins with all visual effects enabled and performing grid reconstruction every frame. A cell size of $n = 2$, and an AO area of influence of 6Å was used.

is the first molecular rendering application that supports real-time per-pixel shadows. A few applications support ambient occlusion, notably MegaMol[45].

4.6.2 Visual quality

Shadows

As stated, PTV is the first molecular rendering application that supports rendering per-pixel shadows on a dynamic scene. However, there are a few molecular renderers that support some implementation of real-time shadows. Of these, BallView with the RTFact ray tracer provides the highest quality shadows.

In order to show the effectiveness of the shadow casting algorithm, a flat structure was generated, with a collection of atoms casting a shadow onto it. This structure was then rendered with PTV and BallView .

Figure 4.19 shows the resulting render. Both BallView and PTV show a similar shadow, indicating the shape of the blue cluster of atoms. Comparing the renders, the shadows generated in BallView (Figure 4.19(B)) suffer from slightly jagged edges in

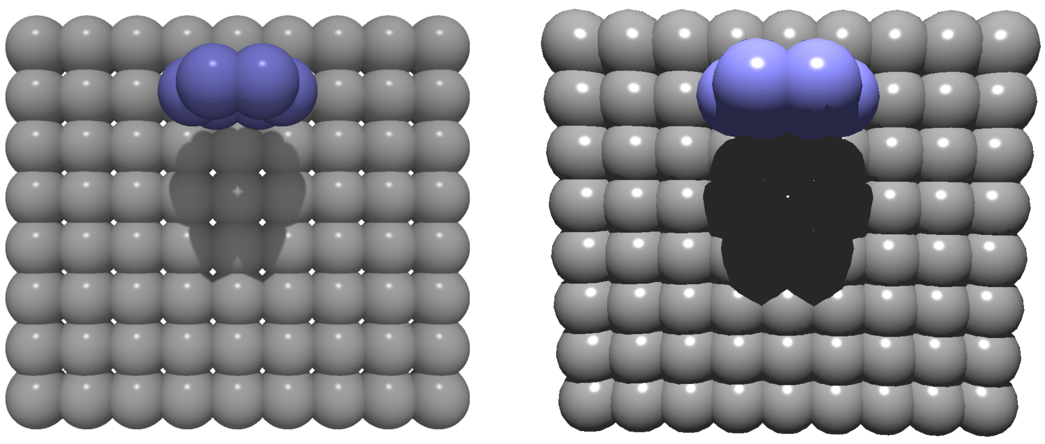


Fig. 4.19 Figure shows the ray cast shadows generated by *Protein Trajectory Viewer* (A) and RTFact within BallView (B). The light was placed approximately above the camera within all of the scenes.

places, and the hole in the centre of shadow is not very clear. Figure 4.19(A) shows the soft shadows included within Protein Trajectory Viewer work as designed, giving a smoother transition from dark to light than the other approach. The same effect could be generated using the ray tracing approach with an area light source but this would be computationally expensive.

Furthermore, PTV achieved 844.4 frames per second whilst rendering the displayed image, BallView managed 10.5 FPS. Within this test, the images were generated at a resolution of approximately 1000 × 1000 pixels. The achieved frame rates show that the render produced by Protein Trajectory Viewer is considerably less costly than that generated by BallView, whilst achieving similar quality.

Ambient Occlusion

To demonstrate the AO effect used, the contrived structure used to illustrate the shadowing algorithm was rendered with both the Tachyon ray-tracer included within VMD and also PTV. The ray-traced rendering (Figure 4.20) shows ambient occlusion between all of the atoms in the base of the model, and a small amount of ambient

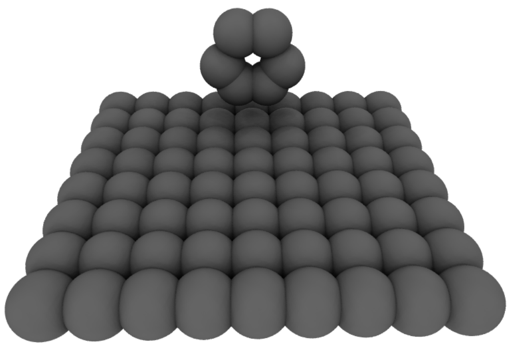


Fig. 4.20 Test scene raytraced using the Tachyon ray tracer within VMD.

occlusion on the base directly beneath the hexagonal structure. This rendering will serve as a ground truth as to what a test structure should look like.

Figure 4.21 (B), shows the test structure, rendered by PTV, with a small cut-off distance of 3Å. When compared with the rendering (A), where there is no AO at all, a definite darkening can be seen in between each of the atoms in the base, however, when compared with the ray-traced image in Figure 4.20, the occlusion caused by the hexagonal structure on the base is absent, and the darkening in the centre of the hexagon is more intense.

Figure 4.21 (C) shows PTV with a cut-off distance of 6 Å. Again darkening between the base atoms is shown, although it is more extreme between the atoms in the centre of the base than those at the edges. Also, the hexagonal structure causes some occlusion on the atoms immediately below it. Figure 4.21 (D) shows the rendering with a 10 Å cut-off. Here, the occlusion caused by the central hexagon is extreme, with considerable darkening occurring between the hexagon in the base. The occlusion between the atoms in the base is less well defined, as the intensity multiplier had to be significantly reduced to prevent the scene becoming over occluded; a side effect of the algorithm used to determine potential occluders.

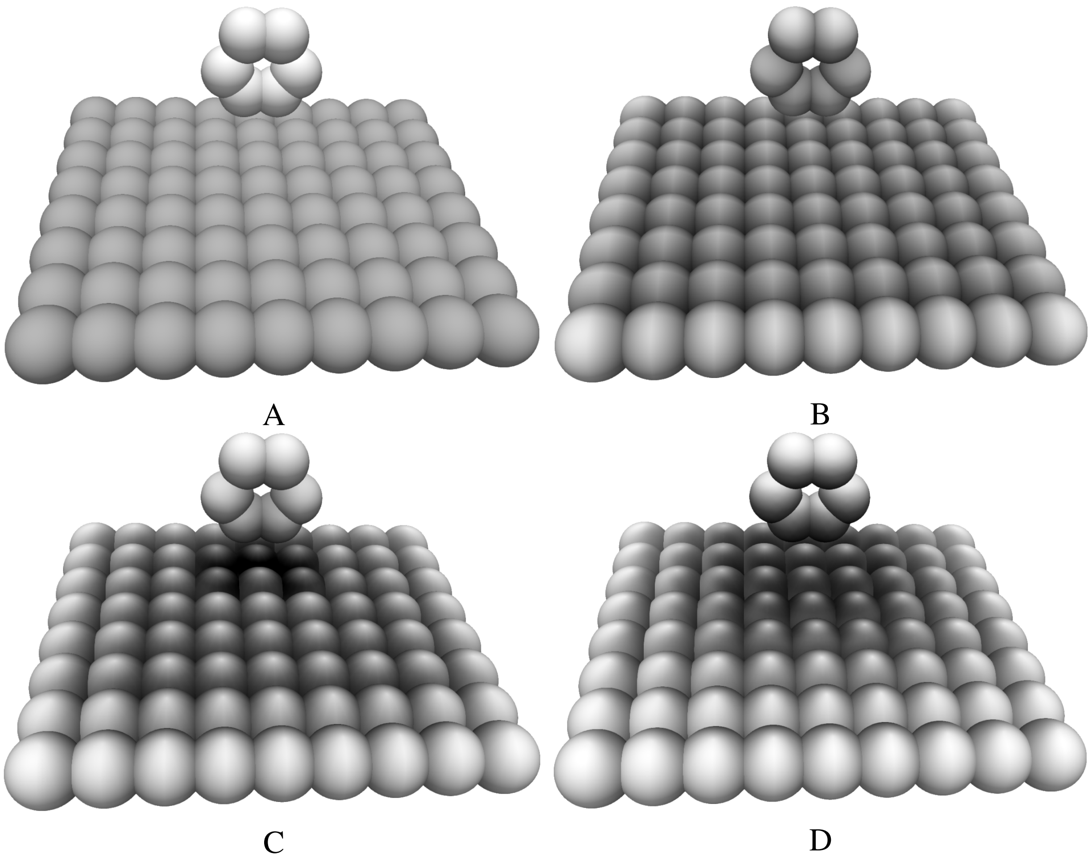


Fig. 4.21 Demonstrating the analytical ambient occlusion effect used by protein viewer with different cut-off distances. (A) No occlusion effect, (B) 3Å, (C) 6Å, (D) 10Å. The intensity values, *i* used were (A) 0.0, (B) 0.4172, (C) 0.0946 and (D) 0.0372.

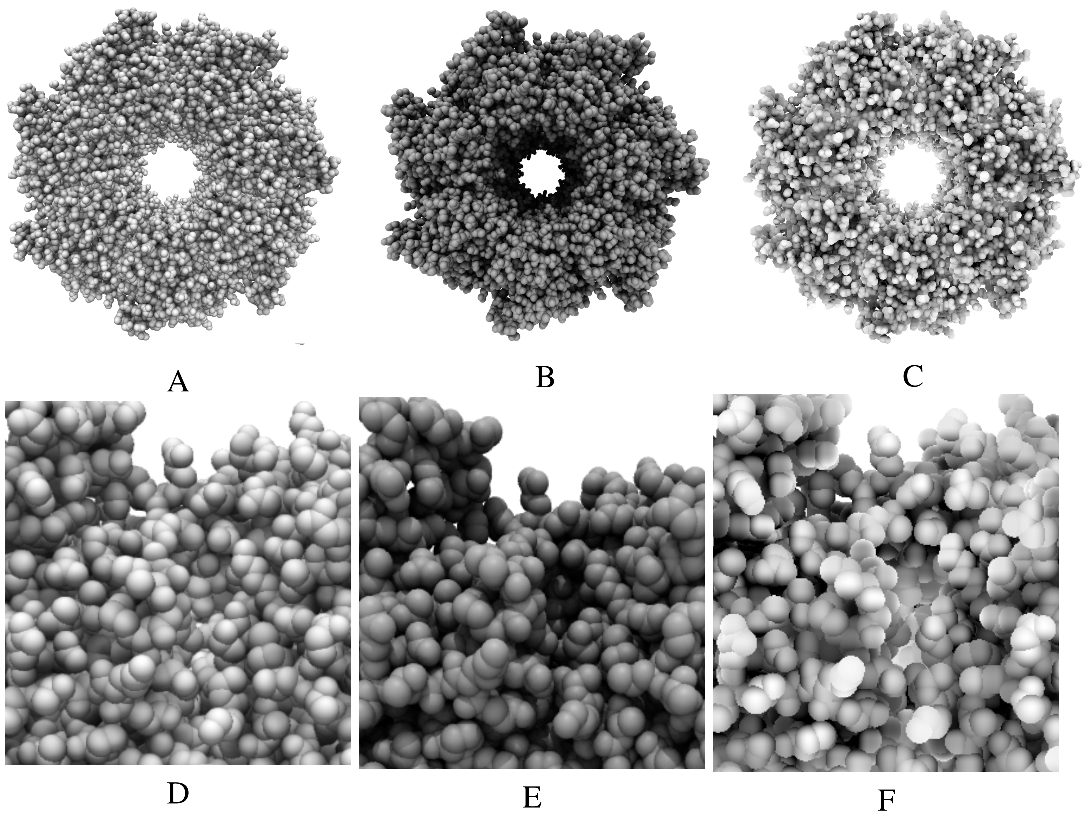


Fig. 4.22 GroEL with ambient occlusion only. Renders generated with (A & D) protein viewer (6Å c/o), (B & E) Taychon ray tracer in VMD, (C & F) Megamol (Volume size used: 28x28x42) (Unfortunately it is not possible to completely disable directional lighting within the Tachyon raytracer, so some shadows are cast.).

The over-occlusion effect is exaggerated in Figure 4.21 as a result of the tightly packed uniform nature of the contrived structure. When rendering a protein the occlusion effect is good, and comparable to a ray traced protein.

Figure 4.22 shows the AO effect generated by (A) PTV, (B) Tachyon within VMD and (C) MegaMol when rendering GroEL. The ray traced structure shows the best global occlusion, with the large hole in the centre clearly darkened. However, the render was produced using an off-line ray tracer: producing the same effect in real time is not currently feasible. Figure 4.22 (C) shows the ambient occlusion effect as generated by the real time rendering software MegaMol. At the chosen volume size, the occlusion effect generated by MegaMol highlights local pockets of occlusion very well, with the small crevices clearly highlighted, however the deeper parts of the structure are not highlighted especially well at all. The same can be said about

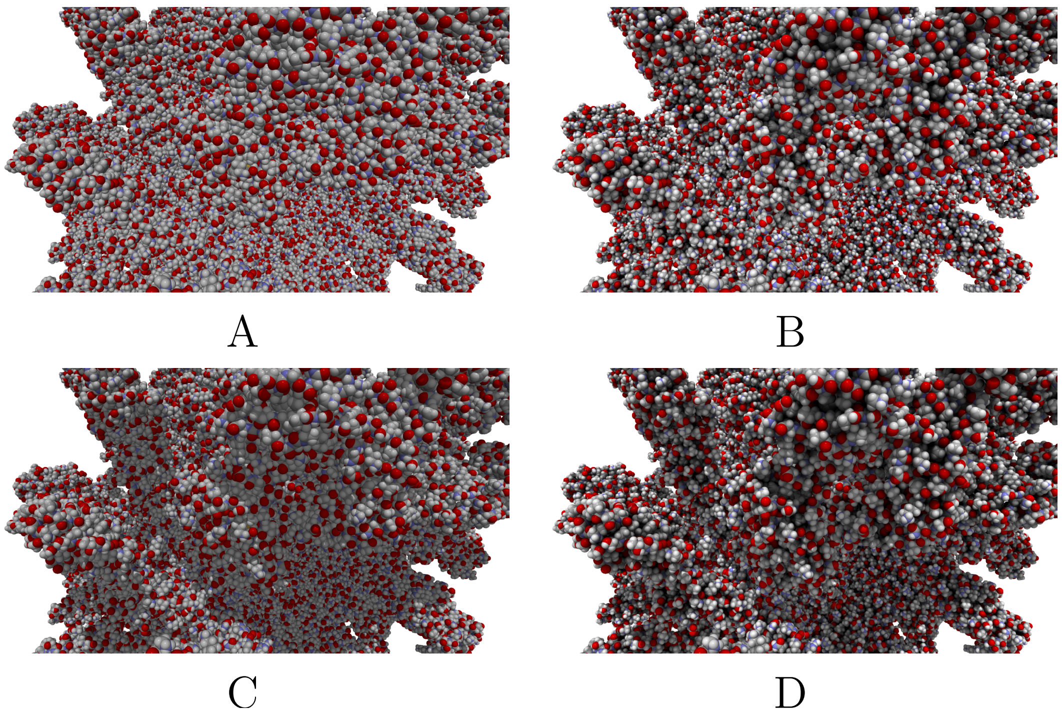


Fig. 4.23 Flagella Filiment rendered with each lighting effect. A) Directional Lighting only; B) Directional lighting and ambient occlusion; C) Directional lighting and shadows; D) Shadows, ambient occlusion and directional lighting

the occlusion generated by PTV. The renders produced by PTV and MegaMol are very similar overall, with the same areas being darkened in both images. However, the rendering approach used within MegaMol does have the edge performance wise, achieving 280 FPS, compared to 201 FPS achieved by PTV.

4.6.3 Final Effect

Combining the ambient occlusion effect with ray cast shadows reduces the impact of some of the short comings of the ambient occlusion algorithm. Figure 4.23 shows the developed lighting effects side by side. Note how the ambient occlusion is a local lighting effect, whilst the shadows are more global. Figure 4.24 shows GroEL rendered using (A) PTV with both shadows and ambient occlusion and (B) the ray-tracer in VMD.

The final renders are very similar - the smaller pockets within the ring of GroEL are nicely highlighted by the ambient occlusion, whilst the central hole is clearly shaded

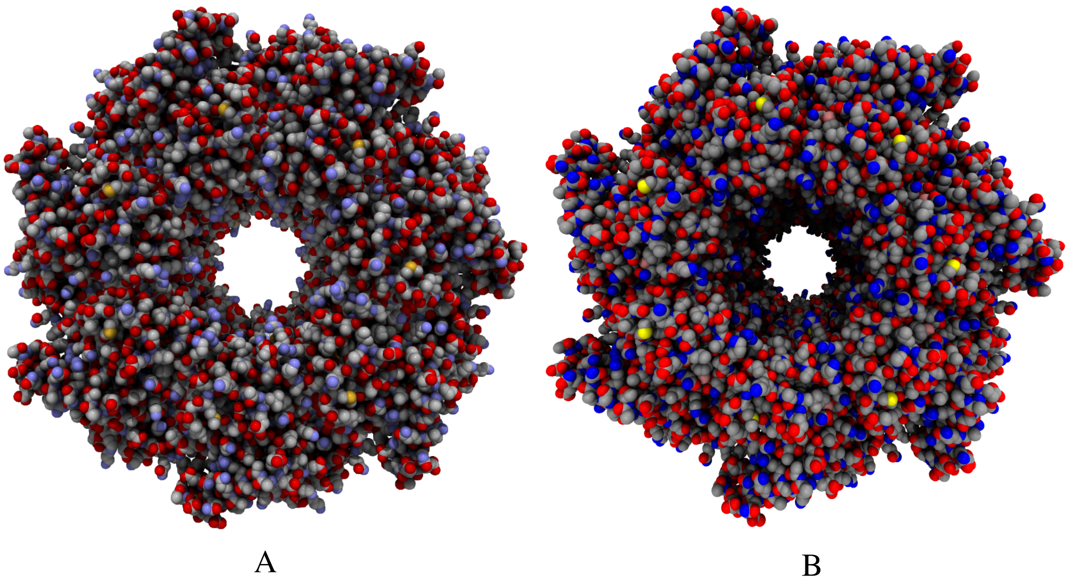


Fig. 4.24 Figure shows the final graphical effect *ProteinViewer* (A), Tachyon ray tracer in VMD (B). The light was placed approximately above the camera within both of the scenes.

using the shadow casting algorithm. Furthermore, the larger gaps in the outer ring are highlighted with the shadow algorithm although perhaps not as clearly as they are in the ray traced structure. Render (A), produced with PTV, achieved a frame rate of 175 FPS, whilst render (B), took 8.6 seconds to complete. The slight drop in quality is therefore acceptable for the massive increase in performance, allowing for real time visualisation.

4.6.4 Protein Trajectory Viewer

The algorithms described in this section were implemented within the software "Haptimol Protein Trajectory Viewer" (PTV). PTV allows the molecular trajectories of large structures to be rendered with high fidelity graphics. A protein trajectory comprises a large number of different viable poses a biomolecule could exist in. Viewing these trajectories can give researchers insight into the structure, dynamics and function of the biomolecule.

Biomolecular trajectories comprise discrete steps, with substantial intra-molecular movement between each step, owing to the computational cost of calculating each

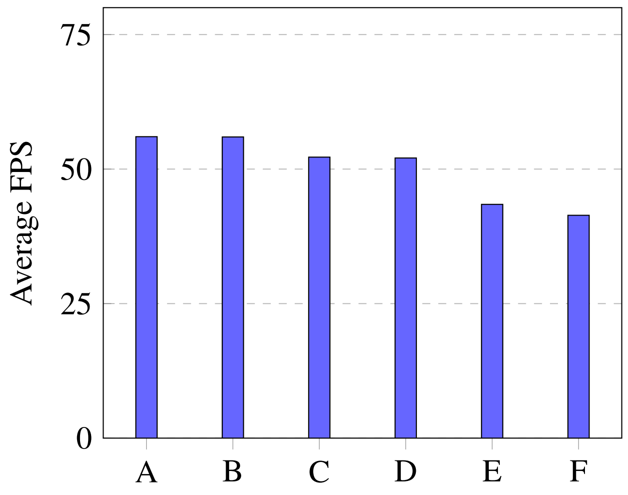


Fig. 4.25 Graph showing the effect on the FPS when rendering the flagellar filament (FLAG) for each of the different supported effects. The *x*-axis shows the shader mode selected: A - No effects enabled, B - Directional lighting only, C - Ambient occlusion only, D - Directional lighting and Ambient Occlusion (no shadows), E - Shadows and Directional lighting, F - All effects enabled. The biomolecule was positioned to fill the screen.

time step. Therefore, when rendering a trajectory, a single pose may remain on the screen for several seconds; the grid structure only needs to be rebuilt when the pose changes.

Figure 4.25 shows the achieved frame rate when rendering the Flagella filament trajectory, whilst reconstructing the structure every time the biomolecules pose changed. The trajectory was advanced once every 2 seconds. The biomolecule was positioned to fill the screen.

Figure 4.25 shows that whilst rendering the trajectory, a real-time frame rate was achieved for the large flagella, even with all of the graphical effects enabled. In the implementation tested, the regular grid was reconstructed on demand, resulting in a reduction in frame rate whilst the structure was being rebuilt. However, if the structure construction was double buffered, with the subsequent frames acceleration structure being computed whilst the current frame is rendered, this drop in frame rate could be reduced.

4.7 Discussion & Conclusion

The objective set out in the introduction of this chapter was to produce an algorithm that would allow per-pixel shadows and ambient occlusion to be rendered on a deforming protein at an interactive frame rate. Within the results section, it was shown that the described algorithms were fast enough to render proteins up to approximately 75,000 atoms in size whilst simulating a dynamic scene, at a refresh rate greater than 24 FPS. The effectiveness of the algorithms is demonstrated in a MD trajectory viewing tool, Haptimol Protein Trajectory Viewer (PTV), which allows large trajectories of biomolecules to be rendered with per-pixel lighting effects.

Adapting ray-casting shadows for use with a dynamic scene proved reasonably inexpensive, as a result of the optimised grid construction algorithms developed in Chapter 3. The resulting directional shadows are of similar quality to true ray-traced shadows, and yet much less computationally expensive to produce.

The ambient occlusion algorithm utilised by PTV produces a good approximation of per-pixel ambient occlusion in real-time. When compared with MegaMol, the results are similar, with common areas of occlusion between the two structures. The algorithm used by PTV provides a good approximation of local ambient occlusion, although over-occlusion can occur when the cut-off distance is large. The over-occlusion is caused by atoms which do not have a direct line of sight to the pixel being included in the occlusion calculation.

To prevent this from occurring, each fragment would need to test each potential occluder for visibility based on the other atoms within the scene, then scale the calculated occlusion by that factor. This could be achieved utilising a ray-casting technique, however it is unlikely this could be achieved in real time, for any but the smallest structures. When the cut-off distance is small, over-occlusion is not a significant problem because atoms that are completely obscured from any one pixel fall outside of the area of influence.

Within the results of the performance tests, the frame rates achieved whilst rendering 1OMP are noticeably lower than those achieved when rendering 5E0T, despite each containing a similar number of atoms. This occurs because the model of 1OMP contains hydrogen atoms, whilst 5E0T does not. As hydrogen atoms are bound tightly to larger atoms, the number of atoms occupying each cell increases, which results in more intersection tests during the shadow ray casting, and an increase in the number of occlusion tests during the AO calculation, resulting in a drop in the frame rate.

Although the AO method used within MegaMol offers higher performance, the quality of the ambient occlusion effect generated by MegaMol is highly dependent on the volume size used. To generate the render in Figure 4.22, a volume size of 28 x 28 x 42 voxels is used, a value that was determined empirically. In a molecular docking application, the dimensions of the protein could change significantly from one scene to the next, which would require the dimensions of the volume to change, in order to maintain the quality of the AO. This could prove challenging to automate, and awkward to do manually whilst performing docking. Therefore, despite the more local effect of the algorithm presented in this chapter, the AO effect offered is more consistent, and therefore easier to use in an interactive environment.

The focus of this chapter has been on generating the high fidelity lighting effects, whilst maintaining real-time performance. The described effects achieve this: the effects are calculated per pixel, with minimal aliasing in all situations. However, they are computationally expensive to compute in real-time, and other approaches for generating these effects could offer better performance, albeit at the expense of quality.

Testing carried out by Easdon[32] on his static protein viewer, which used a similar, but not identical, approach to calculate ambient occlusion, revealed that SSAO was faster to compute. As the algorithm presented here requires more work than Easdon's, in order to support a deforming protein, it follows that the screen space approach would indeed offer faster performance, however Easdon found that the SSAO approach produced an extremely local effect, and required a filter pass to ensure the light effects were smooth; the final result being less aesthetically pleasing than the analytical and ray-traced approaches.

Faster rendering times could also be achieved by approximating the shadows using a shadow map, however again, at the expense of quality, the focus of this chapter. The best way to improve the quality of both the shadows and the ambient occlusion would be to perform full ray-tracing on the scene. Recent advances in computer hardware, namely specialised compute units for ray-tracing, make such a possibility more realistic than it was even at the start of this project, however that hardware is still expensive, and not yet widespread. Therefore, the techniques discussed in this Chapter offer a good balance between quality, accessibility and performance. Also, the primary motive for developing these techniques is for use within a interactive molecular docking tool. The methods described here will always be less computationally expensive than full ray tracing, leaving a greater proportion of the available computing power available for simulating biomolecular deformations.

Theoretically, the methods described in this chapter offer sufficient performance to be incorporated directly into interactive molecular docking software, however, this performance is achieved with near 100% GPU utilisation, leaving nothing in reserve to perform biomolecular interaction calculations with. With the smaller molecules, the frame rate could be limited to 30 FPS, freeing up resources to compute the required interaction calculations. Ideally, a dual GPU system would be utilised, with one GPU used for rendering, one to perform the molecular-docking related calculations, however there are difficulties employed with using such an approach, not least the scarcity of dual GPU systems.

In this chapter, an approach to rendering per-pixel shadows and ambient occlusion on a deforming biomolecule has been presented, and implemented in the software "Haptimol Protein Trajectory Viewer." The presented algorithms produce smooth, high fidelity lighting and are fast enough to render molecules that change pose every frame, for structures up to 75k atoms in size.

In the next chapter, the development of an interactive molecular docking application that supports receptor flexibility, and can theoretically make use of the rendering algorithms described here, is presented.

Chapter 5

Interactive molecular docking with receptor flexibility

5.1 Introduction

The content of this thesis has so far focused on the rendering of a single protein molecule, with high quality, per pixel, visual effects that can improve depth perception in a dynamic scene. Within this chapter, the focus moves onto calculating the conformational change a protein undergoes as a response to ligand binding, in real time, for use within an interactive molecular docking environment.

Currently, the majority of interactive molecular docking applications model protein interactions as rigid, owing to the computational expense of modelling biomolecular flexibility. In this chapter, a method for calculating the deformation of a protein at a haptic refresh rate is presented and incorporated within an application: Haptimol FlexiDock.

By utilising a feature of the internal motions of proteins, whereby most of the fluctuation occurs within a so-called "important subspace"[2,48,70] Haptimol FlexiDock calculates a receptor's conformational change as a response to the forces generated from interactions with a ligand in real time.

This chapter is organised as follows: Firstly, a review of the methods used to model protein flexibility within automated docking is presented, then the current state of the haptic-assisted interactive docking field is reviewed. Following that, the method used to reduce the dimensionality of the docking problem is introduced. Then the chapter describes the implementation of Haptimol FlexiDock and evaluates the effectiveness of the described techniques. Finally, the graphical effects presented in Chapter 4 are incorporated into FlexiDock, and benchmarked.

5.1.1 Contributions

This chapter's main contributions are:

- A GPU based approach for computing the deformation of a biomolecule in real time.
- The first haptic-assisted interactive molecular docking tool that incorporates receptor flexibility whilst maintaining a haptic refresh rate of at least 500Hz.

5.2 Background

Molecular docking is a method which predicts the preferred orientation of one molecule when its binds to a second in order to form a stable complex. Molecular docking is frequently used in the field of structure-based drug design, as a result of its ability to determine how small molecules (ligands) bind to compatible target types. See Chapter 2, Section 2.1 for more details.

Molecular docking is primarily concerned with simulating how two biomolecules interact when they are in close proximity. Ideally, it aims to predict a confirmation involving the ligand and receptor (or protein), such the the overall free energy, (the amount of internal energy of a thermodynamic system that is available to perform work) is minimised.

There are two sub-fields within the field of molecular docking: Automated docking and interactive docking. In automated docking, the computer attempts to determine how two biomolecules fit together, with minimal user input. Interactive docking puts the user in charge of the docking process, allowing them to use their judgement and intuition to determine if and how biomolecules fit together.

Automated

Within automated docking, modelling flexibility has been widely explored[6,98,107]. The main approaches used to achieve this are reviewed here, in order to determine whether any of the methods used can be incorporated into an interactive docking system.

Current automated docking algorithms can be said to model flexibility either explicitly or implicitly. Approaches that explicitly model flexibility actively explore different protein conformations during the docking process. Implicit methods model flexibility indirectly, rather than by actively deforming the structure.

There are four main categories of algorithms that model protein flexibility. These are shown in Figure 5.1. Soft docking and ensemble docking model flexibility implicitly, whilst selective and on-the-fly docking model it explicitly.

Soft docking is the least computationally expensive approach to modelling flexibility[61]. In soft docking, small overlaps are allowed between the receptor and ligand by softening the van der Waals (vdW) potentials. By using a more permissive repulsive term in the Lennard-Jones (LJ) potential, a slightly larger binding site, which simulates minor conformational plasticity, can be emulated[6]. As the receptor's conformation is not changed, it is an implicit method of modelling flexibility. The main advantage soft docking has over other approaches is that it costs nothing extra to compute compared to rigid docking. However, soft docking can only accommodate small, local conformational changes and not the larger global motions that are often related to functional movement. Soft docking has been used as in complement with more complex docking methods[126,146].

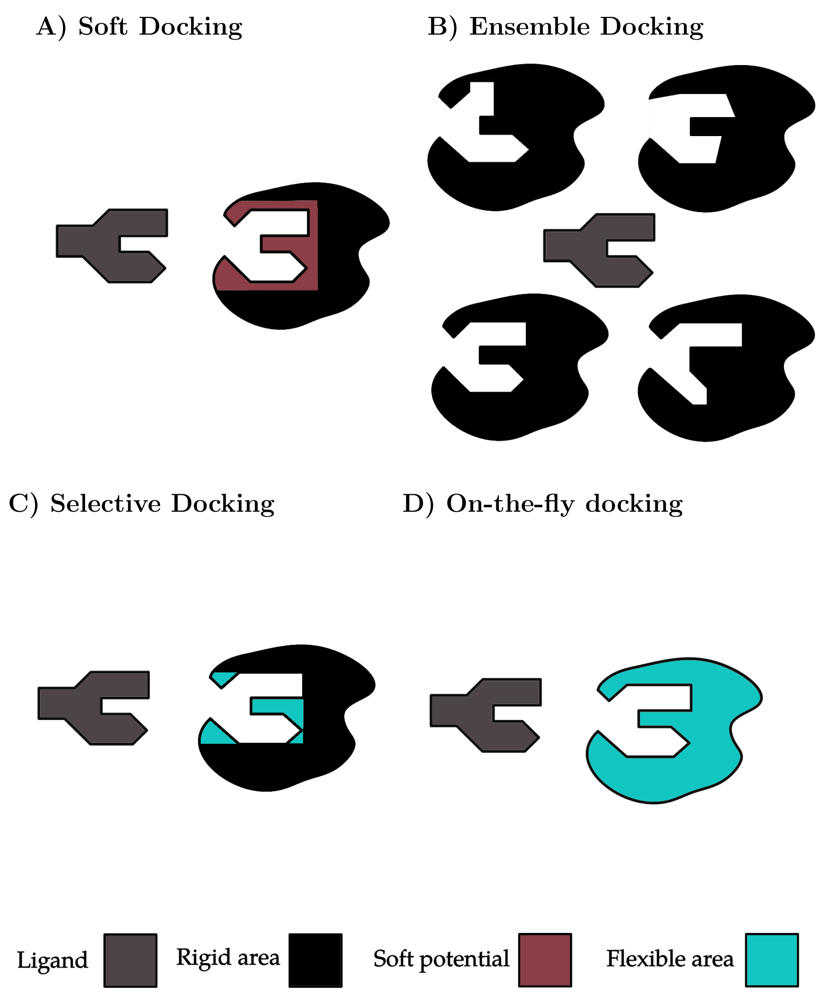


Fig. 5.1 A diagram showing the four main approaches to modelling flexibility used by automated docking approaches. (A) Soft-docking. The interaction forces are softened to increase the size of the docking site. (B) Ensemble docking, docking is attempted with a number of different receptor poses, generated either computationally or experimentally. (C) Selective docking: Flexibility in specific protein side chains is modelled, with the majority of the receptor remaining rigid. (D) On-the-fly docking: receptor is fully flexible, and its degrees of freedom are explicitly explored during the docking process. This is the most computationally expensive approach to docking.

Ensemble docking attempts to dock a ligand to an ensemble of receptor conformations instead of a single pose[71]. Flexibility is accounted for implicitly, as the protein will not deform in response to the ligand during the docking process. Studies have shown that ensemble docking provides significant improvement over rigid docking[18,128]. However, others have highlighted that the additional poses can increase the number of false positives generated during docking[10]. Furthermore, testing a large number of poses instead of a single pose increases the amount of computation required by each docking experiment.

In other docking approaches, receptor flexibility is modelled explicitly. Owing to the computational cost of exploring all of a proteins DoFs, a number of methods have been employed to reduce the complexity of the problem.

The earliest approaches, labelled "Selective Docking" in Figure 5.1, modelled partial, rather than full, flexibility within the receptor, reducing the amount of computation required[81]. Rotamer libraries, libraries that contain precomputed viable poses of side chain conformations, have also been used to reduce the computational cost of calculating new side-chain positions, however doing so reduces the resolution of the flexibility to the number of poses within the library[6].

On-the-fly docking attempts to explicitly model full molecular flexibility by considering both the interaction forces, bond angles and each different type of bond during docking. This is computationally expensive, so various optimisations have been proposed to reduce the computational cost of the docking process.

One approach, presented by Sherman et al.[126], is to dock the ligand into the receptor using soft docking, then, various conformations of the receptor are explored using rotamer libraries. Other approaches also start with soft docking, however, after a potential pose has been found, the ligand and receptor poses are optimised using Molecular Dynamics (MD)[143], Monte-Carlo methods[17] or energy minimisation[7].

Lower-dimensional representations of the receptors internal modes have also been used to model flexibility. These approaches incorporate only the dominant modes of the receptor's internal motion into the docking process. Both Normal Mode Analysis (NMA)[17,69,160] and Principle Component Analysis (PCA)[142,159] have been used as dimensionality reduction techniques.

These techniques have been shown to be an effective way of reducing the computational expense of incorporating flexibility: Zacharias[159] incorporated "soft" modes into the docking process. These modes were generated by performing an MD simulation of the target protein (the receptor), calculating a covariance matrix of atomic fluctuations and performing PCA on the resultant matrix. The resulting eigenvectors with large eigenvalues describe flexible degrees of freedom of the motion. These eigenvectors were then incorporated into Zacharias[159] docking system. It was found that whilst rigid docking failed to identify a docking site close to the experimentally derived site, including the modes resulted in a docking pose close to the experimentally derived pose. Tatsumi et al.[142] notes that Zacharias[159] system ignores local flexibility, and so demonstrated a hybrid algorithm which uses a similar approach to Zacharias[159] for global motions and conventional MD for local motions. They found that their hybrid method reproduced global fluctuations that were not present in ordinary docking simulations.

Interactive Docking

Despite the plethora of ways automated docking has incorporated flexibility, few interactive approaches model flexible docking. The primary reason for this is related to the time constraints that come with working at an interactive refresh rate. For a smooth visual experience, computer displays have to be updated at a refresh rate of at least 24 Hz. Modern haptic technology requires a higher refresh rate: the haptic device needs to be updated at a refresh rate greater than 500 Hz, ideally 1 kHz, because of the sensitivity of the human haptic system[33,96]. If a refresh rate lower than this is used, device vibrations and force discontinuities can occur. As a result of this, the majority of haptic-assisted interactive molecular docking systems limit themselves to rigid docking.

Most existing haptic-assisted molecular docking systems make use of precomputed force grids[110] to limit the amount of computation that has to be calculated in real time. Grid based applications, first proposed by Brooks et al.[19] treat either one, or both of the interacting biomolecules as rigid, and then pre-compute the desired interaction forces (usually vdW and electrostatic), around the receptor[13,15,19,77,83,136], or part of the receptor[155]. These pre-computed forces are then stored in a grid that is placed around the receptor, and then queried during the docking session to determine the forces between the receptor and ligand at that point in space. The fundamental limitation of this approach is that owing to the large pre-computation step, receptor flexibility cannot be accommodated, as the grid would need to be recomputed after every deformation step. Further limitations are that rough force transitions can be felt at grid cell boundaries[155], and high memory consumption caused by the use of a precomputed grid.

Modern CPUs are fast enough to calculate all of the interaction forces between the receptor and ligand in real time, when the receptor and ligand comprise few atoms[39,52,102]. A modern processor can typically achieve a haptic refresh rate of 500 Hz or more, for molecules up to a few hundred atoms in size[130]. Molecule sizes much larger than this are impractical on the CPU. Modern GPUs (Graphics Processing Units) are, however, suited to this problem because they allow a large amount of computation to be performed in parallel. HaptimolRD, a docking system presented by Iakovou et al.[57], made use of the GPU to model rigid docking between two large molecules. Importantly for this docking approach, in which receptor flexibility is modelled, the approach by Iakovou et al.[57] computes the interaction forces in real time, rather than relying on any pre-computation, and as a result, it can be applied to a flexible docking problem.

The earliest haptic-assisted docking application that modelled some degree of flexibility was presented by Brooks Jr et al.[20]. The application allowed users to adjust flexible bonds within the ligand only, before using a pre-computed force grid during the haptic session. This early system only supported updating the haptic device at 60 Hz, and was limited to performing docking between small molecules; a receptor comprising 600, and a ligand comprising up to 60, atoms[106]. Bayazit et al.[13] and Lai-Yuen and Lee[77] also presented interactive docking approaches that utilised a force grid to accelerate docking, whilst also supporting ligand flexibility. Bayazit et al.[13] presented a hybrid system, rather than a fully interactive docking application, in which the user explored the receptor with the haptic device, then probabilistic roadmap planning methods were used to dock the ligand. Lai-Yuen and Lee[77] achieved ligand flexibility by using a conformational search algorithm to determine alternate ligand conformations during docking. The complexity of the search was reduced by grouping atoms into clusters, allowing ligand flexibility to be modelled within the time limit imposed by the haptic device, however grouping the atoms in this fashion limits the accuracy of the deformation.

Other haptic assisted interactive docking approaches that incorporate a degree of flexibility include the work by Daunay et al.[28], Anthopoulos et al.[4] and Zonta et al.[162]. Daunay et al.[28] developed a system that modelled flexibility by using a molecular dynamics engine to compute the relevant forces. This approach proved too costly to compute in haptic time, so wave transformations were used to bridge any gaps between the rendering and simulation. Zonta et al.[162] presented a system that modelled ligand flexibility by using a third party library to accelerate the force computation, however the system only supported small ligands. Anthopoulos et al.[4] incorporated a GPU-accelerated force calculation approach within their molecular modelling system[5]. The methods presented by Anthopoulos et al.[5] model receptor flexibility to some degree, however the forces on the haptic device are only updated at the refresh rate of the display: 33Hz, well below that required for a smooth haptic experience.

Currently, haptic-assisted interactive molecular docking systems that achieve a haptic refresh rate of at least 500Hz have either limited themselves to docking rigid molecules, (comprising up to 184k atoms[57]) or rigid receptor, flexible ligand docking where the ligand comprises a few atoms[162].

In order to model flexibility in a haptic assisted docking environment, an efficient approach to modelling flexibility is needed. Of the techniques used in automated docking, soft docking, selective-docking and on-the-fly docking are theoretically viable to use in an interactive environment. Soft docking would be straight forward to include within an interactive system, as it is no more computationally expensive than rigid docking, which has already been solved. However, the conformational change modelled by soft docking is limited to small local conformation changes, rather than the full global motion that is often required for successful ligand binding. Although less computationally strenuous than full, on-the-fly docking, selective docking is limited in that only a small number of bonds are modelled as flexible, which can limit, or hide, relevant motion from the user.

The only methods that fully model flexibility are the on-the-fly docking approaches. Of these, an adaptation of the dimensionality reduction technique utilised by Zacharias[159] and Tatsumi et al.[142] is most likely to be suited for use in a haptic-assisted interactive environment, because the dimensionality reduction reduces the amount of computation that has to be performed.

Dimensionality Reduction

Zacharias[159] reduced the dimensionality of the docking problem in their automated docking system, by using PCA to extract the dominant fluctuations, which often contain the functional movement from an MD trajectory, with success. This approach works as a result of linear response theory (LRT): the structural changes to proteins that occur during docking correlate to the fluctuations that occur to it in a ligand free state.

Ikeguchi et al.[59] confirmed the validity of LRT by performing MD simulations on three protein systems, and calculating their covariance matrices. Then, they placed the proteins respective ligands into their experimentally derived binding positions and calculated the resulting deformation. The results were consistent with observation from crystal structures; structures that have been determined using x-ray crystallography.

Ikeguchi et al.[59] also confirmed that when PCA is performed on the covariance matrices, and the largest eigenvectors used in place of the entire covariance matrix, a large amount of the protein's experimentally derived movement still occurs, likely because a protein comprises rigid parts that are joined together, which move collectively[48]. The subspace within which most of the fluctuation occurs is often referred to as as the "important subspace[2,48,70]," an aspect of protein dynamics that is pivotal for the proposed system to run at a haptic refresh rate.

HaptimolENM[134] is a tool that utilises an elastic network for studying a protein's deformation when forces are applied to specific, user specified, atoms. In HaptimolENM, a haptic device was used as a way to apply forces to the atoms. HaptimolENM made use of the important subspace of proteins in order to allow large proteins to be studied on regular desktop PCs. By using this technique, the memory consumption and computational cost of calculating the deformation was reduced. Although HaptimolENM[134] utilises a similar dimensionality reduction technique that will be used in this chapter, there are substantial differences between the approaches used. Primarily, HaptimolENM uses NMA combined with an elastic network to determine the motions of a single molecular pose, rather than PCA on a molecular trajectory. Also, HaptimolENM was designed to model a protein's response when forces were applied to a few individual atoms. This allowed optimisations to be made to the algorithm, reducing the computational cost of computing the flexibility. In this chapter, the interactions between two proteins will be modelled, a far more computationally challenging task.

5.3 Method

In order to model receptor flexibility whilst maintaining a 500Hz haptic refresh rate, three main tasks have to be completed within a 2 ms time limit:

1. Calculate the current interaction forces between the receptor and ligand.
2. Calculate the receptor's deformation in response to the interaction force.
3. Apply the conformational change to the receptor.

In addition to these three steps, user input has to be handled and separately, the visual protein depiction has to be rendered. These steps, forming the haptic loop, are shown in the flowchart in Figure 5.2.

5.3.1 Force Calculation

Similar to other haptic-assisted interactive molecular docking approaches, only the vdW and electrostatic interactions between the receptor and the ligand are modelled. Iakovou et al.[57] presented a GPU accelerated algorithm to calculate both of these forces between two molecular structures, that is fast enough to be used with a haptic device. Iakovou's approach solves the equation

$$\vec{\boldsymbol{f}}_i = \sum_{j=1}^{M} \left(\left(24\varepsilon_{ij} \left[\frac{2\sigma_{ij}^{12}}{r_{ij}^{13}} - \frac{\sigma_{ij}^{6}}{r_{ij}^{7}} \right] + \frac{q_i q_j}{4\pi\varepsilon_0 \varepsilon r_{ij}^2} \right) \vec{\boldsymbol{r}}_{ij} \right), \tag{5.1}$$

which calculates the interaction force between atom i in the receptor, and all of the atoms, labelled j, within the ligand, for each atom in the receptor in parallel.

The haptic device "holds" the ligand and transmits the force upon it to the user. The force on the ligand from the receptor is given by $-\sum_{i=1}^{N} \vec{\boldsymbol{f}}_i$. The resulting force is

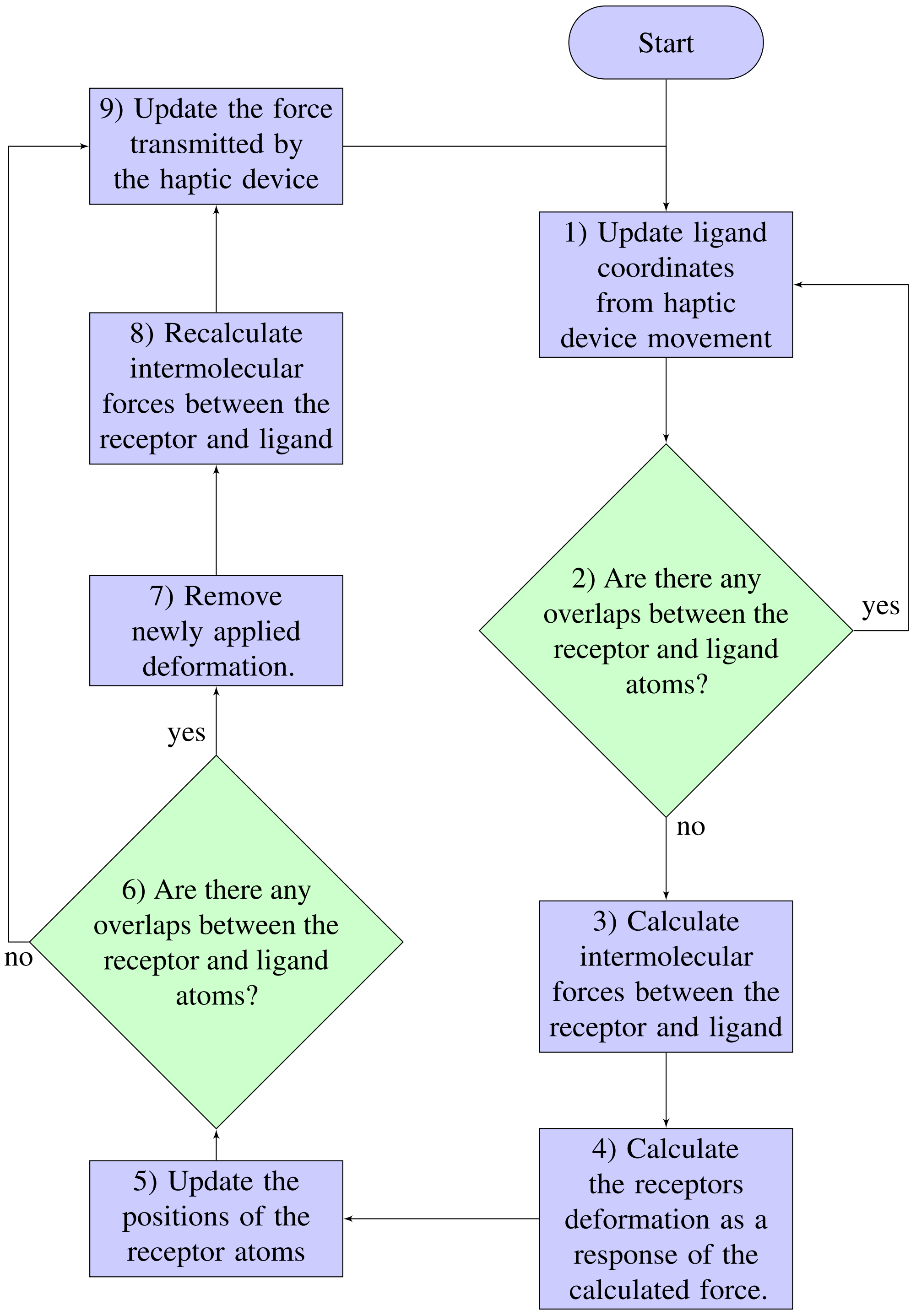

Start
9) Update the force transmitted by the haptic device
1) Update ligand coordinates from haptic device movement
8) Recalculate intermolecular forces between the receptor and ligand
2) Are there any overlaps between the receptor and ligand atoms?
yes
7) Remove newly applied deformation.
yes
no
6) Are there any overlaps between the receptor and ligand atoms?
no
3) Calculate intermolecular forces between the receptor and ligand
5) Update the positions of the receptor atoms
4) Calculate the receptors deformation as a response of the calculated force.

Fig. 5.2 Flowchart describing the *haptic loop* ; All of the steps needs to be completed within a 2 ms window. Section 5.3.1 describes Step 3 of the loop, Section 5.3.3 describes Step 4 and Step 5.

then converted into Newtons, and scaled before being transmitted through the haptic device. Three force scaling profiles, as described by Iakovou et al.[58], are available for use. These profiles are used to ensure a good range of forces can be felt by the user. Within Haptimol FlexiDock the ligand is currently modelled as rigid.

In Equation 5.1 ε_{ij} and σ_{ij} are Lennard-Jones parameters that depend on the characteristics of the interacting atoms, r_{ij} is the distance between the two interacting atoms, q_i and q_j are the atomic charges of the two atoms, ε_0 is the permittivity of free space, ε is the relative permittivity dependent on the dielectric properties of the solvent and $\vec{\hat{\boldsymbol{r}}}_{ij}$ is the unit vector in the direction of atom i to atom j. Torques are omitted, as it is not possible to render them on low cost haptic devices.

Within this chapter, parameters for ε_{ij}, σ_{ij}, q_i and q_j were taken from the ffamber03 forcefield[31], however any forcefield could be used to provide these values. The Lorentz-Berthelot rules were used to compute ε_{ij}, σ_{ij}. These rules state that $\varepsilon_{ij} = (\varepsilon_i\varepsilon_j)^{\frac{1}{2}}$ and $\sigma_{ij} = \frac{1}{2}(\sigma_i + \sigma_j)$, where σ_i, σ_j, ε_i and ε_j are the Lennard-Jones parameters of atoms i and j, as taken from the forcefield.

The Coulomb constant, $\frac{1}{4\pi\varepsilon_0}$ is set to 138.935485 kJ mol^{-1} nm e^{-2} and an approximation of the Coulomb screening potential of water, as described by Mehler and Solmajer[93] is used for ε.

For the deformation calculation, discussed in Section 5.3.3, the per atom forces are translated into receptor space, and then rearranged into a column vector, as depicted in Figure 5.3.

Coulomb screening with implicit water

Usually, molecular interactions occur within a solvent, usually water, rather than in a vacuum. Explicitly modelling water within the interactive environment would increase the computational overhead of performing the simulation, and detract from the user experience: both the ligand and receptor would be obscured with water molecules, in the haptic and visual (See Figure 5.4) scenes. Although the water molecules could be

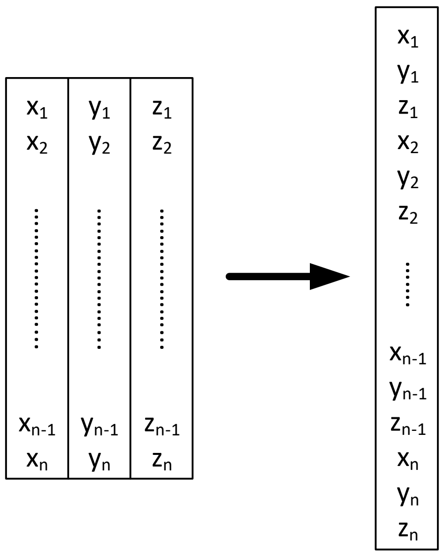


Fig. 5.3 Diagram showing how the three by N matrix of forces is rearranged into a single column vector.

hidden in the visual render, both collisions with the water and the minor interaction forces occurring between the water and the ligand would be felt through the haptic device as noise. Furthermore, computing the interactions between the water molecules and the ligand would add a significant overhead to the force calculation algorithm.

Although water cannot be modelled explicitly within the haptic simulation, it can be modelled implicitly, using the equation

$$\varepsilon = A + \frac{B}{(1+ke^{-XBr})}. \tag{5.2}$$

Mehler and Solmajer[93] demonstrate that the Equation gives a good approximation of the electrostatic screening effect of water with minimal additional computational load, and without the drawbacks of explicitly including it. Therefore, it should be suitable for use within an interactive docking application.

In Equation 5.2, A = -20.929, B = 99.329, k = 3.4781, X = 0.001787 and r is equal to the distance between the interacting atom pair. These values were presented by Mehler and Solmajer[93]. Figure 5.5 shows how ε increases with distance.

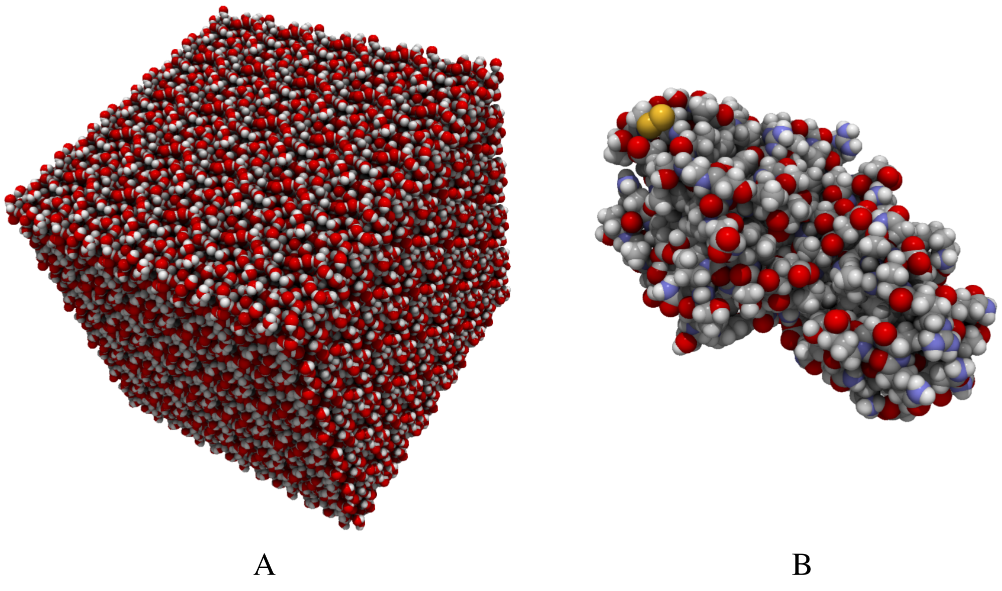


Fig. 5.4 A molecular structure with (A) and without (B) explicit water.

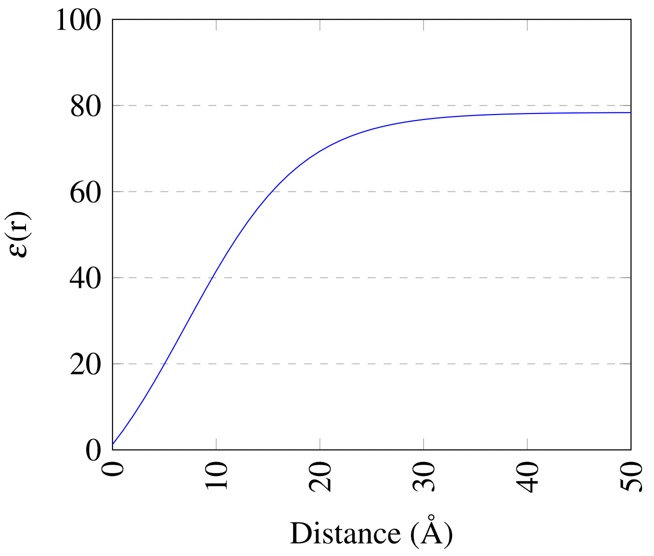


Fig. 5.5 Graph showing how the value of ε, when calculated with Equation 5.2 increases as the inter-atomic distance increases.

Implementation - Force calculation

Algorithm 9 describes the core of the force calculation method. A single thread is created for each receptor atom. This thread then computes the interaction force between the receptor atom and all of the atoms in the ligand. The per atom interaction force is then translated into receptor space and stored within an array, *perAtomForce*, equivalent to $\boldsymbol{F}$ within Equations 5.3 and 5.4. Each of the threads within a work group then sum their interaction forces together, to give *perWorkGroupForce*, which is then copied back to the CPU. All of the work group's *perWorkGroupForce* values are then summed, giving the net force on the ligand, which is then rendered upon the haptic device.

Algorithm 9 deriveInteractionForce(r, l,n_r,n_l) **return** $perAtomForce, perWorkGroupForce$

```
Require: $r$ {Receptor atom information }
Require: $l$ {Ligand atom information.}
Require: $n_r, n_l$ {number of atoms within the receptor ($n_r$ and ligand $n_l$}
Require: $groupId$ {Work group identifier}
 1: for all $i \leftarrow 1$ to $n_r$ in parallel
 2:     $atomForce = (0,0,0)$
 3:     for $j \leftarrow 1$ to $n_l$ do
 4:         $atomForce \leftarrow atomForce +$ computeInteractionForce($r(i)$, $l(j)$);
 5:     end for
 6:     $wi \leftarrow 3 \times i$
 7:     $perAtomForce(wi) \leftarrow -atomForce(1)$
 8:     $perAtomForce(wi+1) \leftarrow -atomForce(2)$
 9:     $perAtomForce(wi+2) \leftarrow -atomForce(3)$
10: end for
11: $perWorkGroupForce(groupId) \leftarrow$ parallelSummation($atomForce$)
```

The per atom force array, arranged as depicted in Figure 5.3, is passed to the deformation calculation kernels, in order to calculate the receptor's movement in response to this force.

5.3.2 Molecular Dynamics

Molecular dynamics (MD) simulations of maltodextrin binding protein (MBP), Glutamine Binding Protein (GlnBP) and B-Raf were started from the structures deposited in the Protein Data Bank (MBP: PDB ID 1OMP[125]); B-Raf: Chain A of 1UWH[149]; GlnBP: Chain A of 1GGG[53]). AMBER ff14SB[86] was used for the proteins. Each protein was initially solvated in a cubic box with SPC/Eb water molecules[140], and Na+ (MBP and GlnBP) or Cl- (B-Raf) ions[62] to neutralize the systems. The simulation boxes were constructed with a margin of at least 10 Å from the proteins to the periodic box boundaries. The total number of atoms included the simulated systems for MBP, B-Raf and GlnBP, including solvent molecules were 80,019, 52,917 and 56,741, respectively. The simulation was conducted with the pmemd.cuda module[42] of AMBER16[22]. The electrostatic interactions were treated with particle mesh Ewald method[34] and the real space cutoff distance was 10 Å. In all cases, after 200 step energy minimization with positional restraints imposed on experimentally-determined heavy atoms with a force constant of 1 kcal/molÅ[149], system was equilibrated at 300 K and 1 atm with weaker positional restraints with a force constant of 0.1 kcal/molÅ[149]. MD simulations without the restraints were performed for 120 ns and the last 100 ns trajectories used for the analysis with the linear response calculation.

5.3.3 Linear Response

To use LRT to calculate the receptor's deformation, induced by interaction forces from the ligand, the equation:

$$\boldsymbol{\Delta} r = \frac{1}{k_b T} \boldsymbol{A} F \tag{5.3}$$

(equivalent to Equation 3 within Ikeguchi et al.[59]) can be used. To do so, calculation of the variance-covariance matrix of atomic fluctuations within the MD trajectory is required. Taking the coordinate trajectory of the protein from the MD simulation, mass-weighted least-squares best fitting is used to superimpose each frame onto the reference structure, which was the starting structure of the MD simulation. The average structure is then calculated, and the $3N \times 3N$ variance-covariance matrix of atomic fluctuations, $\boldsymbol{A}$.

In Equation 5.3, k_b is Boltzmann's constant, T, the absolute temperature, and $\boldsymbol{\Delta r} = (\Delta x_1\ \Delta y_1\ \Delta z_1 ...\ \Delta x_N\ \Delta y_N\ \Delta z_N)^t$, a column vector of atomic coordinate displacements. Within the quasi-harmonic approximation, displacements are from the average, but as this is not a viable structure, the structure from the trajectory which has the smallest RMSD to the average is used as the initial undeformed structure within the haptic docking session.

Given a receptor and ligand, the force on each receptor atom is calculated using Equation 5.1, and then Equation 5.3 is used to calculate the atomic displacements in the receptor. If the user is allowed to control the position of the ligand, via keyboard or haptic device, and the above calculations are performed continually, with the receptor's deformation updated in real time, the result will be an interactive molecular docking system that supports realistic receptor flexibility. Note that due to the fitting procedure used to calculate $\boldsymbol{A}$, whereby global translational and rotational movements are removed, forces applied to the receptor do not result in its global translation or rotation.

Reducing the dimensionality of protein fluctuation

Equation 5.3 could be used to calculate the deformation, however, matrix $\boldsymbol{A}$ is large - $3Nx3N$ in size, resulting in $9N^2$ calculations needed to evaluate Equation 5.3. Even on a modern GPU, this is unlikely to be possible within the haptic time limit of 2 ms. Furthermore, matrix $\boldsymbol{A}$ will consume a large amount of video memory, which is a scarce resource when compared with main system RAM.

These issues can be overcome by making use of the fact that a large amount of the total fluctuation of a protein occurs within a small subspace (sometimes this space is referred to as the "important subspace" or the modes spanning it as "essential modes"[2,48,70]), to reduce both the number of calculations required per frame, and the memory consumption of the algorithms. This is achieved with the function

$$\Delta r \approx \Delta r_m = \frac{1}{k_b T}(\boldsymbol{V_m}(\boldsymbol{\Lambda_m}(\boldsymbol{V_m^t} F))), \tag{5.4}$$

which is equivalent to Equation 4 in Ikeguchi et al.[59]

In Equation 5.4, $\boldsymbol{V_m}$, is a $3N \times m$ matrix of eigenvectors of the covariance matrix $\boldsymbol{A}$ and $\boldsymbol{\Lambda_m}$ is an $m \times m$ diagonal matrix of the m corresponding eigenvalues arranged in descending order. The fact that the $\boldsymbol{\Lambda_m}$ is a diagonal matrix further reduces the number of multiplications required to calculate the deformation from Equation 5.4. The matrix multiplications can be performed in any order, however, the parentheses are included to show the most efficient way to perform the calculation, i.e. from right to left.

By using Equation 5.4 instead of Equation 5.3 to calculate the receptor's response, and by performing the calculation in the order shown by the parentheses, the number of multiplications is required is reduced to $m(6N+1)$. Furthermore, when multiplying the matrices together in this order, the largest transient matrix will be $3N \times 1$ in size (although the largest matrix is the matrix of eigenvectors, $\boldsymbol{V_m}$, which is $3N \times m$ in size). This means for an appropriate choice of m, use of Equation 5.4 can produce significant savings in memory and execution time over use of Equation 5.3.

Implementation - Deformation calculation

The deformation calculation must be performed in the smallest amount of time possible. In this section the algorithm used to solve the equation in less than 2 ms is discussed. The proposed function is memory-bound, because few computations are performed per memory transaction, therefore care must be taken to coalesce memory accesses in order fully utilise the GPU. Also, in some cases, calculations will be performed multiple times, as opposed to loading additional values from memory or communicating data between threads. The algorithm to solve Equation 5.4 is outlined in Figure 5.6.

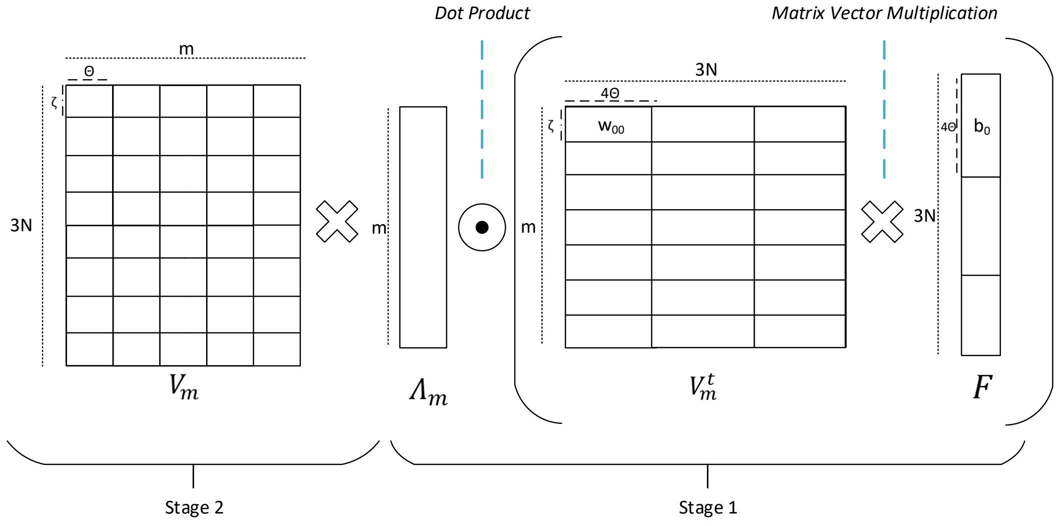


Fig. 5.6 Diagram showing how Equation 5.4 is calculated within Haptimol FlexiDock. Within the diagram, m is the number of eigenvalues in use, N is the number of atoms within the receptor, $\boldsymbol{V_m}$ is a matrix of Eigenvectors, $\boldsymbol{V_m^t}$ is its transpose, $\boldsymbol{\Lambda_m^v}$ is the diagonal of $\boldsymbol{\Lambda_m}$, θ is the number of threads per work group, w_{00} is a subdivision of the Eigenvector matrix, and indicates the area solved by a single thread work group, b_0 is a subdivision of the force vector ζ is the number of rows of the eigenvector matrices handled by a work group. The equation is solved from right to left.

The multiplication shown in Figure 5.6 is performed from right to left. The matrix $\boldsymbol{V_m^t}$ is divided up into work blocks, each of which has a work group of threads assigned to it. The dimension of each block is four times the number of threads (θ) in the work group, multiplied by ζ; 64 was found to be a performant value for ζ. Each thread within a work group is responsible for solving a 4xζ section of the first matrix. Figure 5.7 shows w_{00} being multiplied by b_0. The diagram shows how each block is divided up. Thread 0 is responsible for columns 0-3, and rows 0 to $(\zeta - 1)$ of $\boldsymbol{V_m^t}$.

Within Stage 1, using both row and thread 0 in this example, thread 0 will perform an element-wise multiplication of $[w_0\ w_1\ w_2\ w_3]$ with $[f_0\ f_1\ f_2\ f_3]$ and then sum the result. When all of the threads in the work group have performed their respective multiplications, a work group summation is performed, giving a total for that row of the work block. This row subtotal is then multiplied with the relevant eigenvalue, and then atomically added to an intermediary accumulation array ready for Stage 2. This process is repeated ζ times per work group. Enough work groups are launched

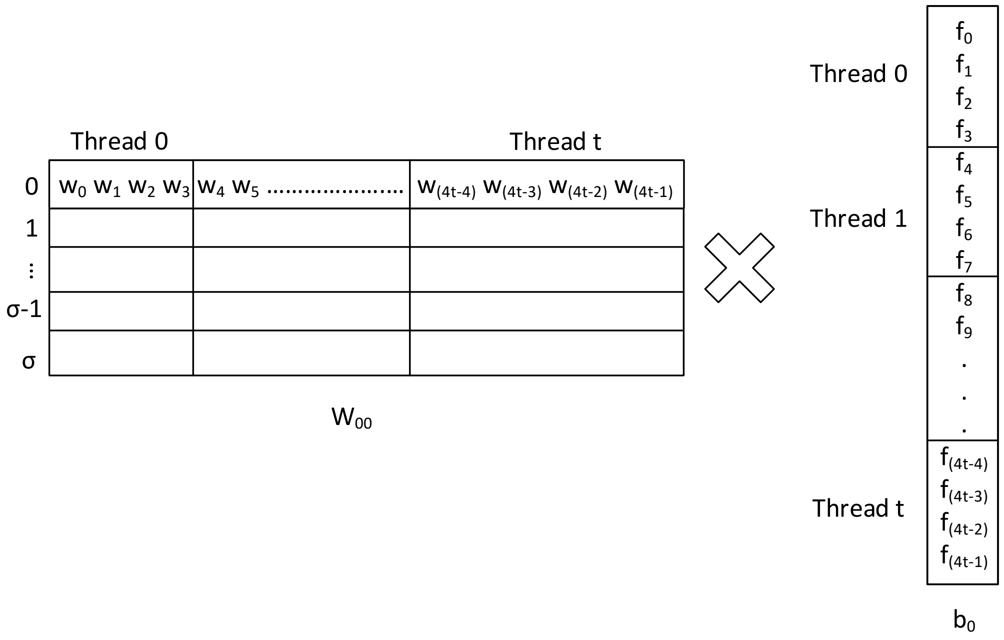


Fig. 5.7 Figure demonstrating how the work assigned to work group W_{00} is divided between threads.

to process the entire matrix. It is important to note that $\boldsymbol{\Lambda_m}$ is the diagonal of a square matrix, so each row needs only an element wise multiplication with the relevant eigenvalue, rather than a full vector/matrix multiplication. This reduces the complexity of the calculation.

When all of the work-blocks employed within Stage 1 have completed, Stage 2 can begin. Again, the matrix is divided up into work-blocks, this time θ x ζ in size. Within the second stage, each thread handles a column vector of four rows from $\boldsymbol{V_m}$, and multiplies them by a single value, taken from the intermediary vector. Although this increases the number of work blocks required to complete the multiplication, it allows for greater efficiency when m is small, leading to better performance overall. The eigenvector matrix is stored within constant memory, in both its original and transpose state, with the elements each thread is responsible for grouped into `float4` memory blocks to ensure maximum utilisation of the cards memory bus.

Finding a stable state

After calculating the deformation, both the haptic and visual scenes need updating. The haptic and visual rendering tasks run in independent threads, to ensure that the visual rendering of the protein does not impact upon the haptic refresh rate. As the haptic loop completes much faster than the visual loop, it makes little sense to update the atom's locations every time they are changed as a result of an interaction force. Therefore, every time the visual thread completes a loop it requests new atom positions from the haptic thread. The updated positions are then copied into a separate buffer, for use by the visual thread. This ensures that the force calculation always has up to date atom positions and the visual rendering of the protein will always be a complete snapshot of a protein; conformations will not get muddled together.

An iterative approach is used to apply translations to the receptor atoms. This approach was taken because in certain binding positions the interaction forces between the receptor and ligand become unstable. In these positions, the initial interaction force between the receptor and ligand is such that a conformational change large enough to noticeably change the interaction force occurs, which in turn alters the receptor's conformation again. If these conformational changes do not converge to an equilibrium, instability occurs. This is illustrated in Figure 5.8.

Within Figure 5.8, the initial force, labelled f_s, causes the atoms in the receptor to move to Δr_i. Because of this, the interaction force changes to f_i, causing the atoms to change position again. In the scenario depicted in the figure the forces and atomic displacements get larger and larger, never reaching a stable state. When using the software in combination with a haptic device, this scenario creates severe vibration.

In order to prevent this from occurring, an iterative approach is used. Rather than apply the entire receptor deformation in one step, a small amount, μ, of the total calculated deformation is applied, and then the force is recomputed. In this fashion, the force and receptor response will eventually reach a stable state. This is depicted as the stepped line from Δr_s to Δr_e in Figure 5.8.

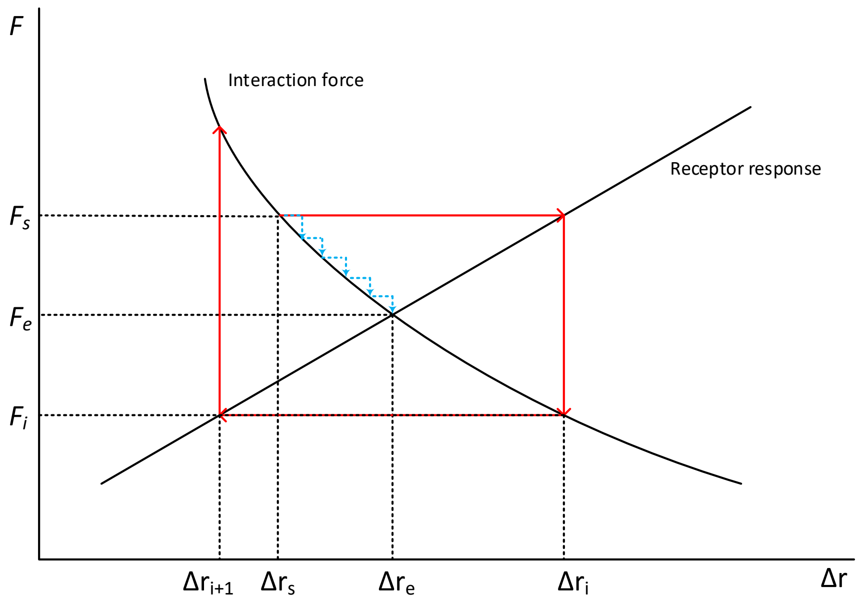


Fig. 5.8 Diagram depicting how using an iterative approach when applying deformation to the receptor improves stability. The initial ligand receptor interaction occurs at Δr_s. The dotted arrows describe the subsequent sequence of events with the iterative approach, the solid arrows without it. The solid arrows show that the initial interaction force, f_s, causes the receptor to deform to Δr_i, which in turn changes the interaction force to force f_i, prompting a different response from the receptor. This cycle can continue ad infinitum. The iterative approach breaks the deformation application up into smaller steps, re-computing the force after each step. In this way, the interaction force and deformation response will come into equilibrium. The global positions of the receptor and ligand are assumed constant throughout the process.

Deforming the receptor in an iterative manner has further usability advantages. Firstly, as the deformation happens more slowly, the receptor's internal movement becomes visible to the user, providing opportunity to learn about the deformation process. Furthermore, the reduced deformation speed gives the user opportunity to respond to tactile and visual cues generated as the protein deforms.

Collision Detection

Detecting collisions between the receptor and ligand is important for the usability of the application. If an atom within the ligand is allowed to significantly overlap with one in the receptor, an enormous, unrealistic force will occur. Therefore, collision detection happens twice per loop: at Stages 2 and 6. At Stage 2, collision detection is performed to determine whether or not the user has moved the ligand into a state where it overlaps with the receptor. At Stage 6, it is performed after the deformation has been applied to the receptor, to determine if it has deformed such that it overlaps with the ligand. When using the haptic device, the collision detection performed at Stage 2 is less important, as the user will feel a large repulsive force, before a significant overlap occurs. As collision detection is performed twice within the loop, it is important that it takes the smallest possible amount of time. Algorithm 10 formally describes the method used.

The first stage of the algorithm is performed on the GPU. Each atom in the receptor tests all of the atoms in the ligand, to see if they are closer in space than the sum of the atom pair's radii. If they are, the *collisions buffer* is incremented. When this stage is complete, the collisions buffer is copied back to the CPU, and a final total calculated. If the final total is larger than 0, one of the atoms is in collision.

This approach is suitable for use when the ligand is small. If it were larger, a spatial partitioning structure may be needed to reduce the number of calculations performed.

Stopping the deformation at first contact between the receptor and ligand was found to make it very difficult to perform docking with any significant receptor movement,

Algorithm 10 moleculeCollisionDetection(r, l,n_r,n_l) **return** *workGroupCollisions*

```
Require: $r$ {Receptor atom information }
Require: $l$ {Ligand atom information.}
Require: $n_r, n_l$ {number of atoms within the receptor ($n_r$ and ligand $n_l$}
Require: $groupId$ {Work group identifier}
1:  $collisions = (0)$
2:  for all $i \leftarrow 1$ to $n_r$ in parallel
3:      for $j \leftarrow 1$ to $n_l$ do
4:          $minDist \leftarrow r_i$.radius - $l_j$.radius
5:          $distVec \leftarrow r_i$.xyz - $l_j$.xyz
6:          $distSqr \leftarrow$ sum($distVec. * distVec$)
7:          $collisions = 0$
8:          if $(minDist^2) > distSqr$ then $collisions += 1$
9:          end if
10:     end for
11: end for
12: $workGroupCollisions(groupId) \leftarrow$ parallelSummation($collisions$)
```

as it often prevented the receptor from deforming away from the ligand. Therefore a small amount of overlap, which is customisable at runtime, is allowed between the ligand and receptor atoms. This allows the receptor to deform away from the ligand where it would naturally do so.

The drawback of allowing any overlap is large forces can be generated, especially if the user forces the ligand through repulsive forces. Therefore, the option to stop deformation when a large force occurs is also available to the user.

Keyboard controlled docking

Haptimol FlexiDock renders proteins in space-filling mode, using CPK colouring[73] by default. The colour of individual residues can be changed by the user. Although the application is designed for use with a haptic device, this may not always be feasible, owing to the comparative expense of haptic devices when compared with conventional input devices.

In order to improve the usability of the software when used without a haptic device, a colour system is provided that varies the brightness of individual receptor atoms in

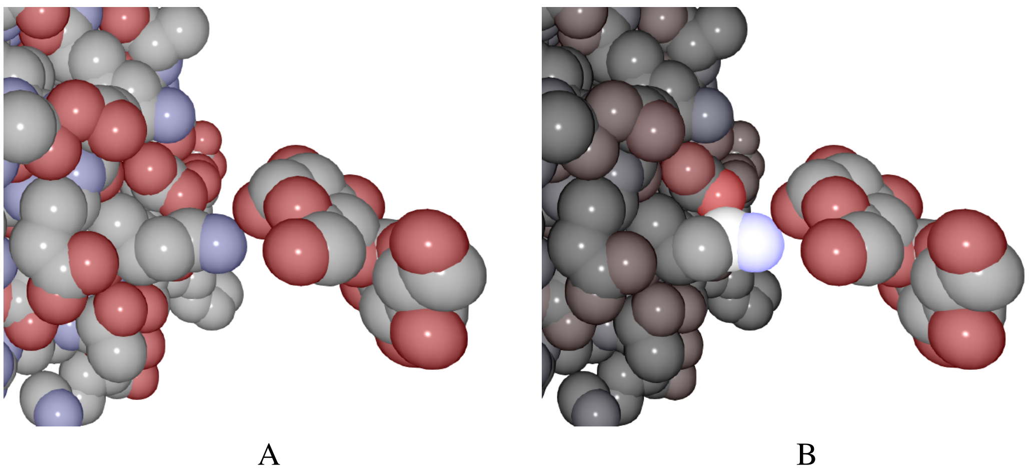


Fig. 5.9 A receptor-ligand pose rendered in Haptimol FlexiDock with (A) ordinary colour and (B) force-contribution colouring.

order to highlight those which are contributing the most to the total interaction force. The effect can be seen in Figure 5.9.

Besides the force-interaction colouring, the interaction energy and forces can be plotted on a graph in real time, allowing the user some understanding of the force interactions that are prompting the rendered response. These features are also available when using a haptic device.

5.4 Method Analysis

There are two main aspects of the described approach that need to be analysed to determine if the algorithms proposed in this section successfully model protein flexibility rapidly enough to be used with a haptic device. These aspects are the performance of the application, and how well the proposed methods model ligand flexibility.

5.4.1 Application Performance

As stated in the introduction, for continuous, smooth and stable force feedback, haptic devices need updating at a refresh rate greater than 500Hz. To ascertain whether the described system meets this criteria, whilst using a sufficient number of eigenvalues to incorporate the important subspace of any tested molecules, ligand movement around the protein is simulated, and the execution time of the haptic loop (Figure 5.2) recorded.

To confirm that reducing the dimensionality of the receptor's deformation is necessary, the time taken to compute Equation 5.3 was measured, using the highly optimised cuBLAS library. The execution time of the deformation calculation, Equation 5.4 is also measured for comparison. Owing to the difficulty in accurately timing the execution time of methods, because processor exclusivity cannot be guaranteed, a large number of samples will need to be taken for all of the timing experiments.

5.4.2 Deformation Testing

In order to determine how much of the total molecular fluctuation is contained within the first m modes, the equation

$$\rho(m) = 100\frac{\sum_{i=1}^{m}\lambda_i}{\sum_{i=1}^{3N-6}\lambda_i} \tag{5.5}$$

can be used. $\rho(m)$ in Equation 5.5 measures the percentage of the total fluctuation that the protein undergoes in its MD trajectory, that is contained within the first m eigenvectors.

Where λ_i is the ith eigenvalue of $\mathbf{\Lambda_m}$, $3N-6$ is the total number of non-zero eigenvalues and $\rho(m)$ is the percentage fluctuation contained within the first m eigenvalues. This formula will be used to determine if a sufficient proportion of the total fluctuation is represented by using m eigenvalues within a docking session. When $m = 3N-6$, $\rho(m) = 100\%$.

To assess how well the system models the receptor's functional movement, experimentally derived ligand-bound and unbound structures will be compared with poses generated within Haptimol FlexiDock.

Before performing these comparisons it is important to know how much of the experimentally derived movement is contained within the high fluctuation eigenvectors. This is measured in the equation

$$\tau(m) = 100\sum_{i=1}^{m}\left(\boldsymbol{v}_i^t\left(\frac{\boldsymbol{\Delta r}_{exp}}{|\boldsymbol{\Delta r}_{exp}|}\right)\right)^2, \tag{5.6}$$

where $\boldsymbol{v}_i$ is the ith column of $\boldsymbol{V_m}$, and $\boldsymbol{\Delta r}_{exp}$ is a column vector of the experimentally derived individual atomic coordinate displacements - the movement required to displace each atom from its position in the ligand free structure to its position in the ligand bound structure, after global superposition. $\tau(m)$ indicates the percentage of the experimental movement that is represented in the space of the first m eigenvectors. When $m = 3N-6$, $\tau(m) = 100\%$.

There are various methods that can be used to compare a docked pose generated within Haptimol FlexiDock to the experimentally derived ligand-bound structure. The most straightforward of these is the root-mean-square deviation of atomic positions (RMSD).

In addition, the atomic displacements that occur upon ligand docking during the haptic session can be compared with the experimentally derived displacements.

In equation

$$p = \frac{\boldsymbol{\Delta r}^t_{hap\_bb}\boldsymbol{\Delta r}_{exp\_bb}}{|\boldsymbol{\Delta r}_{hap\_bb}||\boldsymbol{\Delta r}_{exp\_bb}|}, \tag{5.7}$$

where $\boldsymbol{\Delta r}_{hap\_bb}$ is a column vector of backbone atom coordinate displacements from the starting to bound conformations taken from a haptic docking session and $\boldsymbol{\Delta r}_{exp\_bb}$ is the experimentally derived individual backbone atom displacements, p indicates how well the direction of the experimentally derived and haptic movements align; the closer p is to 1.0, the better the alignment. When calculating $\boldsymbol{\Delta r}_{hap\_bb}$ the start structure and haptic docked structure must be globally aligned. Equation 5.7 does not consider the relative magnitudes of the displacements, which is quantified by q in

$$q = \frac{\boldsymbol{\Delta r}^{t}_{hap\_bb}\boldsymbol{\Delta r}_{exp\_bb}}{|\boldsymbol{\Delta r}_{exp\_bb}|^2}. \tag{5.8}$$

Again, only the molecule's backbone is considered when using this equation. Combined, Equations 5.7 and 5.8 can be used to determine how close a docked 'haptic' structure is to the experimentally derived ligand-bound structure; the closer both values are to 1.0, the closer the movements match.

The distribution of displacements of C^{α} atoms are also compared, as was previously done by Ikeguchi et al.[59]. A final metric is ΔL which indicates the distance between the ligand in its position in the haptic docked pose and the experimental pose. As the orientation of the haptic-controlled ligand relative to the protein is the same as in the experimental structure of the protein-ligand complex, this is simply evaluated by measuring the distance between any atom of the ligand in its two positions.

Together these metrics will give a good picture of how well the haptic-derived movement compares to the experimentally derived movement.

5.4.3 Testing the Iterative approach

The effectiveness of the iterative approach will also be appraised. To do this, the receptor and ligand are placed into an unstable conformation. Then, a simulation long enough for the conformation to stabilise is run, with the total force between the receptor and ligand recorded at each step. The ligand's and receptor's global positions are held constant throughout. This test is repeated for different values of μ, the step size. If the iterative approach works as intended, when using a small value for μ, the net force should stabilise.

Receptor Name	Receptor PDB Code (Atoms)	Trajectory Length	Ligand Name (Atoms)	Ligand Bound PDB Code
Maltodextrin Binding Protein (MBP)	1OMP (5737)	100 ns	Maltose (40)	1ANF
B-Raf	1UWH (4263)	100 ns	Sorafenib (48)	1UWH
Glutamine Binding Protein	1GGG (3431)	100 ns	Glutamine (20)	1WDN

Table 5.1 Receptor/ligand pairs used within testing.

5.5 Results

The performance testing of the described methods was performed using three different receptor ligand pairs, as shown in Table 5.1. These pairs were chosen because it is known that they all undergo a significant and fairly well understood backbone deformation during ligand binding. Furthermore, there are ligand bound structures available on the RSCB[14] for each of the pairs, which can be used in order to verify the output of the system.

The topology for MBP was generated using PDB2GMX[131], The other topologies were generated with AmberTools16[22] and converted into the GROMACS format required by FlexiDock using ParmEd[139]. The ligand unbound version of B-Raf was generated by removing sorafenib from the bound version. All of the testing was performed on a desktop computer with an Intel i7 processor and an NVIDIA GTX 1080.

Multiplying the entire covariance matrix

To demonstrate the necessity of the dimensionality reduction technique, the execution time of Equation 5.3 was recorded. The multiplication was performed with smaller matrices in order to determine the upper limit of the number of atoms at which multiplying the entire covariance matrix becomes infeasible, along with the covariance matrices of the proteins in Table 5.1. Figure 5.10 shows the results.

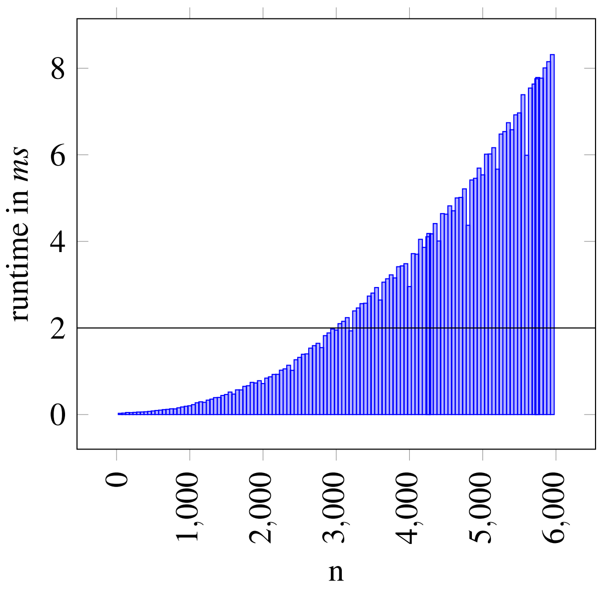


Fig. 5.10 Vector-Matrix multiplication on the GPU. Matrices up to approximately 3000×3000 can be multiplied with a vector in less than 2 ms.

The graph shows that the conformational change resulting from a force can be calculated with the entire covariance matrix in under 2 ms, when the protein comprises 3000 atoms or fewer. However, the time limit for the entire haptic loop is 2 ms so in reality, the largest protein this approach is currently viable for will be much smaller. All of our test structures are larger than this. Therefore, the dimensionality reduction technique described in Section 5.3.3 will be used in order to achieve a real-time haptic refresh rate in our application.

Eigenvalue Dimensionality Reduction

To demonstrate the effectiveness of the dimensionality reduction technique, the time taken to calculate the force response using Equation 5.4 was measured for all three of our test proteins, whilst using a varying number of eigenvalues. The results are shown in Figure 5.11.

Figure 5.11 demonstrates the effectiveness of the dimensionality reduction technique. Whilst calculating receptor deformation using the entire covariance matrix is infeasible for our test proteins, it is easily achievable with Equation 5.4, provided an

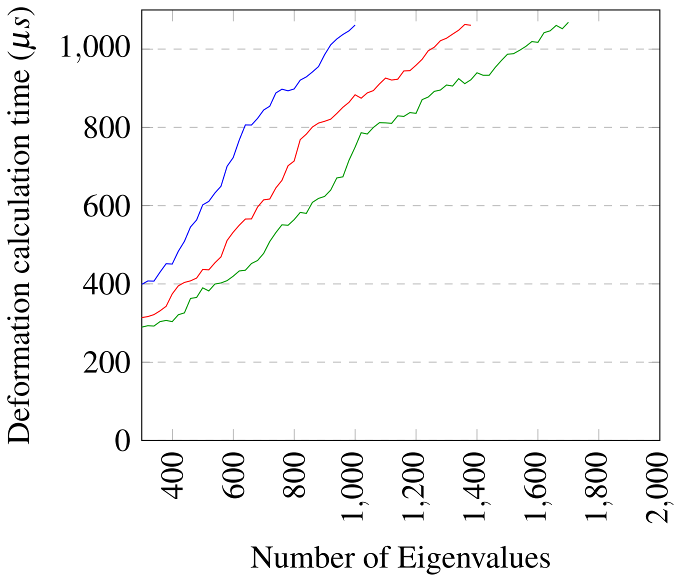


Fig. 5.11 Time taken to calculate the deformation response caused by interactions with the ligand, using Equation 5.4, with a varying number of Eigenvalues. Blue = MBP, Red = B-Raf, Green = GlnBP

appropriate number of modes are used. To determine exactly how many modes can be used, the total execution time of the haptic loop was measured and recorded. The results can be seen in the lower graphs within Figure 5.12.

The results show a fairly large standard deviation (approximately 300 μs). This can largely be explained by the fact that during the haptic loop, two paths are taken; if there is a collision, an extra two steps are performed, adding to the runtime, however as processor exclusivity during testing cannot be guaranteed: other processes maybe claiming processor time, elongating the runtime of random loops.

To ensure smooth feedback from the haptic device, it is important that the majority of haptic loops complete in under 2 ms. By looking at the second standard deviation of the run time data, the dotted line in the lower graphs in Figure 5.12, it can be seen that for MBP, using fewer than 550 Eigenvalues (out of a total of 17208 non-zero eigenvalues), for B-Raf, fewer than 760 (out of 12786 non-zero eigenvalues) and for GlnBP fewer than 1220 eigenvalues (21.4% of the 10287 non-zero eigenvectors), 97.7% of the calculations are completed in less than 2 ms, meeting this goal.

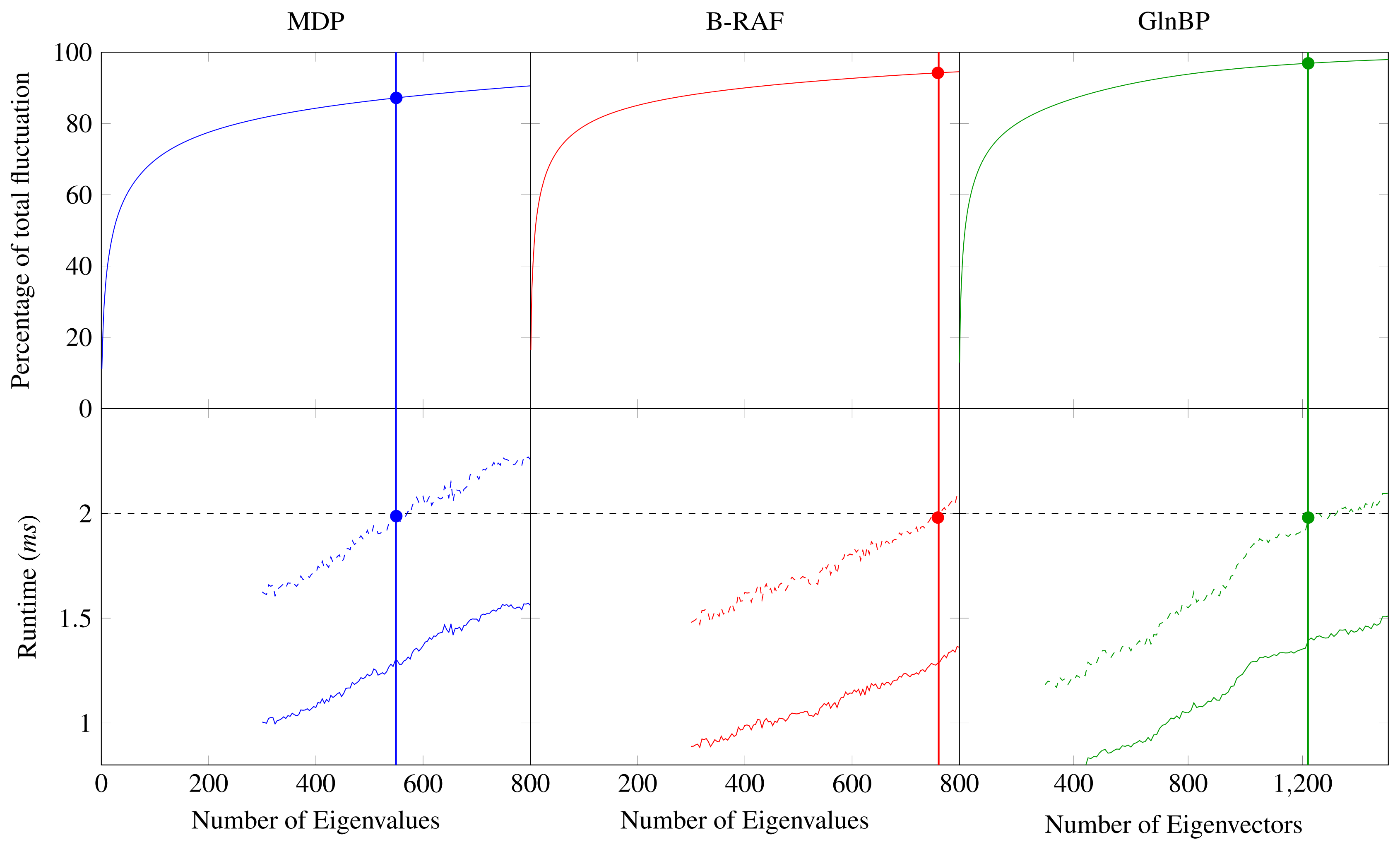


Fig. 5.12 shows the time taken to complete the haptic loop (lower row) and the percentage of the total fluctuation incorporated within the corresponding number of eigenvectors (upper row) for the test proteins MBP (left column) and B-Raf (central column) and GlnBP (right column). The vertical line indicates the number of eigenvectors for which the 2 ms constraint can be satisfied 98% of the time. The graphs show that sufficient eigenvalues to incorporate the important subspace can be included within the deformation calculation whilst achieving the haptic-imposed time limit of 2,000 μs, for our test proteins. The steep region of the upper graph indicates the important subspace.

To determine whether these values are sufficient to model the protein's movements, Equation 5.5 can be used. The results for different eigenvalue counts can be seen in the upper plots of Figure 5.12, which shows the plot of $\rho(m)$ for the three test proteins.

Figure 5.12 shows that for MBP, $\rho(550) = 87\%$, B-Raf, $\rho(760) = 94\%$ and GlnBP $\rho(1220) = 97\%$. Thus despite only being able to use a relatively small number of eigenvectors to calculate the deformation during an interactive session, they are sufficient to span the important subspace.

In the upper parts of Figure 5.12, the important subspace is considered to be the steep portion of the graph, comprising the first few eigenvalues. The results from the timing experiments show that for all tested proteins, the important subspace is included within the number of modes achievable within the 2 ms time constraint. Therefore, during a docking session the receptor should undergo its functional movement, provided the ligand is in the correct position.

Functional movement and space explored in MD trajectory

In this section, docking experiments will be performed in order to determine if the deformation that a protein undergoes as a response to ligand docking, as seen in the crystallographic structures, occurs when performing docking with the methods described in this chapter.

The haptic starting and the experimental docked structures were generated, or solved, using X-Ray crystallography. In order to do this, a crystal containing the structure needs to be created. This is achieved using a process known as protein crystallization[92]. Crystallographers grow these crystals by slow, controlled precipitation from an aqueous solution under conditions that do not denature, or damage, the protein. A simple means of achieving this is to add a precipitant to an aqueous solution of the protein, until its concentration is just below that required to precipitate the protein. Then, the water is allowed to slowly evaporate, which gently increases the concentration of the protein and precipitant until precipitation occurs[117].

At this point, protein crystals can form, but it is not guaranteed; rather a useless solid could form. Identifying the ideal conditions for crystals to form can take many trials, and is as much an art as a science[117].

After a crystal has been successfully formed, a beam of x-rays is fired at it, which then diffract into many specific directions. By identifying the angles and intensities of these diffracted beams, a three dimensional picture of the electron density within the crystal can be constructed. From this, it is possible to determine the average positions of the atoms within the crystal, as well as their chemical bonds[117].

In this project, we use structures made available by other research groups. In the case of 1GGG and 1OMP, ligand bound and ligand unbound structures are freely available. Comparing these bound and unbound structures provides a picture of how the unbound structure deforms into the bound structure upon ligand binding, and gives a good approximation of the location of the ligand binding site. In the case of 1UWH, no ligand unbound structure was available, making it impossible to perform docking experiments with it, as its topology in an unbound state, is not known. Before the MD simulation was performed, sorafenib was removed from the bound structure of 1UWH, in the expectation that the bound structure would relax to an open structure. However, B-Raf remained largely closed making it impossible to dock sorafenib within FlexiDock. Therefore, during deformation testing, the analysis is limited to performing docking with MBP and maltose, as well as GlnBP and glutamine.

In the previous section, it was determined that it is feasible to use 550 eigenvalues whilst performing docking between maltose and MBP, and 1220 eigenvalues whilst performing docking between glutamine and GlnBP. In order to find out whether the functional movement, as derived from the crystallographic ligand-free and ligand-bound structures, is contained within the high-fluctuation modes from the MD trajectory. Equation 5.6 was used. The results show that for MBP $\tau(550)$ = 90.3% of the experimentally derived functional movement is incorporated within the first 550 eigenvectors, and for GlnBP, $\tau(1220)$ = 94.7% of the motion is within the first 1220 eigenvalues. This shows that the majority of the experimental movement is contained within the earliest few modes, and therefore, a movement that closely matches the experimental functional movement would be possible even when comparatively few eigenvectors are used.

Haptic vs Experimental

Using a 3D Systems Touch haptic device (formally known as the SensAble Phantom Omni), docking experiments were performed, attempting to dock maltose into MBP and glutamine into GlnBP. The starting pose, or haptic start structure, is the conformation from the trajectory with the lowest RMSD to the trajectory average structure. In these experiments, 500 eigenvectors were used to calculate the flexibility for MBP and 1200 for GlnBP. The ligand was placed into its experimentally determined bound orientation by superimposing the experimental ligand-bound structure onto the haptic start structure, reorienting the ligand accordingly. Then, using the metrics outlined within Section 5.4.2, each of the poses were tested to determine if the calculated movement aligned with that derived experimentally. The results are shown in Table 5.2.

The results show that for both proteins the calculated receptor deformation, when the ligand is docked into the approximate binding area (as determined from the experimental structure), is aligned with the experimentally derived motion. This is shown both with the RMSD values and the values of p and q.

Initially looking at the RMSD values, it is shown that for most of the test poses, the haptic docked structure is ‘closer’ to the experimentally derived bound structure than it is to its starting structure, showing that the deformation modelled has deformed the structure toward its ligand-bound pose.

The given values for p and q further validate that the calculated movement compares well with the experimental movement. For all of the reported poses $q > 0.8$, in order for the starting pose to deform into the experimental ligand-bound pose. Fur-

MBP

PoseID	ΔL (Å)	Start RMSD (Å)	Docked RMSD (Å)	p	q
H-MBP1	2.9	2.39	2.23	0.85	0.58
H-MBP2	3.0	2.91	1.99	0.85	0.67
H-MBP3	2.6	2.68	2.11	0.83	0.63
H-MBP4	3.3	2.30	2.47	0.83	0.53
H-MBP5	3.2	2.64	2.25	0.84	0.63

GlnBP

PoseID	ΔL (Å)	Start RMSD (Å)	Docked RMSD (Å)	p	q
H-GlnBP1	1.5	3.36	3.31	0.85	0.50
H-GlnBP2	3.3	3.43	3.40	0.83	0.49
H-GlnBP3	1.3	4.41	3.09	0.83	0.63
H-GlnBP4	1.6	3.32	3.29	0.86	0.50
H-GlnBP5	1.8	4.34	3.08	0.83	0.62

Table 5.2 Table detailing the quality of the “Haptic” docking poses for MBP and GlnBP. Start RMSD is the RMSD between the haptic docked pose of the receptor and the haptic starting pose. Docked RMSD is the RMSD between the haptic docked pose of the receptor and the experimentally derived docked structure. Both are calculated between C^{α} atoms only. p is the result of using Equation (5.7) and q the result of Equation (5.8). ΔL is the distance between maltose in the haptic pose and the experimental pose.

thermore, all of the poses except H-GlnBP2 exhibit at least half of the intra-molecular backbone movement required to fully displace the atoms from an open position to a closed position. Together, these metrics show that the described system calculates and renders a deformation that is broadly aligned with the experimentally derived movement. Figure 5.13 shows the haptic docked pose superimposed onto its starting pose, and the experimentally determined closed pose for both GlnBP and MBP. Figures 5.13 (C) and (F) show the displacement of the C^{α} that occurs between the starting pose and experimental pose (red) and the starting pose and haptic pose (blue). In both cases, the lines correlate well, further demonstrating that the deformation calculated within Haptimol FlexiDock is a good approximation of the experimentally derived movement.

5.5.1 Reducing the subspace size in Haptimol Flexidock

Figure 5.14 shows $\rho(m)$, (Equation 5.5) and $\tau(m)$ (Equation 5.6) for the first 20 eigenvectors of the test proteins, MBP and GlnBP.

For MBP, Figure 5.14 shows that despite within the first 20 eigenvectors, which is 0.1% of the total number, nearly 50% of the total fluctuation occurs within the space spanned by their corresponding eigenvectors. Furthermore 75% of the experimentally determined movement is contained within this space. The results are similar for GlnBP.

As a consequence of this, using a small number of eigenvectors will still reproduce the majority of the movement seen upon maltose binding from the crystal structures. This will enable Haptimol FlexiDock to work well on less powerful computers, as reducing the number of eigenvectors used will reduce the computational cost of calculating the conformational response.

The lower graph in Figure 5.14 shows that for MBP, 71.6% of the movement that occurs upon ligand binding is incorporated within the first 3 eigenvectors. In order to allow users to study the resultant effect of adding and removing eigenvectors from consideration, the facility to modify m during a docking session is provided.

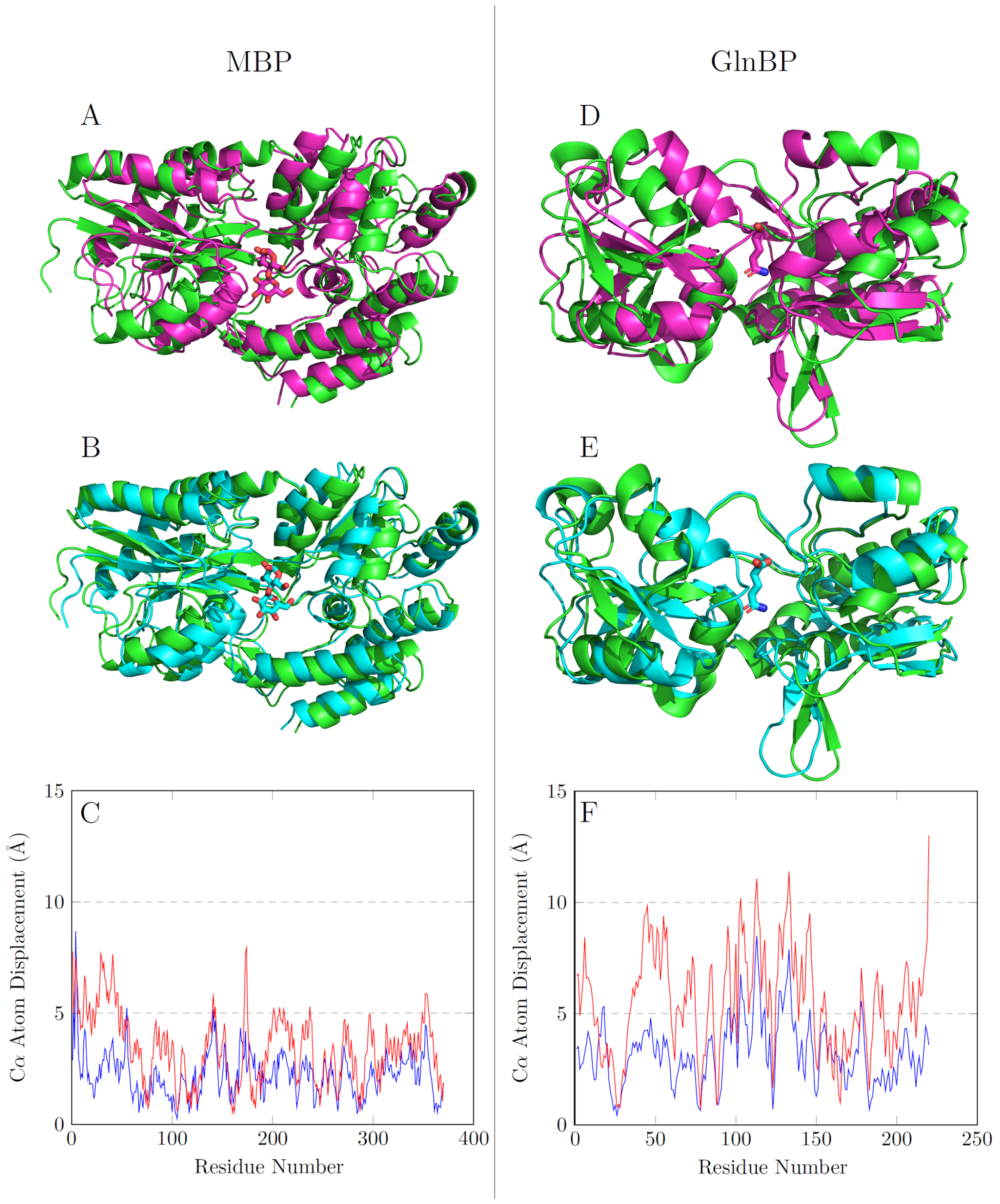


Fig. 5.13 (A) Structure of MBP from an interactive session where maltose is bound close to its binding site (magenta - pose: H-MBP2 - see Table 2) superimposed upon the haptic starting structure of MBP (green). (B) Maltose bound X-ray structure (1ANF - cyan) superimposed on the haptic starting structure (green). (C) Displacements of C^{α} atoms for the Haptic (A - blue) and Experimental (B - red) cases (correlation coefficient is 0.628) for MBP. (D) Structure of GlnBP from an interactive session where glutamine is bound close to its binding site (magenta - pose: H-GlnBP5) superimposed upon the haptic starting structure of GlnBP (green). (E) Glutamine bound X-ray structure (1WDN - cyan) superimposed on the haptic starting structure (green). (F) Displacements of C^{α} atoms for the Haptic (A - blue) and Experimental (B - red) cases (correlation coefficient is 0.761) for GlnBP. (A),(B),(D) and (E) were generated using PyMOL[123].

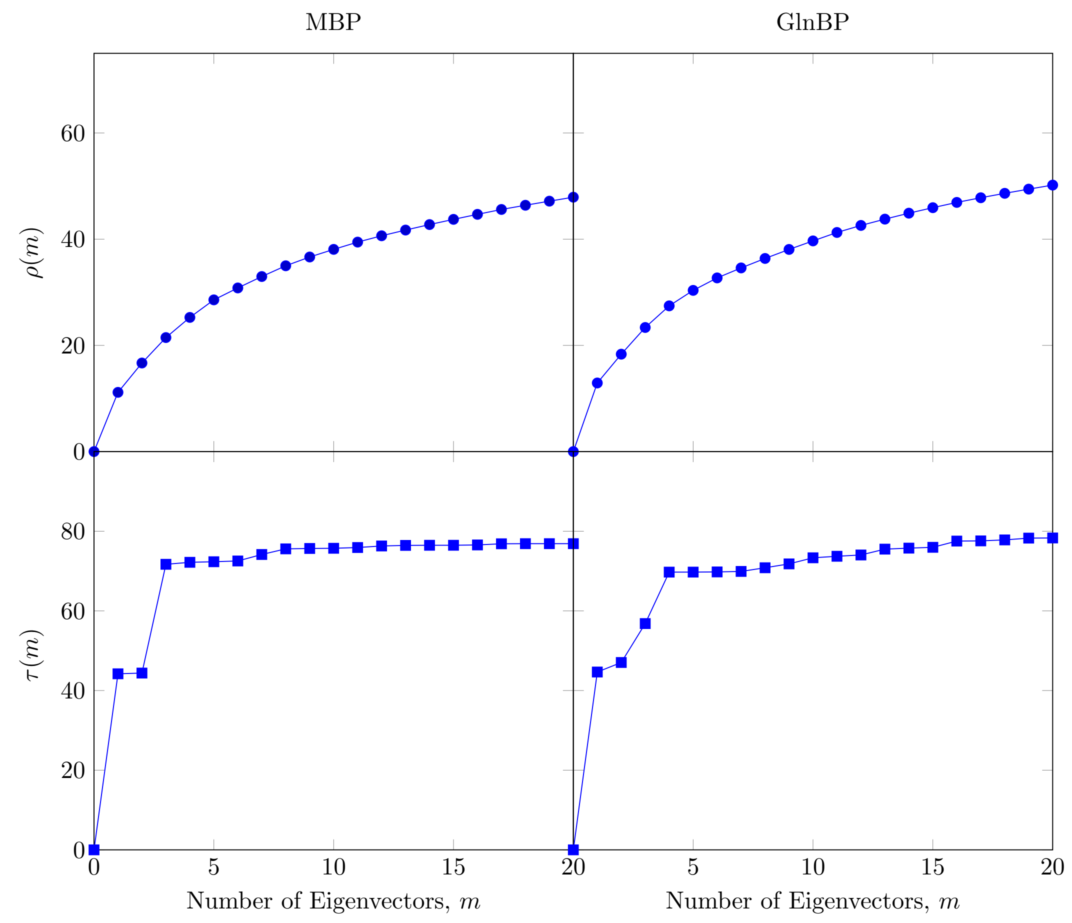


Fig. 5.14 Upper graph shows $\rho(m)$, the percentage of the total fluctuation contained within the space of the first m eigenvectors and $\tau(m)$, the percentage of the experimental movement contained within this space.

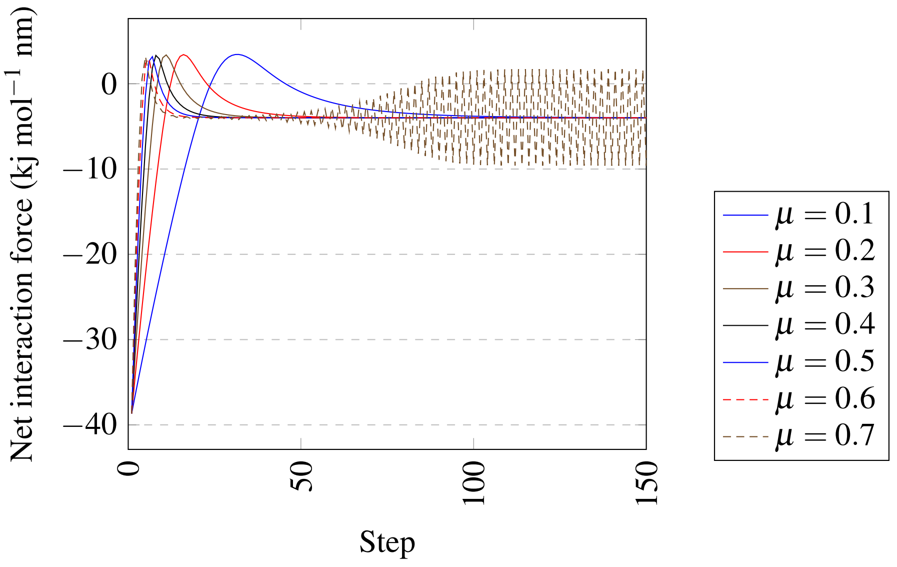


Fig. 5.15 The effectiveness of the iterative approach. With a small value of μ (the step size), the interaction force stabilises, conversely when it is larger, it does not.

Haptimol FlexiDock also incorporates other features, including the option to select and remove side chains from the interaction force calculations whilst still rendering them graphically ("ghostify") and to switch different force components (van der Waals or electrostatic) on and off.

5.5.2 Iterative approach

To demonstrate the effectiveness of the iterative approach, maltose and MBP were placed into an unstable conformation (See Section 5.3.3). A simulation was then allowed to run for 150 iterations. The results can be seen in Figure 5.15.

The results show that when μ, the step size, is small, the forces peak at 3.4 kj mol^{-1} nm^{-1}, before stabilising at -4 kj mol^{-1} nm^{-1}. The smaller the value of lambda, the longer the force takes to settle, as expected. When μ is larger, the forces do not settle, they oscillate between a positive and negative force. This instability is visible in the visual rendering and causes vibration on the haptic device. The stabilisation that occurs when μ is small indicates the iterative approach is working as designed.

5.5.3 Advanced Graphics

Figure 5.12 shows that our software supports up using up to 550 eigenvalues whilst maintaining a haptic refresh rate of at least 500Hz, when performing docking between the protein MBP, and the ligand maltose; it was possible to use far more, over 1200 eigenvectors, when performing docking experiments with the smaller receptor/ligand pair GlnBP and glutamine. However, whilst performing these experiments, the visualisation of the receptor and ligand was basic, incorporating no lighting effects. In Chapter 4, a molecular rendering algorithm that supports ray cast shadows, and per-pixel ambient occlusion was presented, and shown to be efficient enough for use in a real time application. In order to improve the depth perception of the molecules whilst performing docking, the presented algorithm was integrated into Haptimol FlexiDock. Figure 5.16 shows the difference between the original rendering algorithm, as used in HaptimolRD, and the updated rendering algorithm incorporated within Haptimol FlexiDock.

The rendering adds depth to the image, and can help the user to understand where the ligand is, in respect to the receptor. However, the additional load on the graphics processor has a substantial effect on the achieved haptic refresh rate. Figure 5.17 shows the performance achieved during a docking session with advanced graphics enabled.

The results in Figure 5.17 indicate that the haptic loop takes longer than the 2 ms/500 Hz constraint imposed by the haptic device, for both proteins, when run with no constraints on the rendering refresh rate. This, and the fairly large standard deviation, is unsurprising, as the rendering is competing with the force and deformation calculation for GPU resources. By limiting the refresh rate to 30 FPS and therefore reducing how often the scene is rendered, the overall computational footprint of the rendering algorithm is reduced, freeing resources for use by the force and deformation calculation, resulting in a higher average frame rate. On the test system, limiting the refresh rate allows the haptic loop to complete, on average, in less than 2 ms for

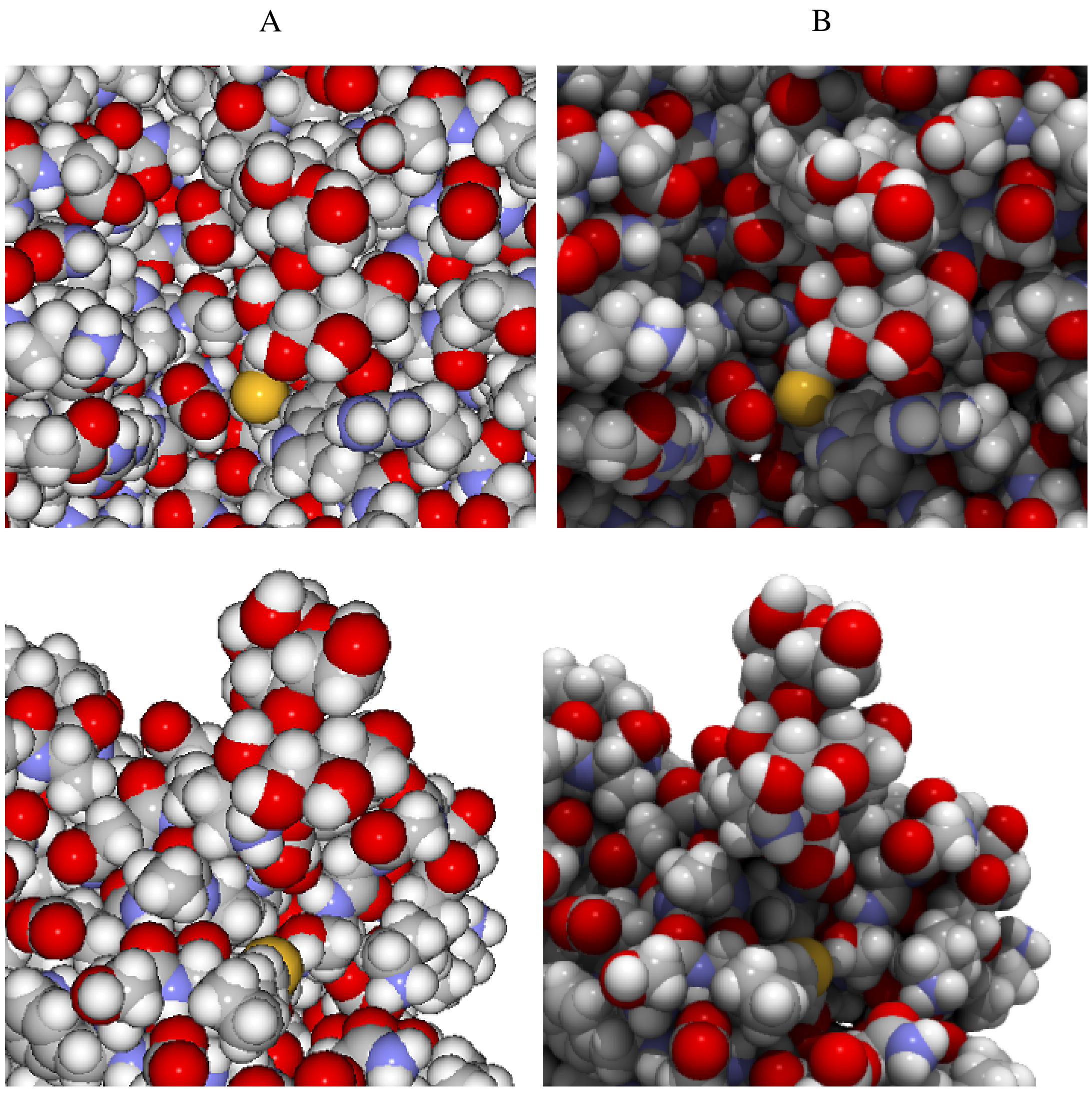


Fig. 5.16 Column (A) Original rendering method, as used within HaptimolRD Iakovou et al.[58], (B) Haptimol FlexiDock with per-pixel lighting.

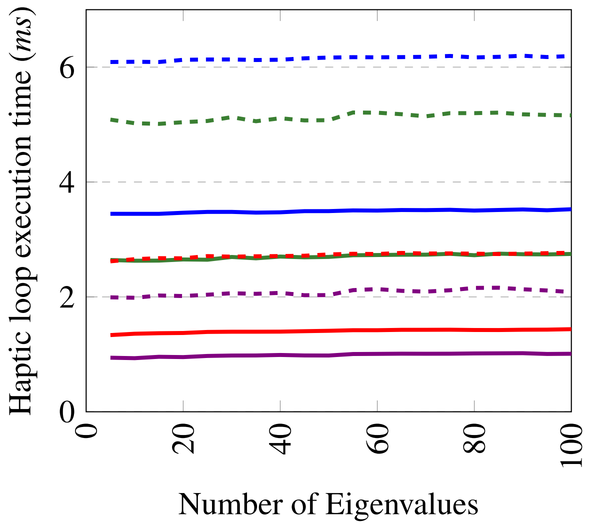
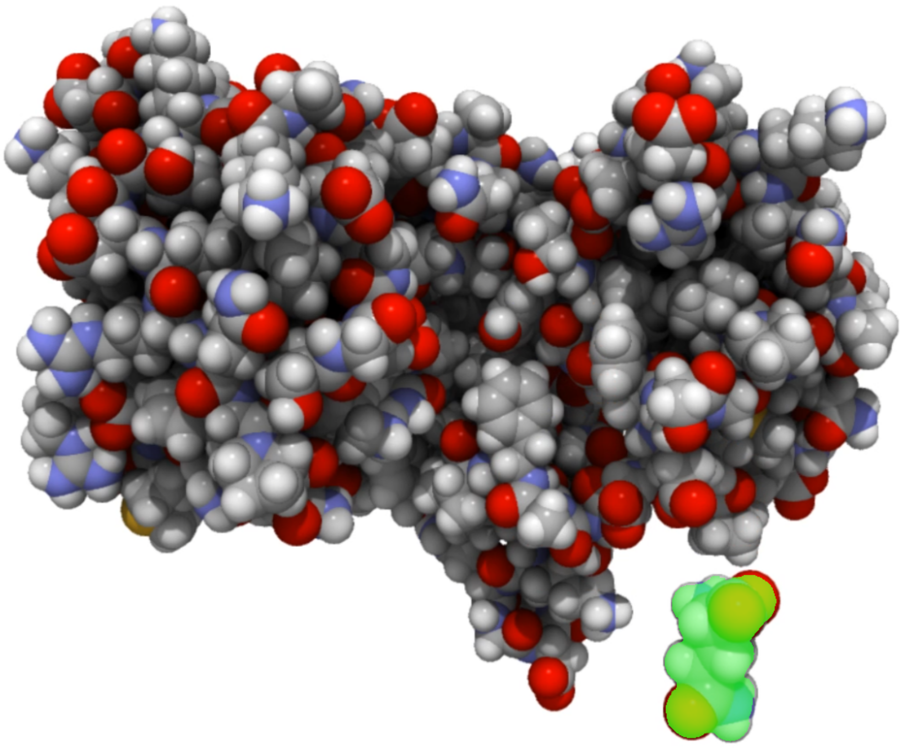


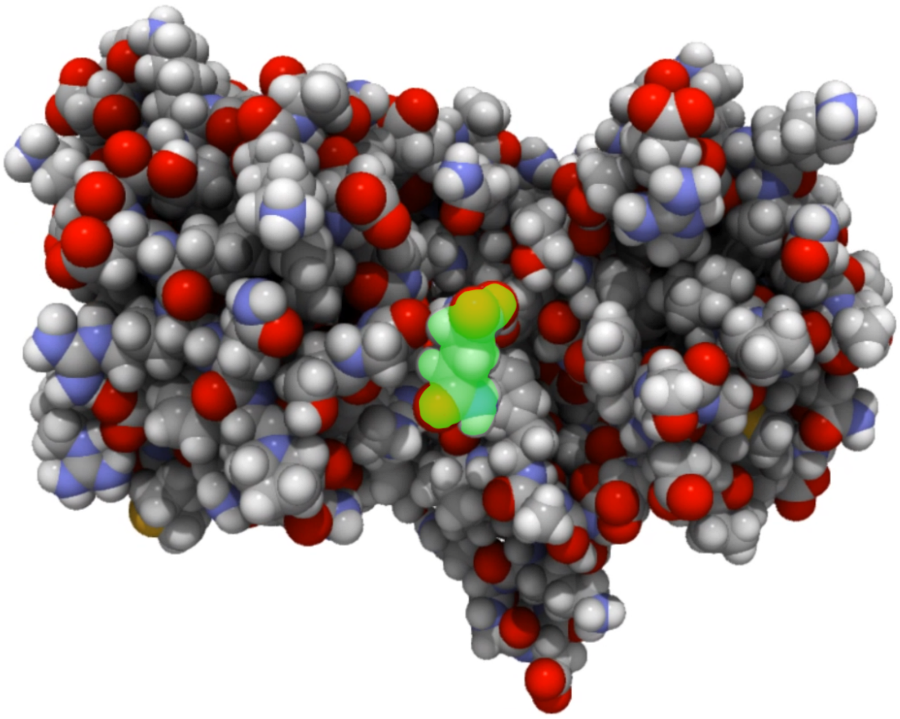
Fig. 5.17 Haptic loop execution time for GlnBP and MBP with advanced graphics enabled. Blue : MBP, red: GlnBP, green: MBP with the visual refresh rate limited to 30 FPS, purple: GlnBP with the refresh rate similarly limited. The dotted lines indicate one standard deviation added to the average runtime.

both proteins. However, when doing so, the standard deviation remains large, as the rendering work still has to be done, even if it is running less frequently. Therefore, some instability may still be felt through the haptic device.

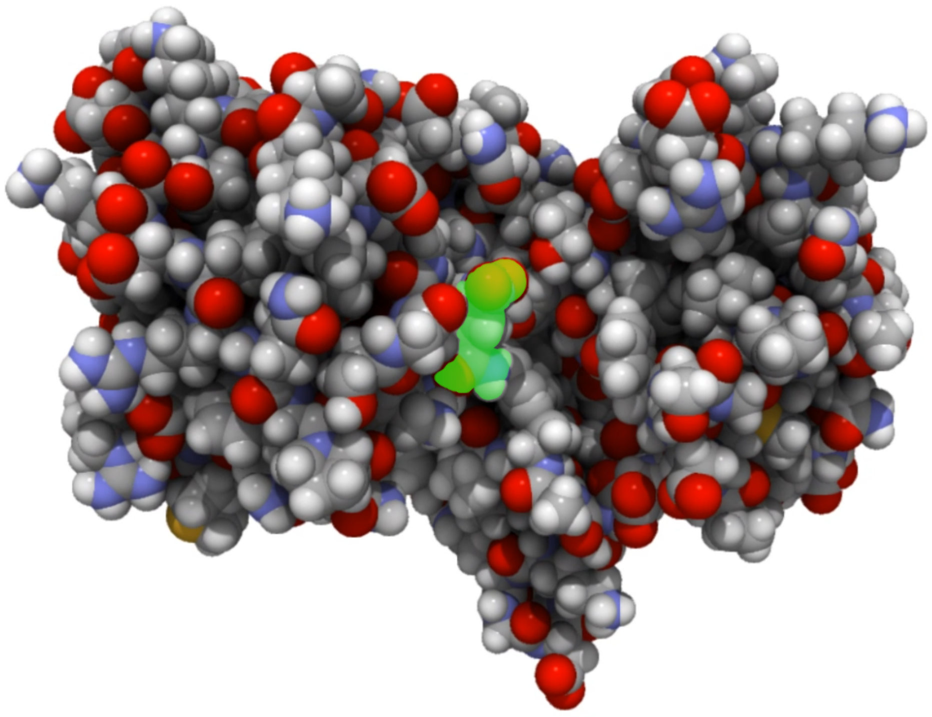
When performing a docking experiment with GlnBP and Glutamine, whilst using 20 modes or fewer, 84% of the haptic loops performed complete in under 2 ms. Therefore, it is still possible to perform a docking experiment with the advanced graphics enabled. Figure 5.17 shows the process of docking the ligand to GlnBP. Between (A) and (E) a closing motion occurs, as glutamine is placed into its approximate binding site.



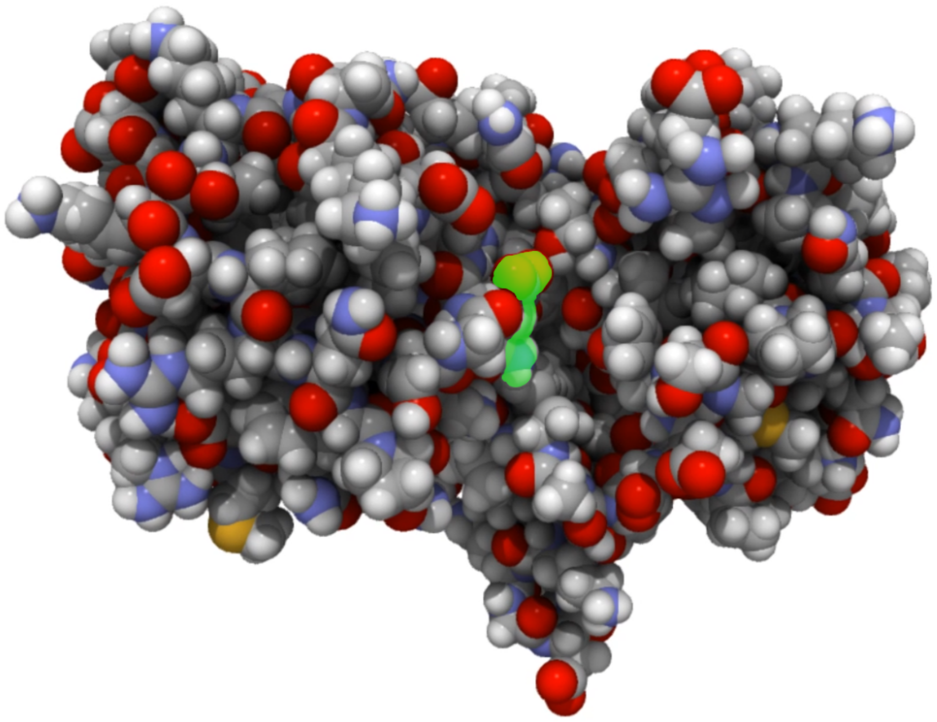
A



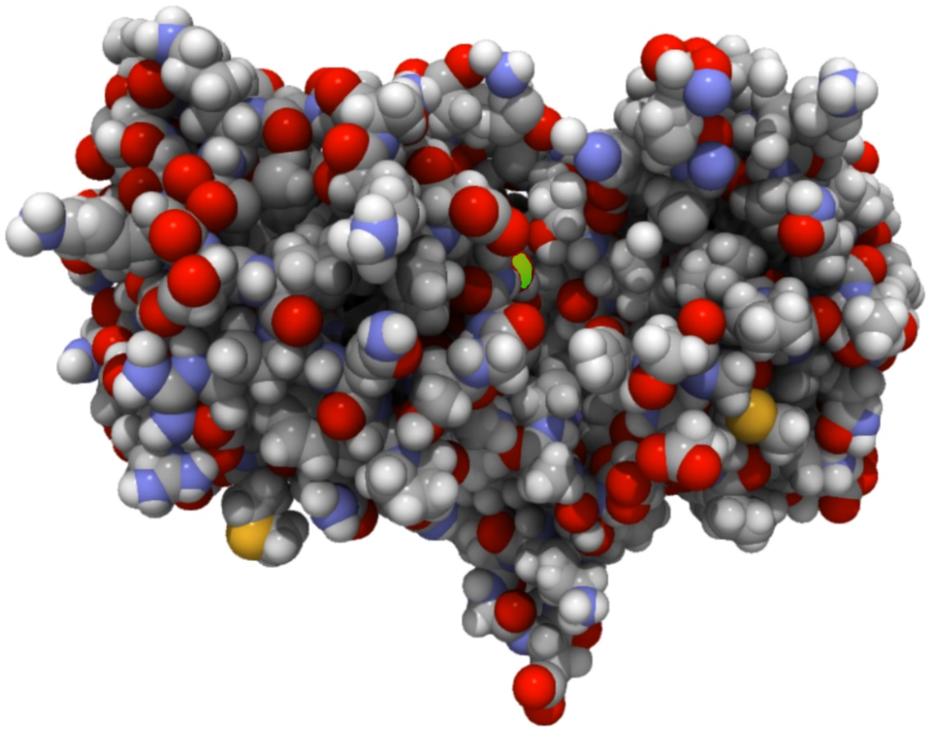
B



C



D



E

Fig. 5.17 Five images depicting how glutamine docks into GlnBP. (A) Shows the starting structure, (E) the “docked” structure. (B), (C) and (D) depict GlnBP in various stages of closure, from nearly open, to nearly closed. The closing motion, which is similar to the experimentally derived motion, involves the ligand closing around the receptor. The ligand is highlighted in green.

5.5.4 Unrealistic distortion of molecular geometry

The main limitation of this approach to modelling flexibility, is that it is possible for a user to push the model too far. In the most extreme cases, this can result in an unrealistic distortion of the molecular geometry. This is a result of the linear model used to calculate the deformation. If the force is greater than occurs naturally, the bond lengths and bond angles can go beyond feasible limits. The design choices that prevent overlap between ligand and receptor atoms, and the option to limit the maximum force permitted before halting deformation, largely prevent this. However, in certain situations this undesirable effect can still occur as shown in Figure 5.18.

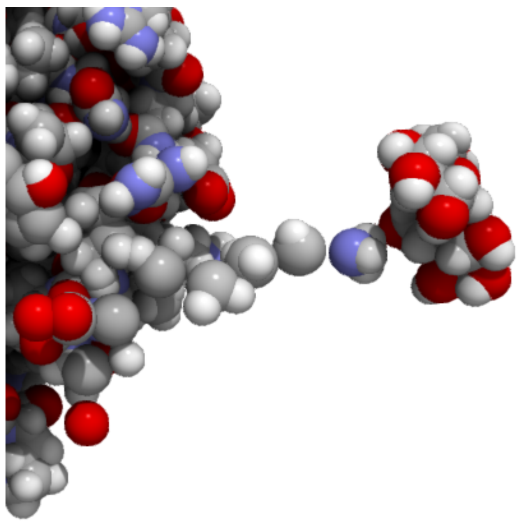


Fig. 5.18 In some receptor/ligand conformations, atoms that should be closely linked by a strong bond can be moved apart. This can be seen in the Figure, between the Nitrogen atom (blue) and the Carbon atom (grey).

Preventing this from occurring during execution, whilst maintaining a haptic refresh rate, and the level for flexibility that is present is not trivial. Simply halting the deformation if the distance between two atoms becomes unrealistic is not viable, as not only will that involve a further computation step every haptic loop, it will also limit how "deformed" a structure can become. Ideally, it should be addressed at a more fundamental level.

5.6 Discussion and Conclusion

It has been shown that linear response theory can be combined with real time force calculations to generate realistic deformation, in response to the presence of a ligand, at an interactive refresh rate. The test poses show that the movement depicted during docking is similar to the experimentally determined movement when the ligand is close to its binding site.

A pertinent question is "What happens to the receptor if the ligand is not at the binding site, but at another area on the surface of the protein?" In most locations of the protein, only a small deformation will be seen, in others, a large movement may occur. Because of this, it is envisaged that Haptimol FlexiDock will be used primarily by experts who are already familiar with the molecular structures they are studying, perhaps using this application in combination with other molecular docking tools.

By using only the eigenvectors spanning a relatively small subspace, it was shown that the number of multiplications can be reduced considerably without an appreciable loss of accuracy. For MBP, even though only 3% of the total number of the eigenvectors (the total number of degrees of freedom of MBP), could be used in order to satisfy the 2 ms time constraint, the subspace they defined contained nearly 90% of the fluctuation that occurred in the MD simulation. Furthermore, 90% of the functional movement that occurs upon binding maltose is within this subspace. Reducing the subspace still further, as necessitated by a slower processor, can still give reasonable results as demonstrated in Figure 5.14 where it is shown that 70% of functional movement in MBP is within the space defined by the first three eigenvectors. On the bench system the 2 ms time was met when using 550 eigenvalues and basic rendering for MBP, and 1220 eigenvalues for GlnBP. In both cases, this is more than sufficient to capture the important fluctuations of our test receptors.

When including the per-pixel lighting developed in Chapter 4, the 2 ms deadline is only achieved by limiting the visual refresh rate to 30 FPS, and even then, the average plus one standard deviation is above the 2 ms cut-off. Furthermore, when zooming in on the structure, and thereby increasing the fill rate, the performance drops further. Nevertheless, the resulting visual effect is crisp, and nicely highlights areas of depth in the receptor.

Of the two theories relating to how conformational change occurs during docking, the work presented in this chapter supports the induced fit hypothesis. During use, the ligand is moved into a position, and the receptor deforms around it. It would, in fact, be impossible for docking to complete using the selected fit theory, as the receptor would shut, making it impossible for the ligand to move into position.

The iterative approach incorporated within Haptimol FlexiDock was shown to stabilise the force during docking, improving the usability of the application.

In this chapter, protein-small ligand docking has been demonstrated. The approach will scale as the power of computing hardware increases, enabling its use with larger molecules and whilst incorporating more eigenvalues.

To conclude, in this chapter a haptic-assisted interactive molecular docking system that incorporates receptor flexibility was presented. Haptimol FlexiDock maintains a stable haptic refresh rate when performing docking, whilst incorporating enough of the receptor's fluctuation space that the receptor's docking response movement is accurately depicted when the ligand approaches the binding site. The system is efficient enough to run on consumer hardware, and as hardware improves, it will allow docking of larger molecules to be performed.

Chapter 6

Conclusions

The objective of this thesis, as outlined in the introduction, was to address some of the limitations of current haptic-assisted molecular docking applications. Two main objectives were outlined: develop a method to incorporate lighting techniques that can improve the perception of depth in a three dimensional scene, and develop an approach for modelling receptor flexibility in a haptic-assisted interactive docking environment. The achievement of these objectives is demonstrated in the software Haptimol FlexiDock.

Haptimol FlexiDock is a haptic-assisted interactive molecular docking application that uses linear response theory in order to calculate receptor flexibility in real time. Furthermore, FlexiDock includes per-pixel lighting effects that highlight crevices within the receptor and ligand, and includes shadows that can help to show the relative position between the receptor and ligand. When the visual frame rate is limited, these effects can be computed whilst achieving the refresh rate demanded by the haptic device.

The contents of this thesis describe the research into the techniques required for of Haptimol FlexiDock. After introducing the molecular docking, molecular interactions and the difficulties of working with haptics: mainly the 2 ms execution time limit required for smooth haptic feedback, three stages of research are discussed in Chapters 3-5.

In Chapter 3, regular grid construction algorithms were introduced, developed and tested. The results presented in Chapter 3 demonstrated that it is possible to reconstruct and use a regular grid within the 2 ms time constraint, and highlighted that the compact grid, although being the slowest to construct, is the quickest overall, especially in scenarios in which the grid is used multiple times per re-construction.

Although the grid construction algorithms presented in Chapter 3 have been touched upon in literature, the results from the experiments performed in Chapter 3 show for the first time, that GPU based regular grid construction is fast enough for use with a haptic device.

In this thesis, the described regular grid construction algorithms are heavily utilised by the rendering algorithms presented in Chapter 4, however that was not the only motivation for the development of the algorithms. In HaptimolRD, a regular grid is used to enable interactive docking to be carried out between receptors and ligands comprising 200,000 atoms each[55]. This is achieved by using the data structure to eliminate atom pairs that are sufficiently far apart for their interaction force to be small enough to omit from the calculation. Although the cut-off is not used within this work, as the receptor and ligand are both fairly modest in size, it is not inconceivable that the grid may be needed in order to accelerate flexible docking between two larger structures, or even a larger structure and a modestly sized ligand.

Indeed, using a cut-off of approximately 30 Å within FlexiDock is unlikely to make any noticeable difference to the rendered deformation when electrostatic screening is enabled, because, at that distance the screening will have reduced the electrostatic force to a negligible value, and the distance will make the vdW interactions similarly small.

Iakovou[55] demonstrated that a brute force docking approach (calculating the interaction forces between all of the atoms in the receptor and ligand) was viable for biomolecules with up to 1500 atoms on their GPU, compared to 184,000 atoms whilst using a spatial partitioning structure. Although their experiments were performed on a less powerful workstation, the premise still holds, albeit with larger values. Therefore, when performing docking between large structures, using a regular grid to reduce the number of atom pairs for which the force is calculated could offer a significant speed up.

Chapter 4 discusses adding per-pixel lighting effects to the visualisation of proteins, whilst maintaining an interactive refresh rate. The primary purpose of the development of the rendering algorithm was for inclusion within Haptimol FlexiDock, however the concept was proven in the software Haptimol Protein Trajectory Viewer (PTV), which allowed the trajectories of biomolecules up to 300k atoms in size, to be viewed in real-time, with the developed lighting effects enabled. Testing of the algorithms within PTV revealed that a visual frame rate of 24 FPS or higher could be maintained whilst rendering biomolecules up to 75k atoms in size using an Nvidia GTX 980, whilst reconstructing the acceleration structure every frame, as required within the docking application, suggesting the algorithms would be usable in an interactive docking environment.

The methods used to generate the shadows and ambient occlusion are reasonably straightforward, and are already mentioned in the literature; calculating shaded portions of the scene with ray-casting is not a new idea, and neither is the ambient occlusion algorithm; besides the presentation by Skånberg et al.[129], both approaches were used within a static protein renderer by Easdon[32]. However, applying the methods to dynamic molecules in real-time, as is achieved within PTV and FlexiDock, is a breakthrough in the field of molecular rendering.

As stated, although PTV is a useful piece of software in its own right, the primary reason for the development of the rendering algorithms was for use within Haptimol FlexiDock. As seen in Figure 2.2, the lighting effects add depth to the image, and highlight the pockets and crevices nicely. Furthermore, the shadowing helps to show where the ligand is positioned in relation to the receptor.

The inclusion of the rendering algorithms does come at a heavy computational cost however: a refresh rate of 500Hz on the haptic device is only achieved by using few eigenvalues and limiting the refresh rate of the visual display to 30Hz or lower. Even then, there is still a large amount of variation in each loop's execution time. This is caused both by the variation in the length of the haptic loop; if a collision is detected, two additional steps are required before the loop finishes, but also by the rendering algorithm competing for the limited GPU resources. However, as the performance of computing hardware improves, it will quickly become feasible to use both the rendering and deformation calculations in tandem.

Real time calculation of the deformation of a biomolecule during molecular docking, discussed in Chapter 5, represents the most significant contribution to the field of haptic-assisted interactive molecular docking within this thesis. The approach developed in Chapter 5 utilises linear response theory in order to calculate the conformational change induced in a biomolecule in response to the force exerted on it by a ligand.

When performing docking with a haptic feedback device, calculating the deformation using the entire variance-covariance matrix of atom fluctuations proved too costly. Fortunately, a feature of protein dynamics known as the important subspace states that a large amount of a biomolecule's fluctuation occurs within a relatively small subspace. This high fluctuation subspace can be defined by the first m eigenvectors of the covariance matrix. Doing so allows the flexibility of a receptor to be calculated within the 2 ms time limit set by the haptic device, a first whilst using consumer-level computer hardware.

The deformation of structures on ligand binding, calculated by Haptimol FlexiDock, were compared with crystallographic structures of the docked pose. The results showed that backbone movement that occurs when the ligand is docked within FlexiDock, aligns well with the experimental movement.

There are some limitations to the approach used to modelling flexibility. Firstly, some of the higher frequency fluctuations are lost. Consequently, some side chain motion is lost, however, as demonstrated in Section 5.5.1 of Chapter 5, even when using comparatively few components when compared to the total, a good approximation of experimentally derived docked poses can be generated, suggesting that the loss of these modes is not a huge drawback.

The second limitation relates to the haptic device itself. When performing docking, the haptic device can become unstable, and vibrates violently. The iterative approach prevents the vibration occurring to some extent, however there is still a limit to how quickly the device can update the force being rendered. When this limit is reached and overcome, which can occur when the ligand is placed into an unstable position, the device can vibrate unpleasantly.

The final limitation relates to the prerequisites of using Haptimol FlexiDock. In order to model molecular flexibility, FlexiDock requires the PDB and topology files for both the receptor and ligand, and the eigenvectors and values of the covariance matrix of atom fluctuations, calculated from an MD trajectory.

As the RCSB hosts many PDB files, these are reasonably straightforward to acquire. The topology file can often be created from the PDB file, using PDB2GMX, which is included in the GROMACS package, however if the PDB file contains non-standard residues, that is to say residues that are not explicitly included within GROMACS, expert knowledge may be needed to generate the topology.

The most difficult pre-requisite to satisfy is the generation of the eigenvalues and eigenvectors. In order to calculate them, a molecular dynamics simulation must be performed. Not only does this take a significant amount of time, expert knowledge is required to set the parameters of the simulation. Furthermore, once the simulation has been performed, generating the covariance matrix of intra-molecular atomic

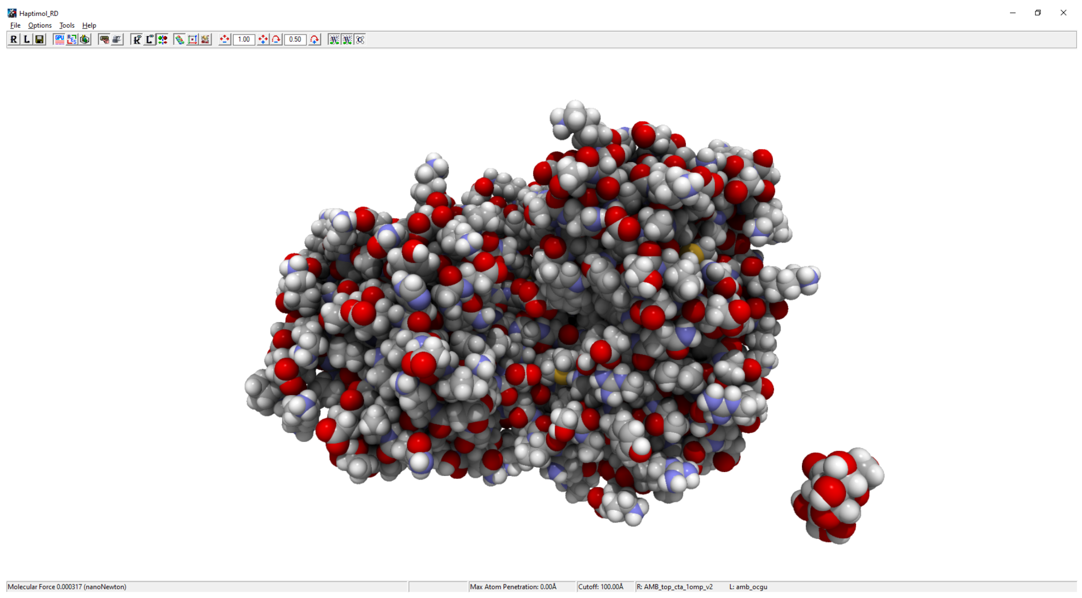


Fig. 6.1 Haptimol FlexiDock. Small biomolecule on the right is the ligand, larger biomolecule the receptor. The receptor undergoes conformational change as the ligand is moved into proximity with it.

fluctuations is a resource heavy computing task; the generation of the eigenvectors for MBP, a structure of modest size, required 58GB of RAM, using MATLAB to process the trajectory. Despite this, Haptimol FlexiDock could prove an effective tool for both computational drug design and education.

The uses of Haptimol FlexiDock

Haptimol FlexiDock (Figure 6.1) could be used within the process of structure based drug design, with the objective of identifying the ligand which binds most strongly to a known docking site, perhaps after a separate automated docking approach has identified potential ligands and binding sites. Haptimol FlexiDock could, at this stage, allow the user to use their intuition to identify the "best" candidate ligands and rule out false positives, saving time and resources that may otherwise be used achieving the same thing in the laboratory.

Although modelling flexibility requires extensive computational effort: performing an MD trajectory and then generating the covariance matrix, this only has to be performed once per receptor. This means that after a simulation has been run, any

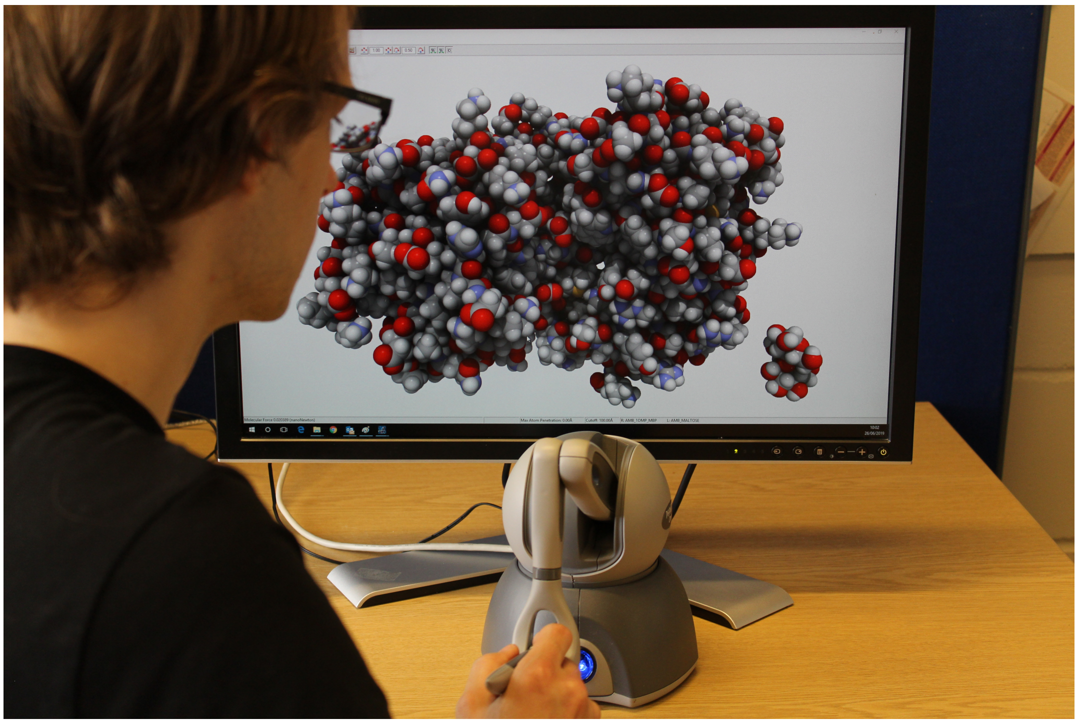


Fig. 6.2 User performing molecular docking using a haptic device and Haptimol FlexiDock

number of ligands can be tested against the receptor, without repeating the computation. This means that the tool could not only be used for comparative studies on existing lead compounds but, in expert hands, help foster ideas as to how the ligands might be improved to enhance binding. In other words FlexiDock could provide an environment where hypotheses are nurtured and tested.

Haptimol FlexiDock could also be used for education. Lancaster[78] highlights the difficulties that some young chemists have visualising 3D structures from 2D diagrams, and how immersive virtual worlds can help understanding. Furthermore the experiments performed as part of project GROPE[20] indicated that haptic feedback helped chemists understand the binding site, and the forcefields surrounding it. Therefore, it follows that Haptimol FlexiDock should prove to be a useful tool for teaching students about how molecules bind together, and how the receptor can deform to accommodate the ligand. Figure 6.2 shows a user performing interactive molecular docking with Haptimol FlexiDock.

Lighting effects

The per-pixel lighting effects developed in Chapter 4 were intended to help enhance the topography of the biomolecules involved in docking. The techniques achieve this: the ambient occlusion darkens local pockets, whilst the shadow algorithm helps to show the position of the ligand relative to the receptor, and also how the parts of the receptor deform in relation to one another.

However, in order for Haptimol FlexiDock to achieve a force feedback refresh rate of 500Hz or greater whilst performing rendering with advanced lighting, the refresh rate of the visual render had to be limited to 30Hz. Although this is fast enough to be considered real time, the movements are not as smooth as they would be at 60Hz, for example. This leads on to an interesting question: “What is more important, the graphical visualisation of the biomolecule, or the haptic feedback?”

Rendering vs Haptics

When considering the relative importance of visuals v haptics, it could be argued that a good visualisation is the most important aspect of the application. The user will use their sense of sight to view the receptor and ligand, judge their topologies, and form hypotheses relating to the location of likely binding sites. Then, during the docking process, they will use their sense of sight to judge and understand the relative position of ligand and receptor. It is vitally important therefore, that an adequate visualisation is provided.

However, the importance of the haptic feedback cannot be understated. Early research[20] indicated that performing molecular docking was much quicker with haptic assistance than without. My own observations agree with this: being *pulled* into the docking site, and feeling strong resistance when trying to move the ligand into areas where repulsive forces are present greatly assists the binding process, and is more informative than rendering the resulting forces as a number, or indeed as a colour.

So what is more important? A good visualisation or a high haptic refresh rate? A high quality visualisation up to a minimum standard - sufficient to be able to quickly judge relative position and depth, but little more than that; reflections for instance, would add little to the software as a molecular docking package. With that satisfied, haptics should be prioritised, as they make for a more ergonomic experience, and offer a useful way to learn about the protein-ligand interaction.

6.1 Impact

In the field of interactive molecular docking, Haptimol FlexiDock represents a major step forward. For the first time, the flexibility of the receptor is modelled in real time on consumer grade hardware, whilst maintaining a haptic refresh rate of 500 Hz or greater. This represents a significant breakthrough, as, until this point, receptor flexibility has been ignored, or implemented poorly, in the field of interactive docking.

It is difficult to judge the overall impact Haptimol FlexiDock will have on the scientific community. FlexiDock was sent to a research group based in Japan, who are interested in biomolecular interactions, in order for them to try it out and highlight any missing features that they consider vital for their research.

Feedback was good, and the tool was met with enthusiasm and interest, with one researcher keen to show it to his employees. This is encouraging, and demonstrates that the development path is correct, however the overall message was: *incorporate ligand flexibility, and this tool will be very useful.* Therefore, that needs to be the next area of research undertaken. When receptor and ligand flexibility are integrated together, the potential impact of the tool, in both learning and teaching, could be massive.

The impact of the presented ideas beyond the field of molecular docking is more difficult to judge. Although the ideas presented may be transferable, it is unlikely the developed algorithms will be, without some level of adjustment for a new purpose, as they are highly optimised for molecular docking. However interactive exploration of a problem using haptics could have a variety of uses in engineering.

An interesting idea is the idea of a virtual wind tunnel. The user could explore areas of high and low pressure using the haptic device, and increase their understanding of the surface they're studying, and the affect of adjusting various spoilers. It is likely computation demands of areas of low and high pressure would, similarly to molecular deformation, exceed the amount of computational power available on an ordinary desktop workstation. Therefore, a dimensionality reduction technique would need to be utilised; whether CFD simulations could be used in the same manor as molecular docking simulations are used in the work presented in Chapter 5 would need to be investigated.

The method for calculating the deformation developed in Chapter 5 may also have uses in other fields. GPUs are highly suited for performing mathematical computations in parallel, more so than a CPU, which have few computation cores when compared to a GPU. One of the limitations of using the GPU for general purpose computation is the overhead caused by having to instruct the GPU to perform computation; every time a kernel is launched, there is some overhead. The method presented in Chapter 5 reduces the amount of overhead generated, by reducing what would be three calls to a conventional blas library down to two kernel calls.

A generalisation of matrix multiplication method could perhaps, be useful in some bespoke applications, as the process allows multiplying a matrix by a vector, and then a scaler with a single kernel call. It is possible that a field that relies heavily on matix-vector multiplication may benefit from it – the field of cryptography, for example. The multiplication process described in Chapter 5 is heavily tuned to perform the deformation calculation for which it was intended, so it is likely that further changes would be required in order to achieve optimum performance for a separate application.

6.2 Future work

6.2.1 Ligand flexibility

The most obvious next step in the development of Haptimol FlexiDock would be the addition of ligand flexibility. For larger proteins, it is possible that the linear response method described in detail in Chapter 5 could be used. However, with smaller ligands, this approach is unlikely to be suitable, owing to the fact that smaller ligands often have rotatable bonds that can move through 360 degrees which cannot be handled using LRT. In these situations, energy minimization, the process of finding an arrangement in space of a collection of atoms where the total inter-atomic force is close to zero, could be used. Modelling ligand flexibility will increase the amount of computation required per haptic loop. Nevertheless, incorporating it within Haptimol FlexiDock will bring the application another step closer to modelling reality.

6.2.2 Unrealistic distortion of molecular geometry

The biggest limitation in the approach to modelling receptor flexibility presented in Chapter 5 is that in some situations, bonds that link atoms can be stretched, and even broken (See Chapter 5, Section 5.5.4). Methods to prevent this happening, either during the haptic session or during a precomputation stage need to be investigated.

6.2.3 User testing and evaluation

In this thesis, there has been no discussion of undertaking formal user testing and evaluation of Haptimol FlexiDock, beyond discussions with the research groups that have supported this research. The reason for this is the desire for the application to be complete before undertaking evaluation. This means that ligand flexibility needs to be accounted for, and the undesirable distortion of the biomolecular geometry handled elegantly. The motivation for this is it will require a significant effort to acquire feedback from a number of biologists who are involved in the field of molecular docking – it makes little sense to do this before the software is finished, as the resulting feedback will likely relate to the limitations that we are already aware of. Better rather, to have the software complete and ready for use, in order to make a good first impression of it, and for it to be ready and useful as the wider community become aware of it.

6.2.4 Automatic Determination of Eigenvalues

In Chapter 5, the number of eigenvalues used during the docking experiments was determined by hand, after evaluating the performance test results. The optimum value for use on any one computer could be determined automatically by utilising a simple benchmark routine, wherein an increasing number of eigenvalues are used until the average haptic refresh rate drops below a specified threshold. This would improve the ergonomics of the application, and insure that the maximum number of eigenvalues are used whilst maintaining the desired haptic refresh rate, regardless of the hardware used.

6.2.5 Torques

Currently Haptimol FlexiDock does not model the torque that the ligand would experience during binding. The torque forces would ordinarily make the ligand rotate upon its axes, into a more optimum orientation. These forces were ignored, largely because low-cost haptic devices cannot render them. Computing the torque caused by the interaction forces would add a negligible overhead to the force calculation, and could improve the user experience, provided their haptic device is 6-DoF capable.

6.2.6 Molecular Visualisations

In Haptimol FlexiDock, the only molecular visualisation supported is space filling mode, however this is by no means the only molecular representation. Figure 6.3 shows some of the other representations available within the molecular graphics package PyMol. These different representations allow viewing of different aspects of the biomolecule. For example the path of the protein chain is easier to view using the "backbone" method than it is with space-fill.

Figure 6.1 (B) shows the solvent excluded surface (SES) of MBP. This visualisation shows the surface area of the biomolecule which comes into contact with the solvent. The SES highlights the topography of the biomolecule, which can be useful when identifying potential binding sites. Computing the surface of a static biomolecule of modest size is not too computationally intensive, however computing it as the biomolecule deforms is not as straightforward. Nevertheless, as a result of the advantages the surface offers, algorithms have been developed that may allow computation of the surface in real time.

A promising implementation that could be developed, and then incorporated within Haptimol FlexiDock was presented by Krone et al.[75] and improved by Jurčík et al.[63]. Krone et al.[75] demonstrated rendering the SES of a molecular trajectory at an interactive refresh rate of a biomolecule comprising approximately 10,000 atoms by calculating the "reduced surface[119]", and then ray casting it using OpenGL. The resulting render is per-pixel accurate, and contains some depth darkening, making it ideal for use within FlexiDock.

Since the beginning of this project, the number of floating point operations consumer graphics cards can perform per second has more than doubled. This increase in performance allows more computation to be performed per haptic or visual frame. Within Haptimol FlexiDock, this translates to performing docking between larger biomolecules, whilst using more eigenvalues or supporting ligand flexibility. On the other hand, the increased floating point performance could also allow more advanced

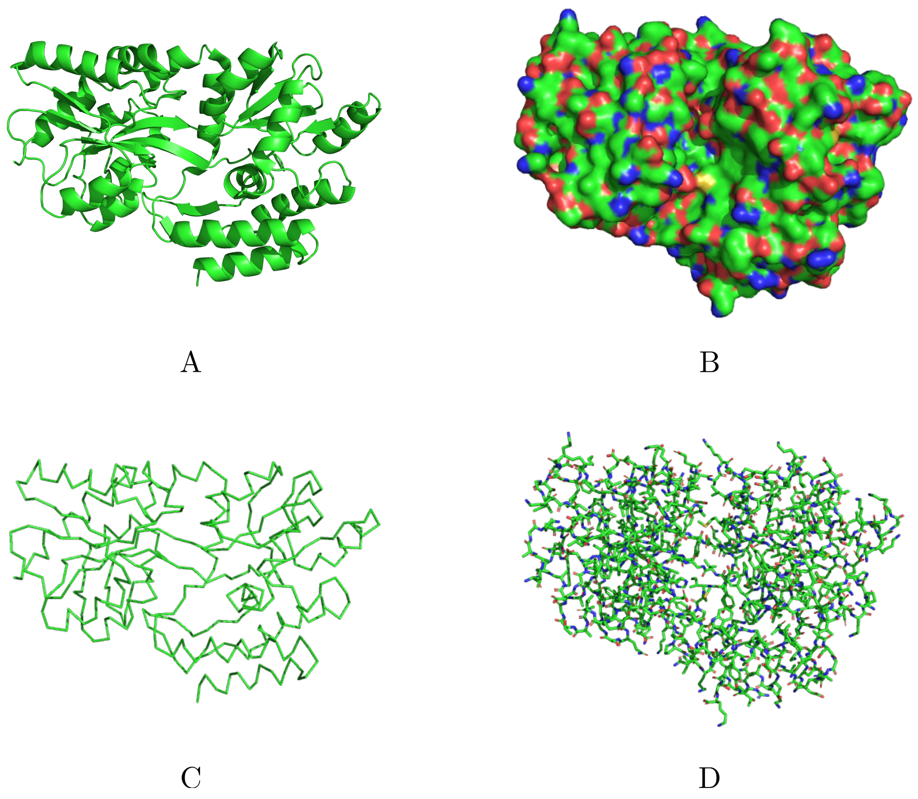


Fig. 6.3 Four different protein visualisation modes generated with PyMol[123]. (A) Liquorice, (B) Surface, (C) Backbone, (D) Sticks

.

rendering techniques: Real time ray tracing, for example, is well on the way to become possible on regular consumer computer systems[94].

If the computational ability of modern graphics cards continues to increase, modelling both ligand and receptor flexibility within the 2 ms time frame imposed by the haptic device should be achievable. The research presented in this thesis represents the first step, and although there is a long way to go, the final goal of fully interactive molecular docking, with molecular flexibility, high fidelity graphics and smooth haptic feedback should be within reach.

6.3 Final Remarks

The contents of this thesis push forward the field of interactive molecular docking. The main contribution to the field is an approach to modelling biomolecular flexibility whilst supporting a haptic refresh rate of over 500Hz. Also, contributions are made to the biomolecular rendering field: for the first time, a space filling representation of a deforming protein has been rendered in real-time, with high quality per-pixel lighting effects including ray-cast shadows.

References

[1] Alberts, B., Bray, D., Hopkin, K., Johnson, A., Lewis, J., Raff, M. C., Roberts, K., and Walter, P. (2014). *Essential cell biology*. W.W. Norton Co.

[2] Amadei, A., Linssen, A. B., and Berendsen, H. J. (1993). Essential dynamics of proteins. *Proteins*, 17(4):412–425.

[3] Annen, T., Dong, Z., Mertens, T., Bekaert, P., Seidel, H.-P., and Kautz, J. (2008). Real-time, All-frequency Shadows in Dynamic Scenes. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 34:1–34:8, New York, NY, USA. ACM.

[4] Anthopoulos, A., Grimstead, I., and Brancale, A. (2013). GPU-accelerated molecular mechanics computations. *Journal of Computational Chemistry*, 34(26):2249–2260.

[5] Anthopoulos, A., Pasqualetto, G., Grimstead, I., and Brancale, A. (2014). Haptic-driven, interactive drug design: implementing a GPU-based approach to evaluate the induced fit effect. *Faraday Discussions*, 169:323–342.

[6] Antunes, D. A., Devaurs, D., and Kavraki, L. E. (2015). Understanding the challenges of protein flexibility in drug design. *Expert Opinion on Drug Discovery*, 10(12):1301–1313.

[7] Apostolakis, J., Plückthun, A., and Caflisch, A. (1998). Docking small ligands in flexible binding sites. *Journal of Computational Chemistry*, 19(1):21–37.

[8] Arunan, E., Desiraju, G., Klein, R., Sadlej, J., Scheiner, S., Alkorta, I., Clary, D., Crabtree, R., Dannenberg, J., Hobza, P., Kjærgaard, H., Legon, A., Mennucci, B., and Nesbitt, D. (2011). Definition of the hydrogen bond (iupac recommendations 2011). *Pure and Applied Chemistry*, 83:1637–1641.

[9] Barbieri, D., Cardellini, V., and Filippone, S. (2013). Fast Uniform Grid Construction on GPGPUs Using Atomic Operations. In *PARCO*, pages 295–304.

[10] Barril, X. and Morley, S. D. (2005). Unveiling the Full Potential of Flexible Receptor Docking Using Multiple Crystallographic Structures. *Journal of Medicinal Chemistry*, 48(13):4432–4443.

[11] Bavoil, L. and Sainz, M. (2009). Multi-layer Dual-resolution Screen-space Ambient Occlusion. In *SIGGRAPH 2009: Talks*, SIGGRAPH '09, pages 45:1–45:1, New York, NY, USA. ACM.

[12] Bavoil, L., Sainz, M., and Dimitrov, R. (2008). Image-space Horizon-based Ambient Occlusion. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH '08, pages 22:1–22:1, New York, NY, USA. ACM.

[13] Bayazit, O. B., Song, G., and Amato, N. M. (2001). Ligand binding with OBPRM and user input. In *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 1, pages 954–959 vol.1.

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

[15] Bivall, P. (2010). *Touching the Essence of Life : Haptic Virtual Proteins for Learning*. PhD thesis, Linköping University.

[16] Bivall, P., Ainsworth, S., and Tibell, L. A. E. (2011). Do haptic representations help complex molecular learning? *Science Education*, 95(4):700–719.

[17] Borrelli, K. W., Cossins, B., and Guallar, V. (2010). Exploring hierarchical refinement techniques for induced fit docking with protein and ligand flexibility. *Journal of Computational Chemistry*, 31(6):1224–1235.

[18] Brooijmans, N. and Humblet, C. (2010). Chemical space sampling by different scoring functions and crystal structures. *Journal of Computer-Aided Molecular Design*, 24(5):433–447.

[19] Brooks, Jr., F. P., Ouh-Young, M., Batter, J. J., and Jerome Kilpatrick, P. (1990). Project GROPEHaptic Displays for Scientific Visualization. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pages 177–185, New York, NY, USA. ACM.

[20] Brooks Jr, F. P., Ouh-Young, M., Batter, J. J., and Jerome Kilpatrick, P. (1990). Project GROPEHaptic displays for scientific visualization. In *ACM SIGGraph computer graphics*, volume 24, pages 177–185. ACM.

[21] Bunnell, M. (2005). Dynamic ambient occlusion and indirect lighting. In *GPU gems 2*, volume 2, pages 223–233. Addison-Wesley Professional.

[22] Case, A, D., Betz, M, R., Cerutti, D., Cheatham, T., Darden, T., Duke, R., Giese, T., Gohlke, H., Goetz, A., Homeyer, N., Izadi, S., Kovalenko, A., Kurtzman, T., Lee, T., LeGrand, S., Li, P., Lin, C., Liu, J., Luchko, T., Luo, R., Mermelstein, D., Merz, K., Miao, Y., Monard, G., Nguyen, C., Nguyen, H., Omelyan, I., Onufriev, A., Pan, F., Qi, R., Roe, D., Roitberg, A., Sagui, C., Schott-Verdugo, S., Shen, J., Simmerling, C., Smith, J., Salomon-Ferrer, R., Swails, J., Walker, R., Wang, J., Wei, H., Wolf, R., Wu, X., Xiao, L., York, D., and Kollan, P. (2016). Amber 2016.

[23] Cheng, J., Grossman, M., and McKercher, T. (2014). *Professional CUDA C Programming*. Wrox Press Ltd., Birmingham, UK, UK, 1st edition.

[24] Corp., N. (2018). Turing architecture whitepaper. https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf. Last checked April 2019.

[25] Cosenza, B. (2008). A Survey on Exploiting Grids for Ray Tracing. In Scarano, V., Chiara, R. D., and Erra, U., editors, *Eurographics Italian Chapter Conference*. The Eurographics Association.

[26] Crassin, C., Neyret, F., Sainz, M., Green, S., and Eisemann, E. (2011). Interactive indirect illumination using voxel cone tracing. In *Computer Graphics Forum*, volume 30, pages 1921–1930. Wiley Online Library.

[27] Crow, F. C. (1977). Shadow Algorithms for Computer Graphics. In *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '77, pages 242–248, New York, NY, USA. ACM.

[28] Daunay, B., Micaelli, A., and Regnier, S. (2007). 6 DOF haptic feedback for molecular docking using wave variables. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 840–845.

[29] Dimitrov, R. (2007). Cascaded shadow maps. *Developer Documentation, NVIDIA Corp*.

[30] Donnelly, W. and Lauritzen, A. (2006). Variance Shadow Maps. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, I3D '06, pages 161–165, New York, NY, USA. ACM.

[31] Duan, Y., Wu, C., Chowdhury, S., Lee, M. C., Xiong, G., Zhang, W., Yang, R., Cieplak, P., Luo, R., Lee, T., Caldwell, J., Wang, J., and Kollman, P. (2003). A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations. *Journal of Computational Chemistry*, 24(16):1999–2012.

[32] Easdon, R. (2013). *Ambient occlusion and shadows for molecular graphics*. masters, University of East Anglia.

[33] Ellis, R. E., Ismaeil, O. M., and Lipsett, M. G. (1996). Design and evaluation of a high-performance haptic interface. *Robotica*, 14(3):321–327.

[34] Essmann, U., Perera, L., Berkowitz, M. L., Darden, T., Lee, H., and Pedersen, L. G. (1995). A smooth particle mesh ewald method. *The Journal of Chemical Physics*, 103(19):8577–8593.

[35] Everitt, C. and Kilgard, M. J. (2003). Practical and robust stenciled shadow volumes for hardware-accelerated rendering. *arXiv preprint cs/0301002*.

[36] Favera, E. C. D. and Celes, W. (2012). Ambient Occlusion Using Cone Tracing with Scene Voxelization. In *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images*, pages 142–149.

[37] Filion, D. and McNaughton, R. (2008). Effects & techniques. In *ACM SIGGRAPH 2008 Games*, pages 133–164. ACM.

[38] Fujimoto, A., Tanaka, T., and Iwata, K. (1986). Arts: Accelerated ray-tracing system. *Computer Graphics and Applications, IEEE*, 6(4):16–26.

[39] Férey, N., Nelson, J., Martin, C., Picinali, L., Bouyer, G., Tek, A., Bourdot, P., Burkhardt, J. M., Katz, B. F. G., Ammi, M., Etchebest, C., and Autin, L. (2009). Multisensory VR interaction for protein-docking in the CoRSAIRe project. *Virtual Reality*, 13(4):273.

[40] Garanzha, K., Premože, S., Bely, A., and Galaktionov, V. (2011). Grid-based SAH BVH construction on a GPU. *The Visual Computer*, 27(6-8):697–706.

[41] Goh, C.-S., Milburn, D., and Gerstein, M. (2004). Conformational changes associated with protein-protein interactions. *Current Opinion in Structural Biology*, 14(1):104–109.

[42] Götz, A. W., Williamson, M. J., Xu, D., Poole, D., Le Grand, S., and Walker, R. C. (2012). Routine microsecond molecular dynamics simulations with AMBER on GPUs. 1. generalized born. *Journal of Chemical Theory and Computation*, 8(5):1542–1555.

[43] Gouraud, H. (1971). Continuous shading of curved surfaces. *IEEE Transactions on Computers*, C-20(6):623–629.

[44] Green, S. (2010). Particle Simulation using CUDA. *NVIDIA Whitepaper*.

[45] Grottel, S., Krone, M., Müller, C., Reina, G., and Ertl, T. (2015). MegaMol #x2014;A Prototyping Framework for Particle-Based Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 21(2):201–214.

[46] Grottel, S., Krone, M., Scharnowski, K., and Ertl, T. (2012). Object-space ambient occlusion for molecular dynamics. In *2012 IEEE Pacific Visualization Symposium*, pages 209–216.

[47] Gumhold, S. (2003). Splatting illuminated ellipsoids with depth correction. In *VMV*, pages 245–252.

[48] Hayward, S. and Go, N. (1995). Collective Variable Description of Native Protein Dynamics. *Annual Review of Physical Chemistry*, 46(1):223–250.

[49] Heidmann, T. (1991). Real shadows, real time. *Iris Universe*, 18:28–31.

[50] Hess, E. L. (1970). Origins of molecular biology. *Science*, 168(3932):664–669.

[51] Hoberock, j. and Yuntao, J. (2007). High-quality ambient occlusion. In Nguyen, H., editor, *GPU gems 3*, pages 257–273. Addison-Wesley Professional.

[52] Hou, X. and Sourina, O. (2010). Haptic Rendering Algorithm for Biomolecular Docking with Torque Force. In *2010 International Conference on Cyberworlds*, pages 25–31.

[53] Hsiao, C.-D., Sun, Y.-J., Rose, J., and Wang, B.-C. (1996). The Crystal Structure of Glutamine-binding Protein fromEscherichia coli. *Journal of Molecular Biology*, 262(2):225–242.

[54] Humphrey, W., Dalke, A., and Schulten, K. (1996). VMD: visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, 27–28.

[55] Iakovou, G. (2015). *Simulating molecular docking with haptics*. doctoral, University of East Anglia.

[56] Iakovou, G., Hayward, S., and Laycock, S. (2014). A real-time proximity querying algorithm for haptic-based molecular docking. *Faraday Discussions*, 169(0):359–377.

[57] Iakovou, G., Hayward, S., and Laycock, S. D. (2015). Adaptive GPU-accelerated force calculation for interactive rigid molecular docking using haptics. *Journal of Molecular Graphics and Modelling*, 61:1–12.

[58] Iakovou, G., Hayward, S., and Laycock, S. D. (2017). Virtual Environment for Studying the Docking Interactions of Rigid Biomolecules with Haptics. *Journal of Chemical Information and Modeling*, 57(5):1142–1152.

[59] Ikeguchi, M., Ueno, J., Sato, M., and Kidera, A. (2005). Protein Structural Change Upon Ligand Binding: Linear Response Theory. *Physical Review Letters*, 94(7):078102.

[60] Ize, T., Wald, I., Robertson, C., and Parker, S. (2006). An Evaluation of Parallel Grid Construction for Ray Tracing Dynamic Scenes. In *IEEE Symposium on Interactive Ray Tracing 2006*, pages 47–55.

[61] Jiang, F. and Kim, S. H. (1991). "Soft docking": matching of molecular surface cubes. *Journal of Molecular Biology*, 219(1):79–102.

[62] Joung, I. S. and Cheatham III, T. E. (2008). Determination of alkali and halide monovalent ion parameters for use in explicitly solvated biomolecular simulations. *The Journal of Physical Chemistry*.

[63] Jurčík, A., Parulek, J., Sochor, J., and Kozlikova, B. (2016). Accelerated visualization of transparent molecular surfaces in molecular dynamics. In *2016 IEEE Pacific Visualization Symposium (PacificVis)*, pages 112–119.

[64] Kajalin, V. (2009). Screen space ambient occlusion. *ShaderX7: Advanced Rendering Techniques*, pages 413–424.

[65] Kalojanov, J. and Slusallek, P. (2009). A Parallel Algorithm for Construction of Uniform Grids. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, pages 23–28, New York, NY, USA. ACM.

[66] Karras, T. (2012). Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. In *Proceedings of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics*, pages 33–37. Eurographics Association.

[67] Karras, T. and Aila, T. (2013). Fast parallel construction of high-quality bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 89–99. ACM.

[68] Kellenberger, E., Springael, J.-Y., Parmentier, M., Hachet-Haas, M., Galzi, J.-L., and Rognan, D. (2007). Identification of nonpeptide CCR5 receptor agonists by structure-based virtual screening. *Journal of Medicinal Chemistry*, 50(6):1294–1303.

[69] Keserû, G. M. and Kolossváry, I. (2001). Fully Flexible Low-Mode Docking:Application to Induced Fit in HIV Integrase. *Journal of the American Chemical Society*, 123(50):12708–12709.

[70] Kitao, A. and Go, N. (1999). Investigating protein dynamics in collective coordinate space. *Current Opinion in Structural Biology*, 9(2):164–169.

[71] Knegtel, R. M., Kuntz, I. D., and Oshiro, C. M. (1997). Molecular docking to ensembles of protein structures. *Journal of Molecular Biology*, 266(2):424–440.

[72] Knoll, A., Wald, I., Navrátil, P. A., Papka, M. E., and Gaither, K. P. (2013). Ray tracing and volume rendering large molecular data on multi-core and many-core architectures. In *Proceedings of the 8th International Workshop on Ultrascale Visualization*, page 5. ACM.

[73] Koltun, W. L. (1965). Precision space-filling atomic models. *Biopolymers*, 3(6):665–679.

[74] Kontkanen, J. and Laine, S. (2005). Ambient occlusion fields. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 41–48. ACM.

[75] Krone, M., Bidmon, K., and Ertl, T. (2009). Interactive Visualization of Molecular Surface Dynamics. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1391–1398.

[76] Lagae, A. and Dutré, P. (2008). Compact, Fast and Robust Grids for Ray Tracing. *Computer Graphics Forum*, 27(4):1235–1244.

[77] Lai-Yuen, S. K. and Lee, Y.-S. (2005). Computer-aided molecular design (CAMD) with force-torque feedback. In *Ninth International Conference on Computer Aided Design and Computer Graphics (CAD-CG'05)*, pages 6 pp.–.

[78] Lancaster, S. J. (2018). Immersed in virtual molecules. *Nature Reviews Chemistry*, 2(10):253–254.

[79] Lauritzen, A. and McCool, M. (2007). Summed-area variance shadow maps. *GPU Gems*, pages 157–182.

[80] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. (2009). Fast BVH construction on GPUs. In *Computer Graphics Forum*, volume 28, pages 375–384. Wiley Online Library.

[81] Leach, A. R. (1994). Ligand docking to proteins with discrete side-chain flexibility. *Journal of Molecular Biology*, 235(1):345–356.

[82] Leach, A. R., Shoichet, B. K., and Peishoff, C. E. (2006). Prediction of Protein-Ligand Interactions. Docking and Scoring: Successes and Gaps. *Journal of Medicinal Chemistry*, 49(20):5851–5855.

[83] Lee, Y.-G. and Lyons, K. W. (2004). Smoothing haptic interaction using molecular force calculations. *Computer-Aided Design*, 36(1):75–90.

[84] Lengauer, T. and Rarey, M. (1996). Computational methods for biomolecular docking. *Current Opinion in Structural Biology*, 6(3):402–406.

[85] Lensink, M. F., Velankar, S., and Wodak, S. J. (2017). Modeling protein–protein and protein–peptide complexes: CAPRI 6th edition. *Proteins: Structure, Function, and Bioinformatics*, 85(3):359–377.

[86] Maier, J. A., Martinez, C., Kasavajhala, K., Wickstrom, L., Hauser, K. E., and Simmerling, C. (2015). ff14sb: Improving the accuracy of protein side chain and backbone parameters from ff99sb. *Journal of Chemical Theory and Computation*.

[87] Manglik, A., Lin, H., Aryal, D. K., McCorvy, J. D., Dengler, D., Corder, G., Levit, A., Kling, R. C., Bernat, V., Hübner, H., Huang, X.-P., Sassano, M. F., Giguère, P. M., Löber, S., Da Duan, Scherrer, G., Kobilka, B. K., Gmeiner, P., Roth, B. L., and Shoichet, B. K. (2016). Structure-based discovery of opioid analgesics with reduced side effects. *Nature*, 537(7619):185–190.

[88] Martin, T. and Tan, T.-S. (2004). Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, EGSR'04, pages 153–160, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[89] Matthews, N., Easdon, R., Kitao, A., Hayward, S., and Laycock, S. (2017). High quality rendering of protein dynamics in space filling mode. *Journal of Molecular Graphics and Modelling*, 78:158–167.

[90] Matthews, N., Kitao, A., Laycock, S., and Hayward, S. (2019). Haptic-assisted interactive molecular docking incorporating receptor flexibility. *Journal of Chemical Information and Modeling*.

[91] McIntosh-Smith, n., Tchertanov, n., Nerukh, n., Stone, n., Baaden, n., Hayward, n., Glowacki, n., Olson, n., Zoppè, n., Petrov, n., Tchertanov, n., Hall, n., Reiher, n., Brancale, n., Haag, n., O'Donoghue, n., Brooks, n., Raffin, n., Iakovou, n., Chavent, n., Montes, n., Baker, n., Thomas, n., Woods, n., and Hirst, n. (2014). Computing power revolution and new algorithms: GP-GPUs, clouds and more: general discussion. *Faraday Discussions*, 169:379–401.

[92] McPherson, A. and Gavira, J. A. (2014). Introduction to protein crystallization. *Acta Crystallographica Section F: Structural Biology Communications*, 70(1):2–20.

[93] Mehler, E. L. and Solmajer, T. (1991). Electrostatic effects in proteins: comparison of dielectric and charge models. *Protein Engineering, Design and Selection*, 4(8):903–910.

[94] Microsoft, D. T. (2018). Announcing Microsoft DirectX Raytracing!

[95] Miller, G. (1994). Efficient algorithms for local and global accessibility shading. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 319–326. ACM.

[96] Minsky, M., Ming, O.-y., Steele, O., Brooks, Jr., F. P., and Behensky, M. (1990). Feeling and Seeing: Issues in Force Display. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, I3D '90, pages 235–241, New York, NY, USA. ACM.

[97] Mittring, M. (2007). Finding next gen: Cryengine 2. In *ACM SIGGRAPH 2007 courses*, pages 97–121. ACM.

[98] Moitessier, N., Englebienne, P., Lee, D., Lawandi, J., and Corbeil, C. R. (2008). Towards the development of universal, fast and highly accurate docking/scoring methods: a long way to go. *British Journal of Pharmacology*, 153(S1):S7–S26.

[99] Moll, A., Hildebrandt, A., Lenhof, H.-P., and Kohlbacher, O. (2006). BALLView: a tool for research and education in molecular modeling. *Bioinformatics*, 22(3):365–366.

[100] Morris, G. M., Huey, R., Lindstrom, W., Sanner, M. F., Belew, R. K., Goodsell, D. S., and Olson, A. J. (2009). AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility. *Journal of Computational Chemistry*, 30(16):2785–2791.

[101] Morris, G. M. and Lim-Wilby, M. (2008). Molecular Docking. In Kukol, A., editor, *Molecular Modeling of Proteins*, Methods Molecular Biology™, pages 365–382. Humana Press, Totowa, NJ.

[102] Nagata, H., Mizushima, H., and Tanaka, H. (2002). Concept and prototype of protein–ligand docking simulator with force feedback technology. *Bioinformatics*, 18(1):140–146.

[103] NVIDIA (2018). CUDA c programming guide.

[104] Ooms, F. (2000). Molecular modeling and computer aided drug design. examples of their applications in medicinal chemistry. *Current Medicinal Chemistry*, 7(2):141–158.

[105] Oostenbrink, C., Villa, A., Mark, A. E., and van Gunsteren, W. F. (2004). A biomolecular force field based on the free enthalpy of hydration and solvation: the GROMOS force-field parameter sets 53a5 and 53a6. *Journal of Computational Chemistry*, 25(13):1656–1676.

[106] Ouh-Young, M. (1990). *Force Display in Molecular Docking*. PhD Thesis, The University of North Carolina at Chapel Hill.

[107] Pagadala, N. S., Syed, K., and Tuszynski, J. (2017). Software for molecular docking: a review. *Biophysical Reviews*, 9(2):91–102.

[108] Papaioannou, G., Menexi, M. L., and Papadopoulos, C. (2010). Real-time volume-based ambient occlusion. *IEEE transactions on visualization and computer graphics*, 16(5):752–762.

[109] Parker, S., Shirley, P., and Smits, B. (1998). Single sample soft shadows. Technical report, Technical Report UUCS-98-019, Computer Science Department, University of Utah.

[110] Pattabiraman, N., Levitt, M., Ferrin, T. E., and Langridge, R. (1985). Computer graphics in real-time docking with energy calculation and minimization. *Journal of Computational Chemistry*, 6(5):432–436.

[111] Persson, P. B., Cooper, M. D., Tibell, L. A., Ainsworth, S., Ynnerman, A., and Jonsson, B.-H. (2007). Designing and evaluating a haptic system for biomolecular education. In *Virtual Reality Conference, 2007. VR'07. IEEE*, pages 171–178. IEEE.

[112] Pharr, M. and Green, S. (2004). Ambient occlusion. In Fernando, R., editor, *GPU Gems*, chapter 17, pages 279–292. Pearson Higher Education.

[113] Phong, B. T. (1975). Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317.

[114] Piana, S., Lindorff-Larsen, K., Dirks, R. M., Salmon, J. K., Dror, R. O., and Shaw, D. E. (2012). Evaluating the Effects of Cutoffs and Treatment of Long-range Electrostatics in Protein Folding Simulations. *PLoS ONE*, 7(6).

[115] Pradeepkiran, J. A. and Reddy, P. H. (2019). Structure based design and molecular docking studies for phosphorylated tau inhibitors in alzheimer's disease. *Cells*, 8(3).

[116] Reinhard, E., Smits, B., and Hansen, C. (2000). Dynamic Acceleration Structures for Interactive Ray Tracing. In Péroche, P. B. and Rushmeier, H., editors, *Rendering Techniques 2000*, Eurographics, pages 299–306. Springer Vienna. DOI: 10.1007/978-3-7091-6303-0_27.

[117] Rhodes, G. (2006). *Crystallography Made Crystal Clear*. Elsevier.

[118] Ricci, A., Anthopoulos, A., Massarotti, A., Grimstead, I., and Brancale, A. (2012). Haptic-driven applications to molecular modeling: state-of-the-art and perspectives. *Future Medicinal Chemistry*, 4(10):1219–1228.

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

[120] Sattler, M., Sarlette, R., Zachmann, G., and Klein, R. (2004). Hardware-accelerated ambient occlusion computation. *Proceedings of Vision, Modeling, and Visualization 2004*, pages 331–338.

[121] Sayle, R. A. and Milner-White, E. J. (1995). RASMOL: biomolecular graphics for all. *Trends in Biochemical Sciences*, 20(9):374–376.

[122] Scheidegger, C. E., Comba, J. L. D., and Da Cunha, R. D. (2005). Practical CFD Simulations on Programmable Graphics Hardware using SMAC†. *Computer Graphics Forum*, 24(4):715–728.

[123] Schrödinger, LLC (2015). The PyMOL molecular graphics system, version 1.8.

[124] Shanmugam, P. and Arikan, O. (2007). Hardware Accelerated Ambient Occlusion Techniques on GPUs. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 73–80, New York, NY, USA. ACM.

[125] Sharff, A. J., Rodseth, L. E., Spurlino, J. C., and Quiocho, F. A. (1992). Crystallographic evidence of a large ligand-induced hinge-twist motion between the two domains of the maltodextrin binding protein involved in active transport and chemotaxis. *Biochemistry*, 31(44):10657–10663.

[126] Sherman, W., Day, T., Jacobson, M. P., Friesner, R. A., and Farid, R. (2006). Novel procedure for modeling ligand/receptor induced fit effects. *Journal of Medicinal Chemistry*, 49(2):534–553.

[127] Sigg, C., Weyrich, T., Botsch, M., and Gross, M. (2006). Gpu-based ray-casting of quadratic surfaces. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, SPBG'06, pages 59–65, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[128] Sinko, W., Lindert, S., and McCammon, J. A. (2013). Accounting for receptor flexibility and enhanced sampling methods in computer-aided drug design. *Chemical Biology & Drug Design*, 81(1):41–49.

[129] Skånberg, R., Vázquez, P. P., Guallar, V., and Ropinski, T. (2016). Real-Time Molecular Visualization Supporting Diffuse Interreflections and Ambient Occlusion. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):718–727.

[130] Sourina, O., Torres, J., and Wang, J. (2009). Visual Haptic-Based Biomolecular Docking and Its Applications in E-Learning. In Pan, Z., Cheok, A. D., Müller, W., and Rhalibi, A. E., editors, *Transactions on Edutainment II*, Lecture Notes in Computer Science, pages 105–118. Springer Berlin Heidelberg, Berlin, Heidelberg.

[131] Spoel, D. V. D., Lindahl, E., Hess, B., Groenhof, G., Mark, A. E., and Berendsen, H. J. C. (2005). GROMACS: Fast, flexible, and free. *Journal of Computational Chemistry*, 26(16):1701–1718.

[132] Staib, J., Grottel, S., and Gumhold, S. (2015). Visualization of Particle-based Data with Transparency and Ambient Occlusion. *Computer Graphics Forum*, 34(3):151–160.

[133] Stamminger, M. and Drettakis, G. (2002). Perspective Shadow Maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 557–562, New York, NY, USA. ACM.

[134] Stocks, M. B., Laycock, S. D., and Hayward, S. (2011). Applying forces to elastic network models of large biomolecules using a haptic feedback device. *Journal of Computer-Aided Molecular Design*, 25(3):203–211.

[135] Stone, J. E., Sherman, W. R., and Schulten, K. (2016). Immersive Molecular Visualization with Omnidirectional Stereoscopic Ray Tracing and Remote Rendering. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1048–1057.

[136] Subaşı, E. (2006). Rigid molecular docking in virtual environments with haptic feedback. Master's thesis, Koç University.

[137] Subasi, E. and Basdogan, C. (2008). A New Haptic Interaction and Visualization Approach for Rigid Molecular Docking in Virtual Environments. *Presence*, 17(1):73–90.

[138] Suresh, P. S., Kumar, A., Kumar, R., and Singh, V. P. (2008). An insilco approach to bioremediation: Laccase as a case study. *Journal of Molecular Graphics and Modelling*, 26(5):845–849.

[139] Swails, J., Hernandez, C., Mobley, D., Nguyen, H., Wang, L., and Janowski, P. (2018). Parmed. http://parmed.github.io/ParmEd. Last checked on April 30, 2019.

[140] Takemura, K. and Kitao, A. (2012). Water model tuning for improved reproduction of rotational diffusion and NMR spectral density. *The Journal of Physical Chemistry*.

[141] Tarini, M., Cignoni, P., and Montani, C. (2006). Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE transactions on visualization and computer graphics*, 12(5):1237–1244.

[142] Tatsumi, R., Fukunishi, Y., and Nakamura, H. (2004). A hybrid method of molecular dynamics and harmonic dynamics for docking of flexible ligand to flexible receptor. *Journal of Computational Chemistry*, 25(16):1995–2005.

[143] Tsfadia, Y., Friedman, R., Kadmon, J., Selzer, A., Nachliel, E., and Gutman, M. (2007). Molecular dynamics simulations of palmitate entry into the hydrophobic pocket of the fatty acid binding protein. *FEBS letters*, 581(6):1243–1247.

[144] Turk, G. (1989). *Interactive collision detection for molecular graphics*. PhD thesis, The University of North Carolina at Chapel Hill.

[145] Vanommeslaeghe, K., Hatcher, E., Acharya, C., Kundu, S., Zhong, S., Shim, J., Darian, E., Guvench, O., Lopes, P., Vorobyov, I., and Mackerell, A. D. (2010). CHARMM general force field: A force field for drug-like molecules compatible with the CHARMM all-atom additive biological force fields. *Journal of Computational Chemistry*, 31(4):671–690.

[146] Venkatraman, V. and Ritchie, D. W. (2012). Flexible protein docking refinement using pose-dependent normal mode analysis. *Proteins*, 80(9):2262–2274.

[147] Wald, I., Ize, T., Kensler, A., Knoll, A., and Parker, S. G. (2006). Ray Tracing Animated Scenes Using Coherent Grid Traversal. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 485–493, New York, NY, USA. ACM.

[148] Walters, W. P., Stahl, M. T., and Murcko, M. A. (1998). Virtual screening—an overview. *Drug Discovery Today*, 3(4):160–178.

[149] Wan, P. T. C., Garnett, M. J., Roe, S. M., Lee, S., Niculescu-Duvaz, D., Good, V. M., Project, C. G., Jones, C. M., Marshall, C. J., Springer, C. J., Barford, D., and Marais, R. (2004). Mechanism of activation of the RAF-ERK signaling pathway by oncogenic mutations of b-RAF. *Cell*, 116(6):855–867.

[150] Wanger, L. (1992). The Effect of Shadow Quality on the Perception of Spatial Relationships in Computer Generated Imagery. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, I3D '92, pages 39–42, New York, NY, USA. ACM.

[151] Wanger, L. R., Ferwerda, J. A., and Greenberg, D. P. (1992). Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications 12(3)*, pages 44–58.

[152] Weikl, T. R. and Deuster, C. v. (2009). Selected-fit versus induced-fit protein binding: Kinetic differences and mutational analysis. *Proteins: Structure, Function, and Bioinformatics*, 75(1):104–110.

[153] Williams, L. (1978). Casting curved shadows on curved surfaces. In *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '78, pages 270–274. ACM.

[154] Wimmer, M., Scherzer, D., and Purgathofer, W. (2004). Light space perspective shadow maps. In *Proceedings of the Fifteenth Eurographics Conference on Rendering Techniques*, EGSR'04, pages 143–151, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[155] Wollacott, A. M. and Merz Jr., K. M. (2007). Haptic applications for molecular structure manipulation. *Journal of Molecular Graphics and Modelling*, 25(6):801–805.

[156] Wong, T. H., Leach, G., and Zambetta, F. (2014). An adaptive octree grid for GPU-based collision detection of deformable objects. *The Visual Computer*, 30(6-8):729–738.

[157] Yang, X., Xu, D.-q., and Zhao, L. (2009). A fast SAH-based construction of Octree.

[158] Yuriev, E., Agostino, M., and Ramsland, P. A. (2011). Challenges and advances in computational docking: 2009 in review. *Journal of Molecular Recognition*, 24(2):149–164.

[159] Zacharias, M. (2004). Rapid protein–ligand docking using soft modes from molecular dynamics simulations to account for protein deformability: Binding of FK506 to FKBP. *Proteins: Structure, Function, and Bioinformatics*, 54(4):759–767.

[160] Zacharias, M. and Sklenar, H. (1999). Harmonic modes as variables to approximately account for receptor flexibility in ligand–receptor docking simulations: Application to DNA minor groove ligand complex. *Journal of Computational Chemistry*, 20(3):287–300.

[161] Zhang, F., Sun, H., Xu, L., and Lun, L. K. (2006). Parallel-split Shadow Maps for Large-scale Virtual Environments. In *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*, VRCIA '06, pages 311–318, New York, NY, USA. ACM.

[162] Zonta, N., Grimstead, I. J., Avis, N. J., and Brancale, A. (2009). Accessible haptic technology for drug design applications. *Journal of Molecular Modeling*, 15(2):193–196.

Appendix A

Parallel Scan algorithms

In the grid construction algorithms described in Chapter 3, two different parallel scan algorithms are used: a parallel prefix sum, and a scan algorithm used to determine the minimum and maximum of an array. These algorithms make effective use of the parallel architecture of graphics processors.

For both algorithms, $\frac{n}{2}$ threads are launched, where n is equal to the number of items within the array to be scanned. Each thread then performs an operation on two elements, and stores the result in memory. The result is then used by another thread, which performs the same operation again with the result of the neighbouring calculation. This is then repeated until the algorithm is complete.

A.1 Parallel Prefix Sum

Figure A.1 shows how the core of the parallel prefix sum algorithm works. In row one, each thread adds two neighbouring elements together and stores the solution in shared memory. Then, in row two, half of the threads in the work group take the summations calculated in row one, and sum them together. This is repeated until the sum of the entire array is stored in the right most memory bank (Column 7 in Figure A.1). In the method used in this thesis, the value is then stored separately for use later.

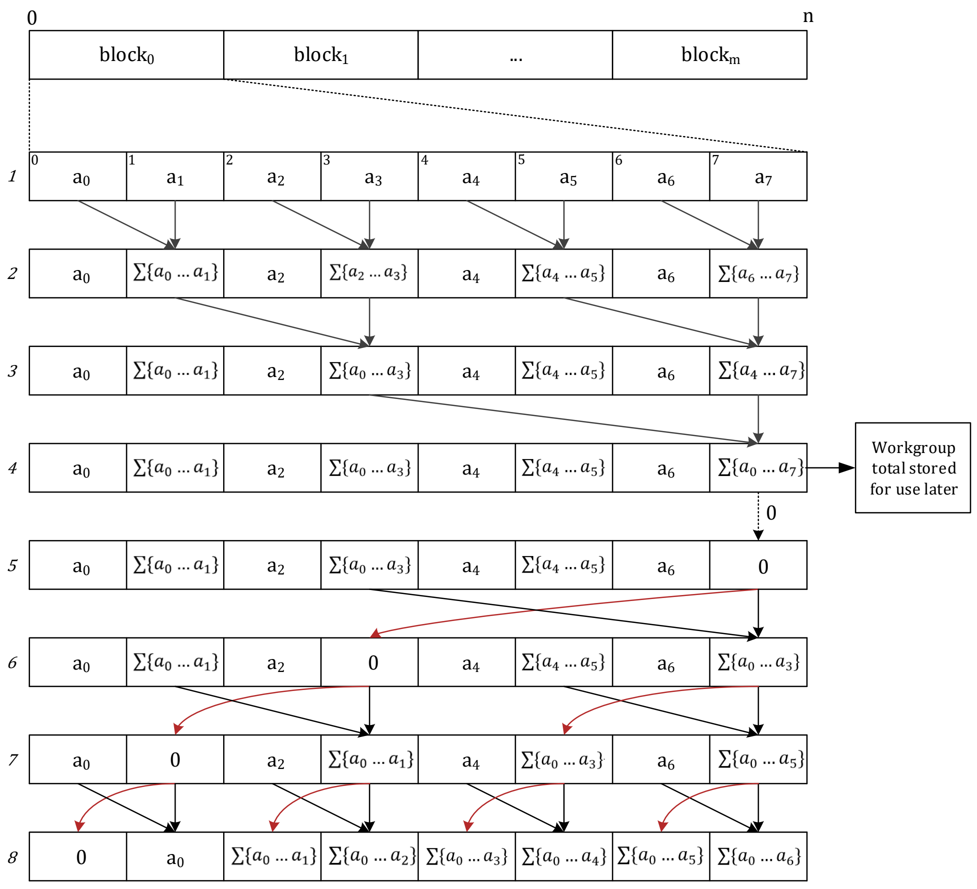


Fig. A.1 Graphic describing the parallel prefix sum calculation. Black arrows indicate an “add” operation has been performed, red arrows indicate a copy operation has been performed.

After the array total has been stored, the right most value is set to zero. Then, over the subsequent rows, the addition is repeated in reverse, with an offset that halves each time. Also at this stage, the solution to the previous summation is copied down the array with each summation, as indicated by the red arrows in Figure A.1.

When using shared memory, as is required to achieve good performance with a scan algorithm on the GPU, only threads within the same work group can access the values calculated by the other threads. As the work group size is limited to 1024 threads on most Nvidia GPUs, this necessitates that further kernels are required in order to complete the prefix sum calculation for the entire array.

In order to combine the results from each work group, the total for each segment of the array is saved (Line 4 in Figure A.1), before it is reset to zero. That value is

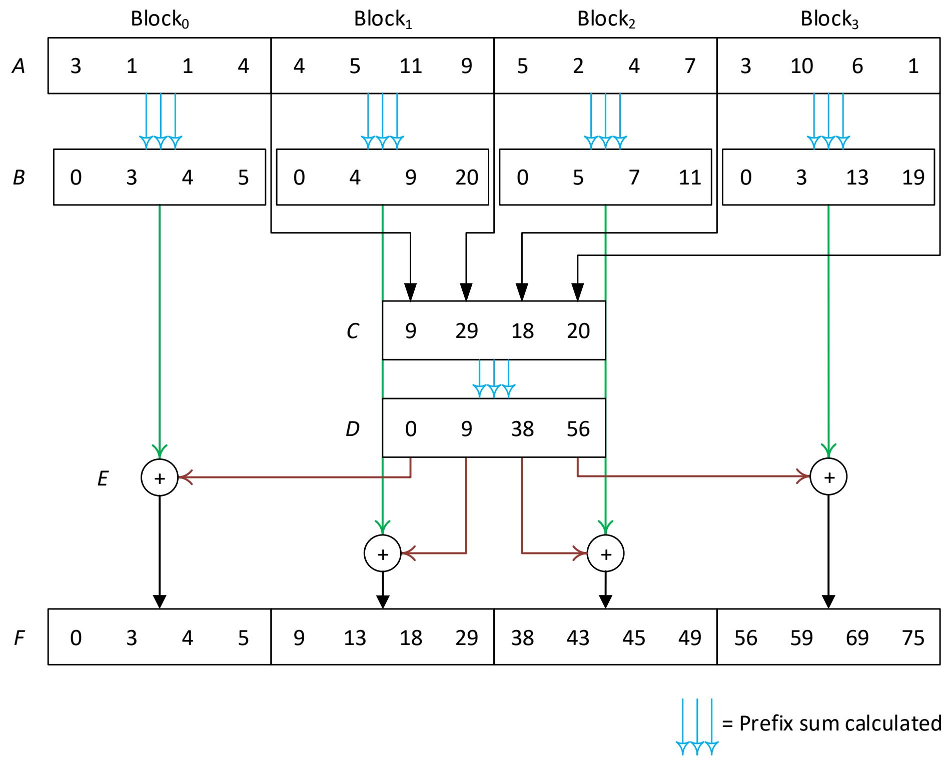


Fig. A.2 An overview of the parallel prefix sum algorithm, with a maximum work group size of 2. A: the un-summed array; B: the array summed at a work group level; C: the array of work group totals; D: The prefix sum of the work group total array; E: The prefix sum of the totals is added to work group level values; F: The prefix sum of the starting array. A group of three arrows indicates that the prefix sum is calculated.

then stored in a global memory, in an array of work group totals (Figure A.2, line C). The prefix sum of this work group totals array is then calculated (Figure A.2, line D). The result is the value each work group (from the original summation), needs to add to each element in that portion of the array, in order to determine the prefix sum for the full array (Figure A.2, line F).

In the scenario depicted in Figure A.2, the maximum length of an array that can be summed is 2048^2, with a work group size limit of 1024. If an array longer than this needed to be summed, a further reduction step would be required.

A.2 Minimum and Maximum

To find the minimum and maximum of an array, a scan algorithm is also used. Figure A.3 shows how the algorithm works at a work group level. In the initial step, each thread compares two values, and moves the larger to the right hand side of the array, and the smaller to the left.

Then, each thread compares two further values, and if the “left” value is larger than the “right” value, they are swapped. In the subsequent rows this is repeated with an offset that increases with each reduction. The number of threads performing comparisons is halved at each level, until only two are left. At this point, a final comparison is performed. The largest element in the array will be the right most value, and the smallest value will be the left most value.

As with the parallel prefix sum, an additional kernel is needed to combine the results of each work group. Therefore, the largest and smallest values from each work group are copied into a separate, intermediate array. The minimum and maximum of the intermediate array is then found, giving the overall minimum and maximum of the original array.

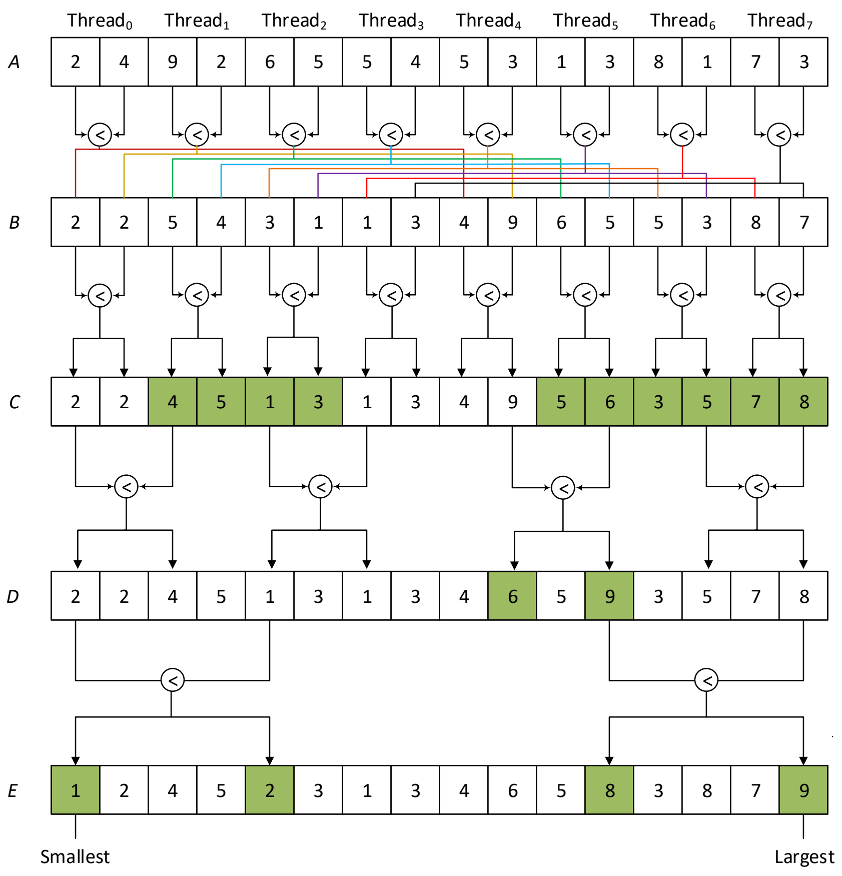

Thread$_0$ Thread$_1$ Thread$_2$ Thread$_3$ Thread$_4$ Thread$_5$ Thread$_6$ Thread$_7$
A 2 4 9 2 6 5 5 4 5 3 1 3 8 1 7 3
B 2 2 5 4 3 1 1 3 4 9 6 5 5 3 8 7
C 2 2 4 5 1 3 1 3 4 9 5 6 3 5 7 8
D 2 2 4 5 1 3 1 3 4 6 5 9 3 5 7 8
E 1 2 4 5 2 3 1 3 4 6 5 8 3 8 7 9
Smallest
Largest

Fig. A.3 work group level depiction of the minimum and maximum algorithm. (A) original array for which the minimum and maximum is required; (B) smaller values have been moved to the left hand side of the array, larger values moved to the right; (C) Each thread compares a pair of values. If the RHS is smaller than the LHS, they are swapped, as indicated by a green background; (D) The smallest values from (C) are compared, and swapped again if necessary; (E) Final array - the smallest value is furthest left, the largest furtherest to the right; In the diagram, the work group comprises 8 threads. In reality, it comprises of 512 or 1024 threads, necessitating many more levels per work group.

Appendix B

Lighting Fragment shader

```

//Calculates lighting for the spheres - textures for
    positions, colours and normals are filled in the
    geometrypass.

#version 430

in vec3 ex_LightDir;
//textures constructed from the geometry pass to store
    positions, colours and normals
uniform sampler2D diffuseTexture;
uniform sampler2D positionsTexture;
uniform sampler2D normalsTexture;
in vec2 ex_TexCoord;

//lighting properties
uniform vec4 light_ambient;
uniform vec4 light_diffuse;
uniform vec4 light_specular;

//regular grid
uniform float gridCellSize;
uniform int totalAtomCount;
uniform int finalCell;
uniform vec3 gridMin;
uniform ivec3 gridNumBoxes;
uniform float aoIntensity;
uniform int maxOccluders;

//model view matrix for the protein
uniform mat4 mvMatrix;

//output colour.
out vec4 colour;


//ssbo structures
layout(std430, binding = 1) buffer grid_data
{
  float cellId_ssbo[];
};

layout(std430, binding = 2) buffer sphere_positions
{
  float spherePositions_ssbo[];
};
```

```

layout(std430, binding = 3) buffer cell_lookup
{
  int cellStartPoint[];
};

layout(std430, binding = 4) buffer ao_data
{
  float aoDATA[];
};

void ToLocal(in vec3 p, in mat4 matrix, inout vec3 newP)
{
  float a = matrix[0][0], b = matrix[0][1], c = matrix[0][2];
  float d = matrix[1][0], e = matrix[1][1], f = matrix[1][2];
  float g = matrix[2][0], h = matrix[2][1], j = matrix[2][2];
  float k = matrix[3][0], l = matrix[3][1], m = matrix[3][2];
  newP.x = a*p.x + b*p.y + c*p.z + (a*-k + b*-l + c*-m);
  newP.y = d*p.x + e*p.y + f*p.z + (d*-k + e*-l + f*-m);
  newP.z = g*p.x + h*p.y + j*p.z + (g*-k + h*-l + j*-m);
}

void ToLocalRay(in vec3 p, in mat4 matrix, inout vec3 newP)
{
  float a = matrix[0][0], b = matrix[0][1], c = matrix[0][2];
  float d = matrix[1][0], e = matrix[1][1], f = matrix[1][2];
  float g = matrix[2][0], h = matrix[2][1], j = matrix[2][2];
  float k = matrix[3][0], l = matrix[3][1], m = matrix[3][2];
  newP.x = a*p.x + b*p.y + c*p.z;
  newP.y = d*p.x + e*p.y + f*p.z;
  newP.z = g*p.x + h*p.y + j*p.z;
}

vec3 getCellMinPoint(int x, int y, int z)
{
  vec3 cellBoundsMin;
  cellBoundsMin.x = (x * gridCellSize) + gridMin.x;
  cellBoundsMin.y = (y * gridCellSize) + gridMin.y;
  cellBoundsMin.z = (z * gridCellSize) + gridMin.z;
  return cellBoundsMin;
}

vec3 getCellMaxPoint(int x, int y, int z)
{
  vec3 cellBoundsMax;
  cellBoundsMax.x = ((x + 1) * gridCellSize) + gridMin.x;
  cellBoundsMax.y = ((y + 1) * gridCellSize) + gridMin.y;
  cellBoundsMax.z = ((z + 1) * gridCellSize) + gridMin.z;
  return cellBoundsMax;
}

bool insideGrid(int x, int y, int z)
{
  return ((x >= 0) && (y >= 0) && (z >= 0)) && ((x <
    gridNumBoxes.x) && (y < gridNumBoxes.y) && (z <
    gridNumBoxes.z));
}

void IntersectRayAABB(vec3 ro, vec3 rd, vec3 gridMin, vec3
    gridMax, out float t0)
{
  vec3 invR = 1.0 / rd;
  vec3 tbot = invR * (gridMin - ro);
  vec3 ttop = invR * (gridMax - ro);
  vec3 tmin = min(ttop, tbot);
  vec2 t = max(tmin.xx, tmin.yz);
  t0 = max(t.x, t.y);
}
```

```glsl

struct LightingResults
{
  vec4 ambient, diffuse, specular;
};

//calculates if the ray intersects with any spheres located in the grid cell specified by gridCellIndexID
void softShadowGrid(vec3 ro, vec3 rd, float sphereID, int gridCellIndex1D, inout float s)
{

//based on the gridCellIndex1D passed in - this could be swapped out for a different data structure.
  int accessId;
  int loopLimit;
  int readId;

  accessId = cellStartPoint[gridCellIndex1D];
  if (gridCellIndex1D + 1 == finalCell){
    loopLimit = totalAtomCount - accessId;
  }
  else {
    loopLimit = cellStartPoint[gridCellIndex1D + 1] - accessId;
  }

  for (int i = 0; i < loopLimit; i++)
  {
    float shadowSphereID;
    shadowSphereID = cellId_ssbo[accessId];
    accessId++;

    if(shadowSphereID < -0.5)  //gridDataTexture includes values of -1 when the index list ends
      break;                  //break the for loop as there are no more atoms in this grid cell

    vec4 sphere;
    int lookupId = int(shadowSphereID) * 4;
    sphere.x = spherePositions_ssbo[lookupId];
    sphere.y = spherePositions_ssbo[lookupId + 1];
    sphere.z = spherePositions_ssbo[lookupId + 2];
    sphere.w = spherePositions_ssbo[lookupId + 3];

    vec4 sphereEye = mvMatrix * vec4(sphere.xyz, 1.0);
    float t0 = dot((sphereEye.xyz - ro), rd);
    float b = 0.020;
    float R = sphere.w - b;
    float d = length((t0 * rd) - sphereEye.xyz + ro);

    if(t0 < 0.0){
      s = min(s, 1.0);
    } else {
      if(d < R){
        s = 0.0;
        break;
      } else if(d > sphere.w){
        s = min(s, 1.0);
      } else {
        s = min(s, smoothstep(0.0, 1.0, (d - R) / b));
      }
    }
  }
}

void traverseGrid(vec3 ro, vec3 rd, float sphereID, inout float s)
{
```

```
vec3 rayOrigin;// = (inverse(mvMatrix) * vec4(ro,1.0)).xyz;
vec3 rayDir;

// transform ro and rd into grid's local space
ToLocal(ro, mvMatrix, rayOrigin);
ToLocalRay(rd, mvMatrix, rayDir);

int indexX = int((rayOrigin.x - gridMin.x) / gridCellSize);
int indexY = int((rayOrigin.y - gridMin.y) / gridCellSize);
int indexZ = int((rayOrigin.z - gridMin.z) / gridCellSize);

// if the ray origin doesn't start inside the grid, adjust
  rayOrigin so it does
if(!insideGrid(indexX, indexY, indexZ))
{
  float t = 0.0;
  vec3 gridMax = gridMin + (gridCellSize * gridNumBoxes);

  IntersectRayAABB(rayOrigin, rayDir, gridMin, gridMax, t);
  rayOrigin += rayDir * t;

  indexX = int((rayOrigin.x - gridMin.x) / gridCellSize);
  indexY = int((rayOrigin.y - gridMin.y) / gridCellSize);
  indexZ = int((rayOrigin.z - gridMin.z) / gridCellSize);
}

float tDeltaX = abs(gridCellSize / rayDir.x);
float tDeltaY = abs(gridCellSize / rayDir.y);
float tDeltaZ = abs(gridCellSize / rayDir.z);

int stepX = 1;
int stepY = 1;
int stepZ = 1;

if(rayDir.x < 0.0)
  stepX = -1;

if(rayDir.y < 0.0)
  stepY = -1;

if(rayDir.z < 0.0)
  stepZ = -1;

vec3 cellBoundsMin = getCellMinPoint(indexX, indexY, indexZ
  );
vec3 cellBoundsMax = getCellMaxPoint(indexX, indexY, indexZ
  );

float tMaxNegX = (cellBoundsMin.x - rayOrigin.x) / rayDir.x
  ;
float tMaxNegY = (cellBoundsMin.y - rayOrigin.y) / rayDir.y
  ;
float tMaxNegZ = (cellBoundsMin.z - rayOrigin.z) / rayDir.z
  ;

float tMaxPosX = (cellBoundsMax.x - rayOrigin.x) / rayDir.x
  ;
float tMaxPosY = (cellBoundsMax.y - rayOrigin.y) / rayDir.y
  ;
float tMaxPosZ = (cellBoundsMax.z - rayOrigin.z) / rayDir.z
  ;

float tMaxX = rayDir.x < 0 ? tMaxNegX : tMaxPosX;
float tMaxY = rayDir.y < 0 ? tMaxNegY : tMaxPosY;
float tMaxZ = rayDir.z < 0 ? tMaxNegZ : tMaxPosZ;

bool done = false;
while(!done)
{
    int gridCellIndex1D = (indexZ * gridNumBoxes.x * gridNumBoxes.y) + (indexY * gridNumBoxes.x) + indexX;
    softShadowGrid(ro, rd, sphereID, gridCellIndex1D, s);

    if(tMaxX < tMaxY)
    {
      if(tMaxX < tMaxZ)
      {
        indexX += stepX;
        tMaxX += tDeltaX;
      } else {
        indexZ += stepZ;
        tMaxZ += tDeltaZ;
      }
    } else {
      if(tMaxY < tMaxZ)
      {
        indexY += stepY;
        tMaxY += tDeltaY;
      } else {
        indexZ += stepZ;
        tMaxZ += tDeltaZ;
      }
    }
    //this loop might be able to terminate early if the shadow is found.
    done = !insideGrid(indexX, indexY, indexZ) || (s < 0.01);
  }
}

void directional(in float ao, in vec4 diffuseTexel, in vec4 positionTexel, in vec4 normalTexel, inout LightingResults colour)
{
  LightingResults col;
  col.ambient = vec4(0.0);
  col.diffuse = vec4(0.0);
  col.specular = vec4(0.0);

  float sphereID = positionTexel.w;
  vec3 n = normalTexel.xyz;

  // light direction
  vec3 l = normalize(ex_LightDir.xyz);

  float NdotL = max(0.0, dot(n, l));

  // ambient
  col.ambient += ao * diffuseTexel * light_ambient * (normalTexel.z + 1.0);

  if (NdotL > 0.0)
  {
    // diffuse
    col.diffuse += (diffuseTexel * light_diffuse) * NdotL;

    // specular
    vec3 v = normalize(-positionTexel.xyz);
    vec3 r = normalize(-reflect(l, n));
    float RdotV = max(0.0, dot(r, v));
    col.specular += 0.8 * light_specular * pow(RdotV, 80.0);

    float s = 1.0;
    traverseGrid(positionTexel.xyz, l, sphereID, s);
    col.diffuse *= s;
    col.specular *= s;
  }

  colour.ambient += col.ambient;
  colour.diffuse += ao * col.diffuse;
  colour.specular += ao * col.specular;
}

void aoOnly(in int index, in float ao, in vec4 diffuseTexel,
    in vec4 positionTexel, in vec4 normalTexel, inout
    LightingResults colour)
{
  colour.ambient = ao * diffuseTexel * 1.2 * (1.0 + (0 * 0.5)
    );
}

float calcAO(vec3 pos, float sphereID, vec3 norm)
{
  //if sphere does not exsist, return default value.
  if (sphereID < -0.5)
    return 0.7f;

  //determine lookup index of occluder list
  int id = int(sphereID) * maxOccluders;
  float ao = 0.0;

  for (int i = 0; i < maxOccluders; i++)
  {
    //load occluder sphere ID.
    int occluderSphereID = int(aoDATA[id + i]);

    //if ID < 1, all occluders tested, break loop.
    if (occluderSphereID < -0.5){
      if (i == 0){ //If no occluders in list, return constant
    .
        return 0.7f;
      }
      break;
    }

    //load occluding sphere from SSBO.
    vec4 sphere;
    int lookupId = int(occluderSphereID) * 4;
    sphere.x = spherePositions_ssbo[lookupId];
    sphere.y = spherePositions_ssbo[lookupId + 1];
    sphere.z = spherePositions_ssbo[lookupId + 2];
    sphere.w = spherePositions_ssbo[lookupId + 3];

    //position sphere in space.
    vec4 sphereEye = mvMatrix * vec4(sphere.xyz, 1.0);

    //calculate vector between occluding and original sphere.
    vec3 dir = pos - sphereEye.xyz;
    float len = length(dir);
    float sphereLen = sphere.w / len;
    float sqrtV = 1.0 - (sphereLen * sphereLen);
    sqrtV = (sqrtV < 0) ? 0 : (sqrtV > 1) ? 1 : sqrtV;

    //Determine occlusion contribution (Section 4.3.3)
    float t = dot(norm, dir);
    float AOProportion = (clamp((t + sphere.w), 0.0, 2 *
   sphere.w)) / (2 * sphere.w);
    ao += 1.0 - (AOProportion * sqrt(sqrtV));
    }
  //calculate final occlusion
  return clamp(1.0 - (ao * aoIntensity), 0.0, 1.0);
}

void main()
{

  vec4 positionTexel  = texture(positionsTexture, ex_TexCoord
   );
  vec4 normalTexel = texture(normalsTexture, ex_TexCoord);
  vec4 diffuseTexel = texture(diffuseTexture, ex_TexCoord);

  LightingResults colourRes;
  colourRes.ambient = vec4(0.0);
  colourRes.diffuse = vec4(0.0);
  colourRes.specular = vec4(0.0);

  if (positionTexel.xyz == vec3(1.0))
  {
    colour.rgb = vec3(1.0);
    colour.a = float(positionTexel.xyz != vec3(1.0));
    return;
  } else {
    float ao = calcAO(positionTexel.xyz, positionTexel.w+0.5,
    normalTexel.xyz) ;
    directional(ao, diffuseTexel, positionTexel, normalTexel,
    colourRes);
  }

  colour = colourRes.ambient +colourRes.diffuse +colourRes.
   specular;
}
```