TU WIEN Informatics

Constraint-basierte 3D Manipulation für molekulare Modellierung im Web

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Lucas da Cunha Melo
Matrikelnummer 01429462

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Assistant Prof. Dr.techn. MSc Manuela Waldner

Wien, 4. September 2023

Lucas da Cunha Melo | Manuela Waldner

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Constraint-Based 3D Manipulation for Molecular Modelling on the Web

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Lucas da Cunha Melo
Registration Number 01429462

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dr.techn. MSc Manuela Waldner

Vienna, 4th September, 2023

Lucas da Cunha Melo   Manuela Waldner

Erklärung zur Verfassung der Arbeit

Lucas da Cunha Melo

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. September 2023

Lucas da Cunha Melo

Kurzfassung

Computer-Aided Molecular Design (MolCAD), oder molekulare Modellierung, beschäftigt sich mit dem computergestützten Entwurf und der Manipulation von molekularen Strukturen. Dieses Gebiet erlebt derzeit einen Aufschwung an Interesse und Entwicklung, wobei der Fokus oft auf fortschrittlichen Visualisierungstechniken liegt. Allerdings mangelt es den bestehenden MolCAD-Tools oft an der Benutzerfreundlichkeit und effizienten Interaktion, die in traditioneller CAD-Software üblich sind. Diese Arbeit befasst sich mit dieser Lücke durch drei Methoden: (1) eine Untersuchung der etablierten CAD- und MolCAD-Literatur und -Software, (2) die Implementierung identifizierter vielversprechender Interaktionstechniken und (3) Case Studies zu deren Validierung.

Im Rahmen dieses Prozesses werden zwei Interaktionstechniken in einer webbasierten Umgebung implementiert: ein PCA-basiertes Ausrichtungstool und ein Echtzeit-Kollisionserkennungssystem. Die Entscheidung, diese Tools für das Web zu implementieren, wurde getroffen, um eine einfache Zugänglichkeit und Bereitstellung auf verschiedenen Plattformen zu ermöglichen, da keine Installation erforderlich ist.

Die durchgeführten Case Studies zielten darauf ab, diese beiden implementierten Ansätze zu validieren. Das Echtzeit-Kollisionserkennungssystem erhielt positives Feedback und zeigte großes Potenzial, den MolCAD-Prozess weniger frustrierend und effizienter zu gestalten. Das PCA-basierte Ausrichtungstool erhielt jedoch gemischte Feedbacks, was auf Bereiche für zukünftige Arbeiten hindeutet. Dennoch demonstrieren beide Funktionen das Potenzial, die Benutzerzufriedenheit und Effizienz in MolCAD zu verbessern.

Abstract

Computer-Aided Molecular Design (MolCAD), or molecular modelling, is the computational design and manipulation of molecular structures. This field is experiencing a surge in interest and development, where the focus is often on advanced visualization techniques. However, existing MolCAD tools often lack the level of usability and efficient interaction commonly found in traditional CAD software. This thesis addresses this gap through three methods: (1) a survey of established CAD and MolCAD literature and software, (2) the implementation of identified promising interaction techniques; and (3) case studies to validate their effectiveness.

As a result of this process, two interaction techniques are implemented in a web-based environment: a PCA-based alignment tool, and a real-time collision detection system. The decision to implement these tools for the web was made with the aim to provide ease of accessibility and deployment across various platforms, as no installation is required.

The case studies conducted were aimed at validating these two implemented approaches. The real-time collision detection system received positive feedback, and showed great potential to make the MolCAD process less frustrating and more efficient. The PCA-based alignment tool, however, received mixed responses, indicating areas for future work. Nonetheless, both features demonstrate the potential to improve user satisfaction and efficiency in MolCAD.

Contents

CHAPTER 1

Introduction

The topic of Molecular Modelling has recently gone through an increasing growth in interest and development. Modern graphics hardware, combined with newer algorithms and advancements in nanotechnology led and continues to lead to a more widespread usage of *in-silico* protein/DNA design. Tools such as Adenita [dLMA+20] and oxDNA [vRO+12] show how far this topic has come, with advanced multiscale visualization of nanostructures and web tools for Molecular Dynamics Simulations (MDS) respectively, among other advancements (analysed further in Chapters 2 and 3).

Biotechnology is vulnerable to disruption in the next decades due to its lack of digital transformation [SPG18]. As user interviews conducted at early stages of the work presented in this thesis have shown, one area is most crucially lagging behind when it comes to the modelling of protein/DNA nanostructures: advanced interaction techniques. We observed and queried experienced biotechnologists working in the field of DNA/protein nanotechnology to find out about their difficulties in their *in-silico* work despite the available advanced technology. We concluded that one crucial reason holding these tools back from becoming more widespread is a lack of usability.

Another aspect that emerged in the preliminary user interviews is the difficulty of setting up Computer-Aided Molecular Design (MolCAD) systems. The majority of them require the software's installation in the user's computer, which is not only an extra (sometimes cumbersome) step, but may result in incompatibilities with the user's operating system. Moreover, installing software in corporate hardware often requires the approval of an information technology (IT) or cybersecurity department. A solution to these issues, and therefore a central point in this thesis, is the implementation of MolCAD tools in a web-based environment.

While Computer-Aided Design (CAD) systems have long been the subject of extensive usability research (explored further in Section 4.3), the same cannot be said for MolCAD.

Usability in MolCAD presents its own set of challenges, distinct from those in traditional CAD. Several factors contribute to this increased complexity:

1. The intricate nature of molecular structures, often defined by their low-level atomic structure, while requiring higher level interaction depending on the specific domain (e. g., DNA/protein-hybrid structures, DNA origami structures).
2. Traditional CAD software for engineering and architecture may have a natural point of reference (such as the floor, a room, or the surface of other objects) that allows for straightforward modelling. Modelling and movement in MolCAD software, on the other hand, has no up-vector, and thus no immediately obvious point of reference. Moreover, in CAD software, objects in a 3D scene are typically inherently hierarchical, such as a window belonging to a building, or a gear belonging to a machine.
3. Atomic structures are only loosely related to each other, at most forming higher-level structures together (nucleotides form nucleic acid strands, which form DNA; or amino acids form proteins, etc.)
4. Atomic structures are very densely packed and in many cases must be tightly and precisely aligned to each other, or else the entire structure may not be suitable for further steps in a validation pipeline, such as MDS.
5. One of the most fundamental interaction principles in traditional CAD is constraint-based interactions [ASF+13, SMS+16]. In order to constrain MolCAD interactions in a similar way, it is necessary to detect (and perhaps correct) collisions between molecular structures. This creates the need for efficient collision calculations that can be performed in real-time. This is particularly challenging for web environments, as widely established computation tools are becoming obsolete, lacking support for state-of-the-art general-purpose GPU computations.

Research Questions Observing the potential for improvement in MolCAD software and its challenges, this master's thesis aims to explore, design, and implement novel ways to perform 3D modelling in the MolCAD context. More specifically, we aim to answer the following research questions:

RQ1. Which CAD interaction techniques exist that are also applicable for MolCAD and are currently not in use yet, but could be useful?

RQ2. How can the novel MolCAD constraint-based interaction methods be implemented efficiently (in real-time) in a web-based environment?

RQ3. Which of the novel MolCAD constraint-based interaction methods increases user satisfaction and efficiency for constraint-based MolCAD?

Methodology To implement the technical contribution of this thesis while keeping user satisfaction (a qualitative measure) in a central position, the design study methodology [SMM12] is used. This methodology is a structured approach commonly used in visualization research to address specific real-world problems faced by domain experts. It involves a series of steps that begin with the analysis of the research problem, followed by the design of a visualization system tailored to that problem. The methodology emphasises the importance of validating and refining the design with real users and real data. As for the technical aspects of this thesis, Catana [KMS^{+}22] (a web-based MolCAD framework) is at the core of this work. The algorithms presented throughout this thesis are implemented on top of it. To validate them, benchmarks are performed.

Contribution In summary, the contribution of this master's thesis is two-fold: (i) to achieve a deeper understanding of useful interaction techniques facilitating MolCAD (RQ1., design contribution); and (ii) a prototypical implementation, demonstrating that a real-time web-based MolCAD implementation is technically feasible, even with the additional challenge of integrating newer computing technologies (namely WebGPU [Webb]) on top of an older standard (WebGL [Weba]) (RQ2., technical contribution). These two contributions are further qualitatively evaluated, to ensure their robustness (RQ3.). Together, they provide a blueprint for future MolCAD tools.

This thesis is structured as follows: Chapters 2 and 3 respectively explain introductory concepts necessary to understand this thesis and offer a brief discussion of existing technical solutions for MolCAD. Each research question is then explored and answered in a respective chapter. Chapter 4 answers RQ1., Chapter 5 answers RQ2., and Chapter 6 answers RQ3.. A final discussion then takes place in Chapter 7, followed by a summary of potential future work.

CHAPTER 2

Background

Before delving deeper into the related work, methods, and results of this thesis, biological and software concepts must be presented. The following sections give a brief introduction to these concepts.

The biological part starts with an introduction to nucleic acids (more specifically, DNA and RNA) and proteins. The uses of these components to build nanostructures are also described, together with a short description of the work of the biotechnologists designing and synthesizing these structures.

The software part starts with an introduction of MolCAD software. Finally, MDS is briefly explained, as it is often the next step after molecular modelling in the work pipeline of biotechnologists.

2.1 Nanostructures and Their Building Blocks

In biotechnology, nanostructures are essentially molecular constructs made up of biological molecules. Among the key components used to build them are DNA, RNA, and proteins. They are a crucial part of MolCAD, as they are high-level building blocks for nanostructures.

In this thesis, the term "nanostructures" refers to functional DNA-protein hybrid molecular assemblies. In this section, fundamental aspects of nucleic acids (specifically DNA and RNA) as well as proteins are briefly explained.

2.1.1 Proteins

Proteins comprise amino acid chains, and are essential building blocks for nanostructures. The amino acids are held together in a chain by what is called the protein's "backbone", which helps provide the protein's function and the structure upon which the protein is built. An amino acid chain has two ends, which are called the N-terminus and the C-terminus. All of these components are illustrated in Figure 2.1.

A protein's highly specific binding sites enable targeted interactions, making them crucial for applications like drug delivery and diagnostics. Additionally, proteins have the capability to fold into intricate three-dimensional shapes, self-assemble, and alter their conformation in response to environmental changes, all reducing the need for external manipulation. Because of that, they may serve various roles within a nanostructure from structural elements to enzymes. Finally, their biocompatibility minimises the risks of adverse immune responses, making them ideal for medical applications.

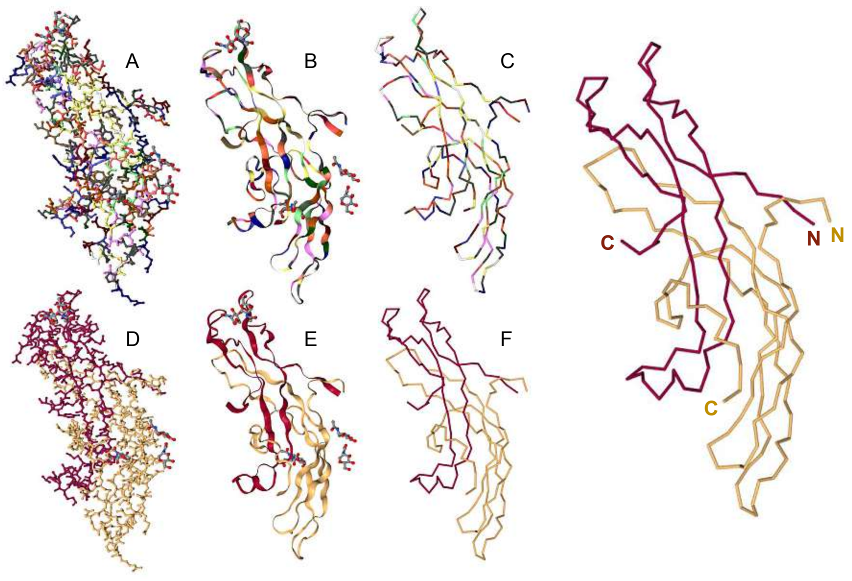


Figure 2.1: The protein 1HRP from the PDB is visualised with different representations and different colour schemes. The left column (A, D) uses the "ball-and-stick" representation, the middle column (B, E) uses the "cartoon" representation, and the right column (C, F) uses the "backbone" representation (where the tube goes through the protein's backbone). In the top row (A, B, C), the proteins are coloured according to their amino acids, while in the bottom row (D, E, F), the proteins are coloured according to their chains (this protein contains two chains, represented here in yellow and red). In the enlarged view on the right, which has the same representation and colour scheme as F, the N- and C-termini are labelled.

2.1.2 Nucleic Acids

Nucleic acids (primarily DNA and RNA) are large biomolecules fundamental for the storage and transfer of genetic information in cells. They consist of strands of nucleotides (analogous to a protein's chain of amino acids) and follow base pairing rules (C with G, and A with T – or A with U for RNA).

Atoms and molecules, in general, have the intrinsic property of self-organisation. This is a fundamental property that biologists take advantage of to create nanostructures. What makes nucleic acids special in this regard is the predictable nature in which they assemble. This property was first demonstrated to be able to produce 2D rasters [Rot06], and shortly after, demonstrated to also be able to produce 3D shapes [DDL$^+$09].

This process and its resulting nanostructure are called DNA origami, currently one of the most popular approaches for the design of DNA nanostructures [dLMA$^+$20]. The name comes from the fact that the DNA is designed to fold into desired shapes. One example of such a DNA origami structure is the one presented by Ahmadi [ANW$^+$20], where a (nanomechanical) DNA rotor is developed. Such structures are becoming increasingly popular due to their potential application in fields including nanocomputing, robotics, and drug delivery.

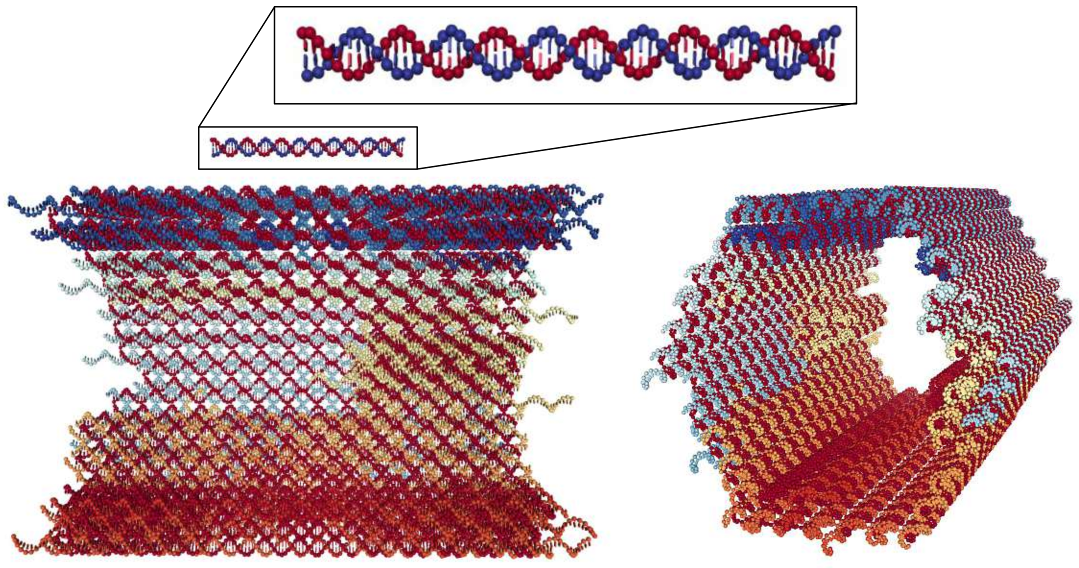


Figure 2.2: On top, for scale, a DNA strand with 118 nucleotides (59 per strand). Below it, a DNA origami structure with 16,670 nucleotides is shown. To the right, the same DNA origami structure is shown, but combined with a second DNA origami structure to form a larger structure with 32,383 nucleotides.

2.2 Software

Several software tools have been developed to facilitate the design of DNA-only nanostructures. One prominent example is caDNAno [DMT+09], and its web-based variant scaDNAno [DLS20]. These platforms can be integrated with molecular dynamics/relaxation software, such as oxDNA [vRO+12] (shown in Figure 2.3), to simulate their folding behaviour before synthesis in the laboratory.

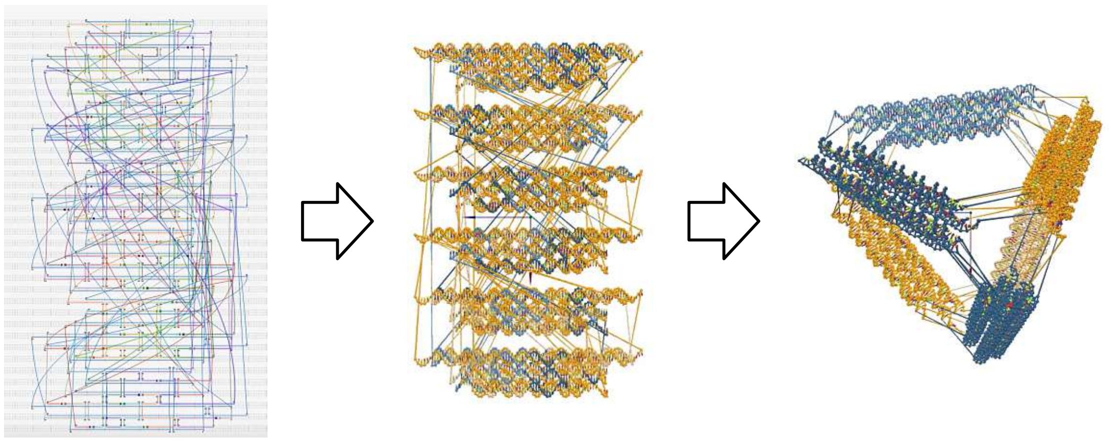


Figure 2.3: A DNA origami structure assembled in caDNAno [DMT+09] (left) is imported into oxDNA [vRO+12] (centre), where it undergoes a spring-force-based rigib-body relaxation process to assemble into its final tridimensional shape (right). Images taken from `https://www.youtube.com/watch?v=bwmUpTdrXdk` (accessed 2023-09-04).

However, when considering the integration of proteins, there is a lack of comprehensive tools that support the design of protein/DNA-hybrid nanostructures. Traditional platforms, such as PyMol [SD], are antiquated, with algorithms that lack the required performance to execute essential computations without disrupting the user's flow of thought [Nie93], like collision detection between atoms (found in this thesis to be a crucial component of MolCAD). Furthermore, these tools require command-line input for basic operations, which can be cumbersome and inefficient, especially compared with the advancements experienced in the field of general CAD.

2.2.1 Molecular Dynamics Simulations

MDS are computational techniques for studying the physical movements and interactions of atoms and molecules over time. This capability is particularly useful for gaining insights into the structural and dynamic properties of molecular structures without the need to synthesise them in a laboratory environment. The simulations work by solving equations of motion for a system of interacting particles. This enables researchers to model complex biological phenomena, including protein folding, ligand binding, and enzymatic reactions, all crucial for the application of the nanostructures assembled with MolCAD.

CHAPTER 3

Existing MolCAD Software

This chapter provides an overview and discussion of existing MolCAD software, graphics processing on the web, and collision detection algorithms . A more in-depth analysis of further related work and the aspects of CAD and MolCAD software most relevant to this work is provided in Chapter 4, where a survey on academic contributions to CAD is also available (in Section 4.3).

MolCAD and Visualization Software have seen widespread use since at least the early 2000s. PyMol [SD] is one of the first such tools. It enables modelling and visualization, much of it via written commands and scripting. Chimera [PGH+04], which is equivalent to PyMol, also has widespread use and was released 4 years later in 2004. Similarly, it does not provide modern interaction capabilities like modern CAD software (such as basic translation/rotation gizmos, illustrated in Figure 3.1).

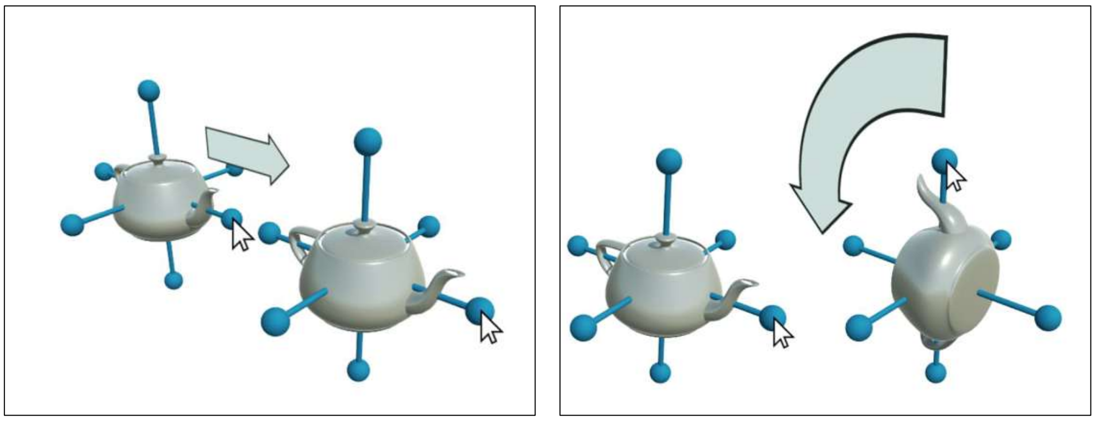


Figure 3.1: Translation and rotation gizmos have become a standard tridimensional widget in CAD applications. Image taken from [MCG+18].

SAMSON Connect [SAM] is a modern tool that, among other uses, provides tools for MolCAD. However, as stated by biotechnogists during interviews conducted in early stages of the work presented in this thesis (described in Section 4.1), it is currently not as widespread as PyMol and Chimera. It supports fine-grained control over the creation and editing of molecules, as shown in Figure 3.2. It also uses constraints to add atoms, where freely added atoms are placed on a fixed plane and further atoms can be created with a bond to it by clicking on hydrogen atoms bound to previously placed atoms.

Three reasons came up in the interviews as to why it did not gain popularity with domain experts: (1) the already widespread presence of PyMol and Chimera provides easy support from peers and early familiarity with these tools (starting in university courses), (2) the fact that they are open-source (while SAMSON is proprietary), and (3) difficulty of setting up the software (which this thesis aims to overcome by providing a web tool).

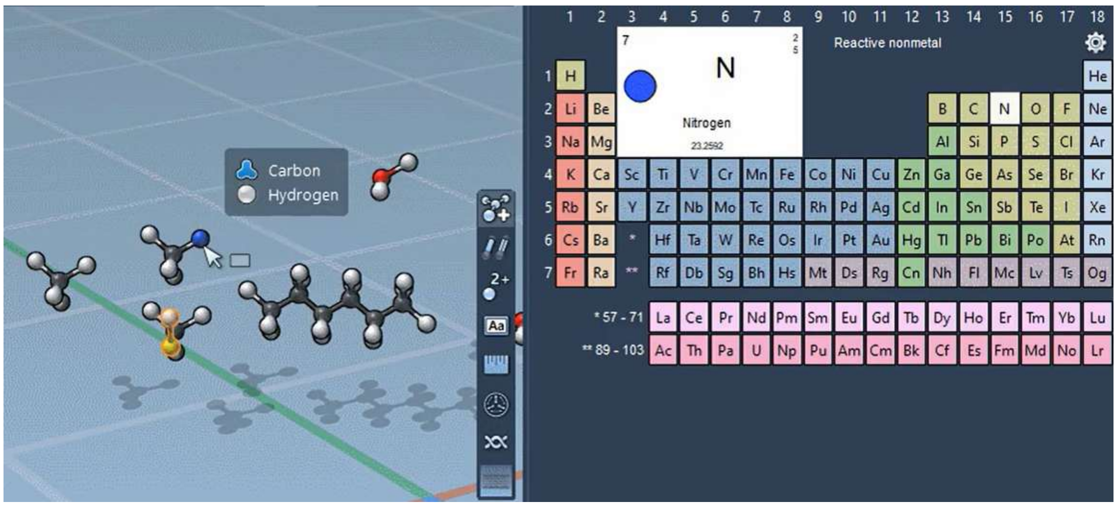


Figure 3.2: SAMSON Connect [SAM] supports fine-grained control over the creation and editing of molecules. In this figure, a nitrogen atom is added by first selecting nitrogen from the periodic table, and then clicking on a hydrogen atom that shares a bond with a previously placed carbon atom. By doing that, two new hydrogen atoms will be automatically added to the newly added nitrogen atom. The assembly of a molecule can continue by clicking on further hydrogen atoms. Clicking anywhere else on the 3D environment other than an existing atom will place a new atom in that position, without bonds to other atoms present in the scene. Images taken from `https://documentation.samson-connect.net/users/latest/page_building_molecules.html`.

Tools specialized in DNA origami have also grown in the last years. For instance, caDNAno [DMT+09] enabled DNA modelling using combined 2D views to generate a 3D DNA Origami structure. It heavily constrains the design space to allow for simple and intuitive interactions, limiting the freedom in how nanostructures are assembled, and what kinds of structures can be designed (DNA only). A web version with additional features has also been released: scadnano [DLS20].

Adenita [dLMA+20] is another DNA origami modelling tool, where interaction takes place in a 3D environment. It implements multiscale representations of DNA and was developed as an extension to SAMSON Connect. Although the extension from 2D to 3D designs expands the space of design possibilities, it presents the challenges inherent to the loss of one dimension (when interacting with a 3D scene through a 2D display). Because the tool was developed as an extension to SAMSON Connect, it presents the same problems as the base application.

Another tool, VIVERN [KSB+21] (shown in Figure 3.3), offers similar functionalities in virtual reality as an attempt to solve the challenges of interacting with the 3D environment. One advantage that virtual reality applications have over traditional 3D applications is that the 3D interaction is inherent to the space where the user is immersed, rather than relying on a 2D display and the loss of a dimension. Despite of this advantage, it requires a virtual reality setup (e. g., a virtual reality headset) to function, which is an extra barrier for domain experts, not only in the setting up of the application, but also in actively using it over long periods of time. This is especially a problem when the MolCAD must take place in parallel with other activities, such as consulting papers, external computer applications, or other external resources.

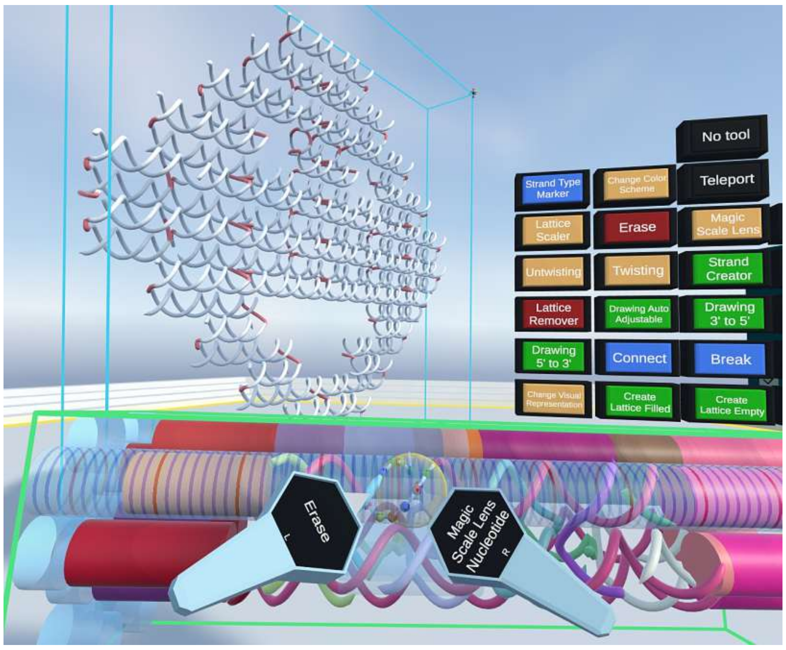


Figure 3.3: VIVERN [KSB+21] is a virtual reality application that enables the design and visual examination of DNA origami nanostructures. Image taken from [KSB+21].

NGL Viewer [RBV+18] is another example of a molecular visualization tool. While it does not offer modelling capabilities, its web-based nature aligns closely with the objectives of this thesis.

This capability made it a viable candidate to power the molecular visualization present in Catana [KMS+22], shown in Figure 3.4. Built on top of the NGL Viewer molecular visualization engine, Catana offers DNA/protein-hybrid modelling capabilities, allowing users to work with all-atom models (such as those found on the PDB, where each atom is described), as well as coarse-grained models (such as those created with caDNAno [DMT+09]). Catana [KMS+22] offers the possibility for both importing and creating structures from scratch, although the latter does not offer that with the detail that SAMSON Connect does.

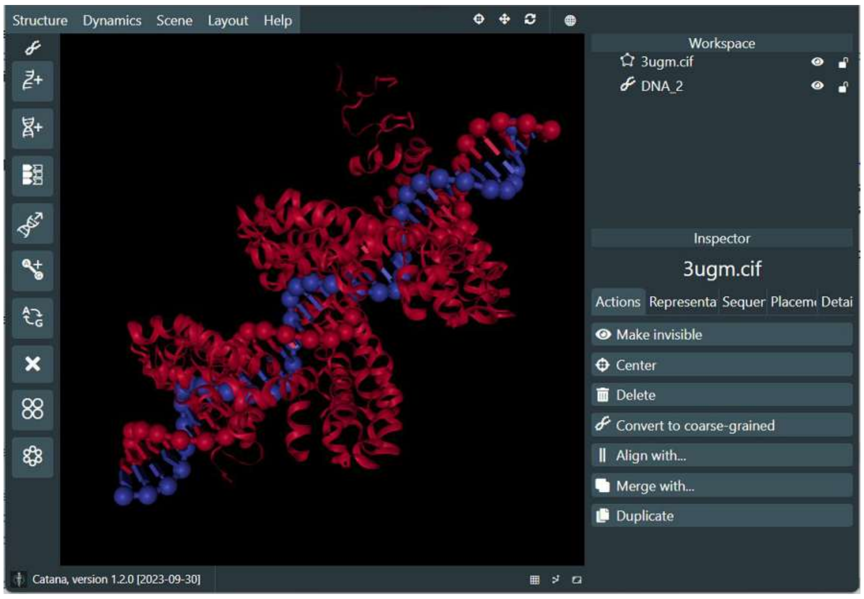


Figure 3.4: Catana [KMS+22] is a web-based MolCAD application, on top of which this thesis' work is implemented.

CHAPTER 4

Design Challenge

The design challenges faced in this thesis arise through RQ1. (*Which CAD interaction techniques exist that are also applicable for MolCAD but are currently not in use yet, but could be useful?*). In order to answer this question, the following plan is devised:

1. Interview domain experts and observe real MolCAD scenarios
2. Study the scenarios, and compile the tasks involved
3. Survey the available literature for existing CAD solutions
4. Identify the design challenges and propose constraint-based MolCAD interaction methods to solve the design challenges

Each step of this plan is described in its respective section below, with an added fifth section where RQ1. is answered. The technical feasibility and implementation of the interactions proposed in this chapter are explored and discussed in detail in Chapter 5.

4.1 Preliminary Interviews

As a first step in defining the design challenges (and ultimately answering RQ1.), user interviews are conducted to identify key real scenarios that domain experts face with MolCAD software. These scenarios will then be used to evaluate the usability and performance of the tool through case studies [SP06] in Chapter 6. Due to our close contact with domain experts and the subjective, human-centred nature of our measure of success of this work, this methodology in the context of the Design Study Methodology [SMM12] is suitable because it encourages constant inclusion of the target users into the design and implementation process in multiple rounds.

The interviews were conducted with 5 domain experts (two biotechnologists and three biophysicists). The format of the interviews is unstructured, without a specific set of questions to steer the interview. Instead, users are encouraged to show their typical MolCAD tasks with their own data while describing their thought process.

One prominent aspect observed during the interviews is that the most widely tools used to design nanostructures are very old. Two prime examples are Pymol [SD] and VMD [HDS96], released in the years 2000 and 1995 respectively. These tools rely heavily on text commands to perform certain tasks such as mutating a protein's amino acid, lack basic 3D modelling tools such as translation/rotation gizmos [CSH$^+$], and have poor performance for modern standards (for instance, for collision detection, an essential aspect of MolCAD discussed further in Section 4.2). The reason that specialists continue to use them is not only because of their familiarity (as these tools are introduced early in university courses), but also because they feel that there is no better alternative.

Two users expressed that a step in the right direction was made with SAMSON Connect [SAM] (discussed in Chapter 3). However, the conducted interviews showed that tools needed for DNA nanostructure design are not present in it, such as the support for the caDNAno [DMT$^+$09] format. The participants mentioned that support for plugins/add-ons helps, such as Adenita [dLMA$^+$20], but added that it crashes often, leading to their progress being lost when unsaved.

Another problem raised by the interviewees was the lengthy and complicated installation process of SAMSON Connect, which includes creating an account. In academic and business environments, such installation processes often require the approval of system managers. That is aggravated by the additional requirement of installing Adenita plugin. The frustration that the users showed by this stresses the importance of a web-based MolCAD application.

Another software solution that arose during the interviews is caDNAno [DMT$^+$09], which simplifies the DNA design process by decomposing the 3D scene into 2D views. However, this alternative only supports DNA in a field where protein-DNA hybrid nanostructures are essential. Additionally, caDNAno heavily constrains the user, and free 3D movement becomes no longer possible, which is a requirement in structural biology (the branch of biology that focuses on understanding molecular structures: their shapes and arrangements, and how they relate to a molecule's function and interactions).

Several tasks performed by the users faced no challenges. For instance, exporting nanostructures after changes have been made, mutating (i. e., changing the type of amino acids in a protein, or mutating a nucleotide in a DNA strand) were all well supported by the applications used.

Some scenarios, however, caused difficulties, frustration, and were found to be excessively time-consuming. Three of these concrete scenarios were collected and are explained in detail below. They are real tasks performed by the users and show the importance of certain tasks (some tasks are present in multiple scenarios) and allow us to get to conclusions about which ones are most essential in MolCAD.

S1. This scenario (shown in Figure 4.1) consists of aligning a newly-created DNA strand with an imported large DNA origami nanostructure.

S2. This scenario (shown in Figure 4.2) consists of loading a protein-DNA hybrid nanostructure from the Protein Data Bank (PDB), removing its DNA part, and replacing it with a newly-create DNA strand with a custom sequence.

S3. This scenario (shown in Figure 4.3) consists of loading two proteins from the PDB and aligning them in such a way that one protein's N-terminus.

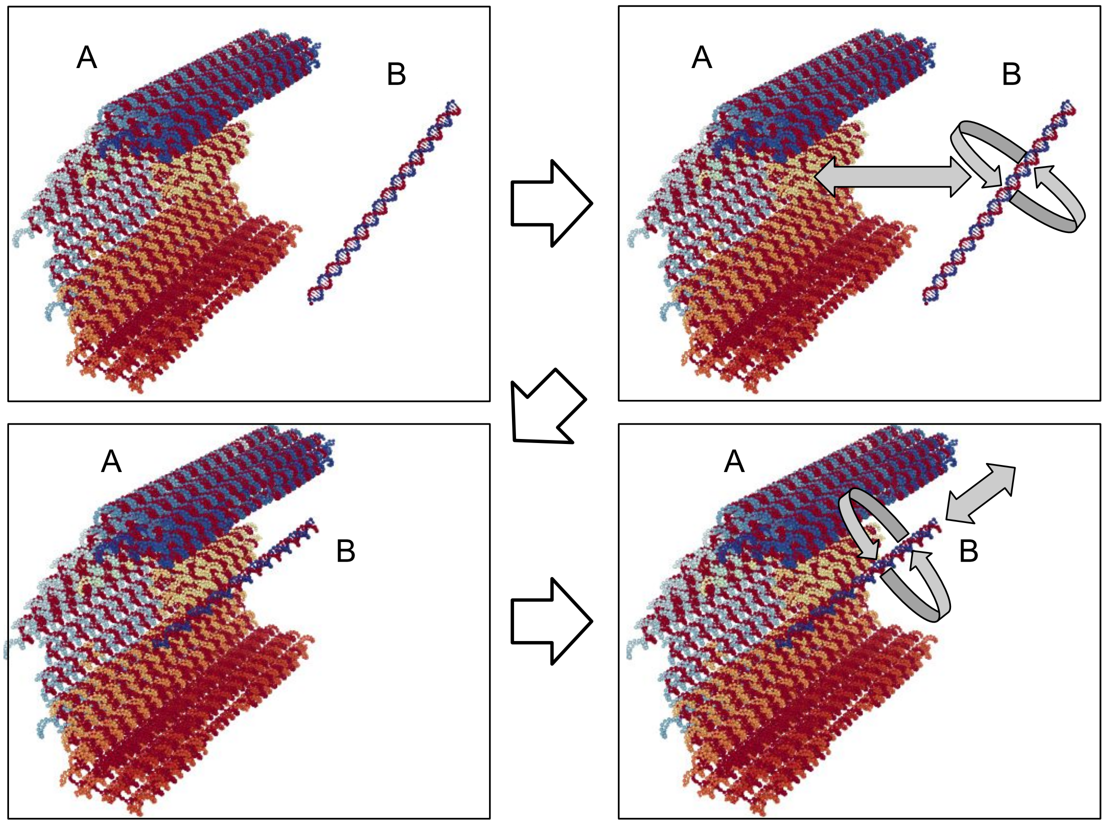


Figure 4.1: The first scenario (S1.) consists of aligning a double DNA strand (B) with a large DNA origami nanostructure (A) (top-left). Through rigid-body transformations alone (translation and rotation, top-right), the DNA strand can be positioned at the desired position (bottom-left). After positioned, small adjustments may be necessary to avoid collisions or ensure that the DNAs bind as intended.

4.2 Core Tasks

The scenarios discussed in Section 4.1 can be narrowed down into tasks. These represent the core tasks observed during the preliminary interviews. From them, the design

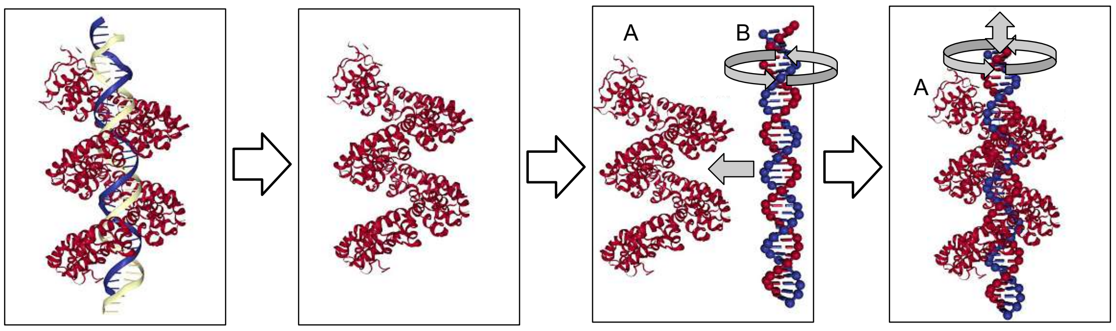


Figure 4.2: Left to right: (1) The second scenario (S2.) starts with a protein-DNA hybrid structure (3UGM) loaded from the PDB (left). (2) The DNA is then removed, and (3) A new one is created with a specific desired sequence and it must be placed where the initial DNA strand used to be. This can be achieved with rigid-body transformations. (4) After alignment, additional precise alignment may need to be performed to avoid collisions and ensure that DNA and protein bind as intended.

challenges are derived. The core tasks are categorised as follows:

CT1. Add: The first step of all three scenarios is being able to add a certain kind of nanostructure, either by importing it (e.g., from the PDB) or by creating one from scratch (create DNA based on a given sequence).

CT2. Remove: The "remove" task is present in Scenario S2., as it involves the removal of a DNA strand.

CT3. Align: All scenarios require the adjustment of a structure with rigid transformations (translation and rotation) in relation to a specific point or axis. Scenarios S1. and S2. require that the DNA strand is not only moved into position (align a strand's helical axis with another's), but also rotated along its helical axis (the axis around which the nucleic acid strands coil). Scenario S3. requires that two structures are positioned at a specific point in relation to each other, as well as that they are able to be rotated around a pivot point. In all scenarios, it is also essential to detect whether atoms collide or whether bonds would be formed (i. e., an atom's covalent radius collides with another's).

Core tasks CT1. and CT2. were both found to already be well supported in MolCAD software. However, the conducted interviews show that current tools implemented in MolCAD software do not account for the challenge that is alignment (CT3.). Users expressed that this task is cumbersome, time-consuming, and frustrating.

As mentioned in Section 4.1, one reason for the magnitude of the alignment challenge is the fact that the user interaction takes place through a 2D display, while the alignment takes place in 3D space [HvDG94, SW10]. Because of that, it becomes challenging to

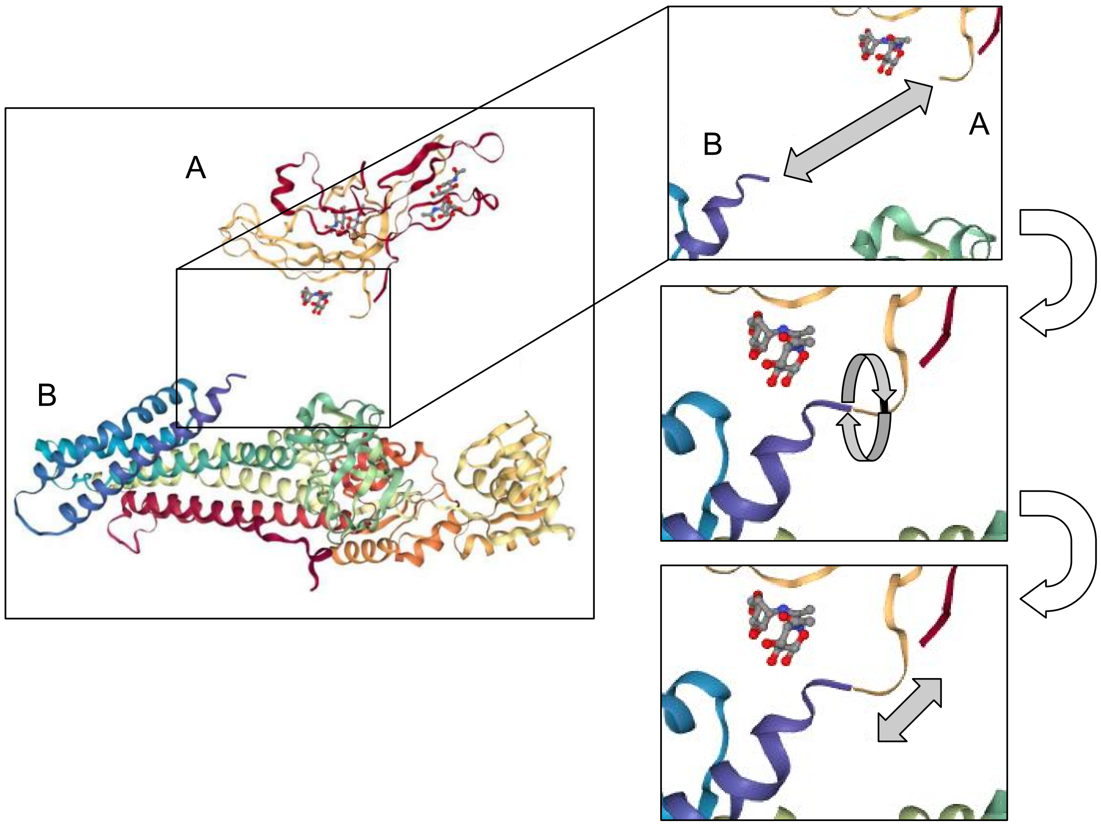


Figure 4.3: (left) Two proteins (`1HRP` and `7R0I`) are loaded from the PDB. The goal is to align one protein's N-terminus with the other protein's C-terminus, so that they form a bond. (right-top) First, the termini are brought together. (right-centre) Then, they are oriented in order to avoid collisions between them. (right-bottom) Once they are positioned and oriented as roughly as desired, small adjustments can be made to make sure the bonds between the termini are formed, or to fix remaining collisions.

translate and rotate objects [MCG+18]. The loss of a dimension makes it difficult to perceive the depth of objects, whether they are closer or farther from the camera. This problem is illustrated in Figure 4.7.

This challenge may be solved by virtual reality applications, such as the one by Kutak et al. [KSB+21], as the user then acquires a more accurate depth perception via the stereo 2D displays. However, domain experts have expressed the difficulty of working with VR applications, which, during the interviews, came mainly from two factors: (1) the cumbersomeness and discomfort of the setup, and (2) the necessity to switch between work environments, e. g., to consult a scientific article or laboratory protocol while working on a molecular design.

Traditional CAD software naturally also faces the same problem. One example of a situation where alignment plays a crucial role in general CAD is shown in Figure 4.4. When creating two adjacent rectangles, the designer must make sure that both rectangles share a common edge. Allowing the user to freely draw a shape is a very powerful feature that gives a paper sketch-like feel to the user, as that too allows for unconstrained drawing. However, the software may give the designer the ability to draw with precision by offering snapping (first introduced as snap-dragging [Bie]) functionality, where interactions that occur close to certain object features are assumed to be taking place on that particular feature, as also shown in Figure 4.4.

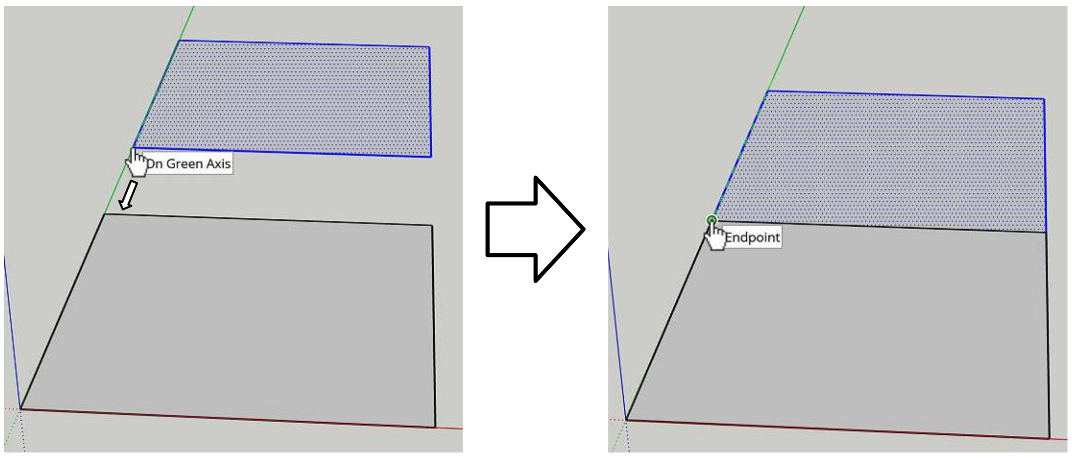


Figure 4.4: SketchUp [Ske] is a CAD application heavily reliant on constraint-based interactions. When dragging the plane with the mouse cursor and following the green line (green axis), the corner of the plane that is being dragged follows the axis. In this process, when the mouse cursor is near enough to the other plane's vertex (right), the plane being dragged snaps in position.

Another way in which constrained interactions are crucial in the example shown in Figure 4.4 is in deciding in which orientation the rectangle is to be drawn. When a rectangle is only defined by its two corner points, there are infinite ways to orient it in 3D space. By using constraints, the software can allow the designer to have intuitive control over the orientation of the objects being interacted with. They achieve that by limiting the range of possible orientations, ensuring that objects align as intended and reducing errors, thus simplifying the task. Figure 4.5 shows one example of this scenario.

Some degree of alignment functionalities are also present in MolCAD software. Several of them (such as PyMol [SD] and NGL Viewer [RBV$^+$18]) offer that in the form of structural superposition. Superposition works by aligning one structure onto another by performing translations and rotations in such a way that they "match". There are several measures for how to define a match (e. g., simple root-mean square deviation, or more complex algorithms that take into account a structure's sequence). Figure 4.6 shows two proteins that have been superposed.

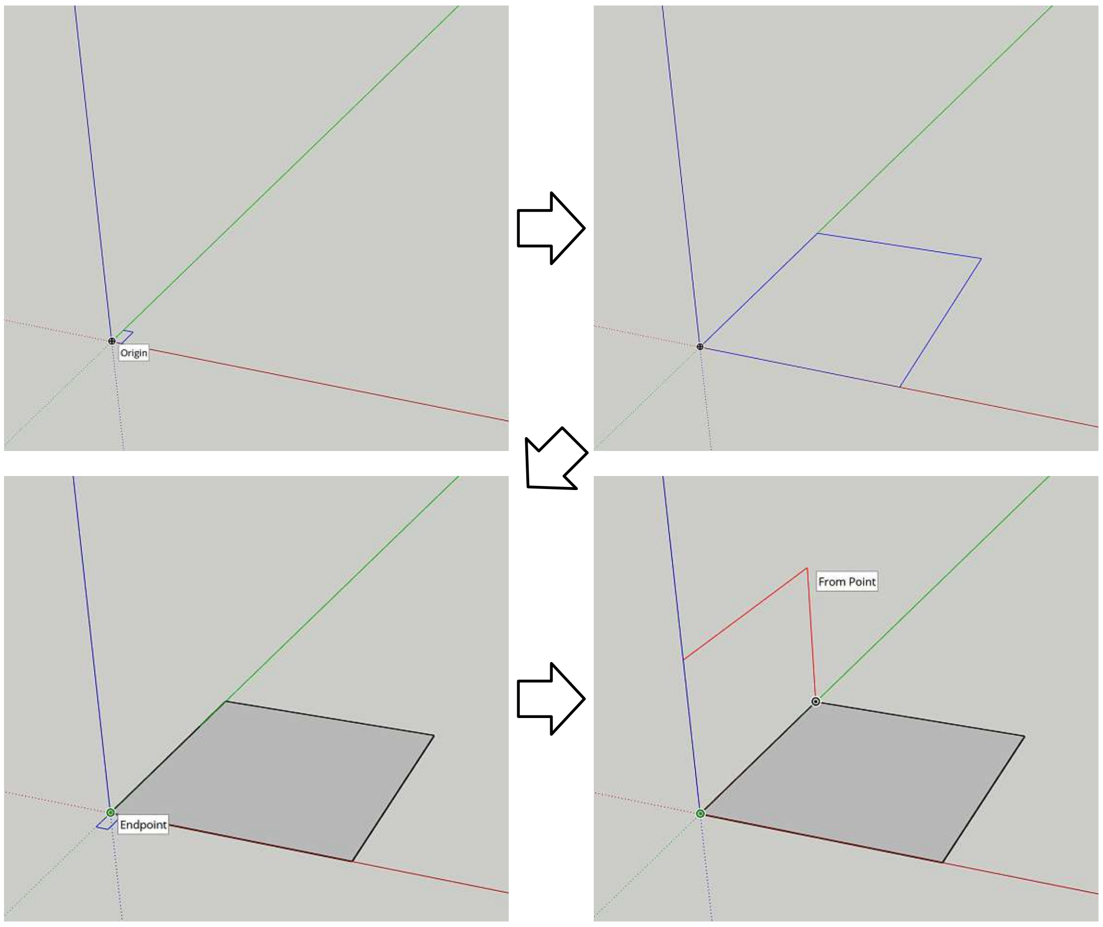


Figure 4.5: SketchUp [Ske] is a CAD application heavily reliant on constraint-based interactions. When creating a plane and placing the mouse cursor near the origin (top-left), the plane-creating tool snaps to it. After a mouse click, a plane starts being created. When there is no other geometry on the scene, the plane will always be created at ground level (top-right). If a new plane is created from already-existing geometry (bottom-left), then it is possible to create it in other orientations (bottom-right).

Although very helpful, this method only covers the cases where two molecular structures are similar in their shape (atomic arrangement) and/or sequence. In many cases, such as those described in Scenarios S1., S2., and S3., the structures that we wish to align are very different.

Together with the alignment interaction, CT3. also shows the importance of collision detection in the studied scenarios. Typically, general CAD software does not have special handling of collisions. All AutoCAD [Auta], SketchUp [Ske], and Blender [Ble] (three of the most well-established CAD software) allow the user to place objects in positions where they intersect/collide. Instead, the responsibility to ensure that objects do not

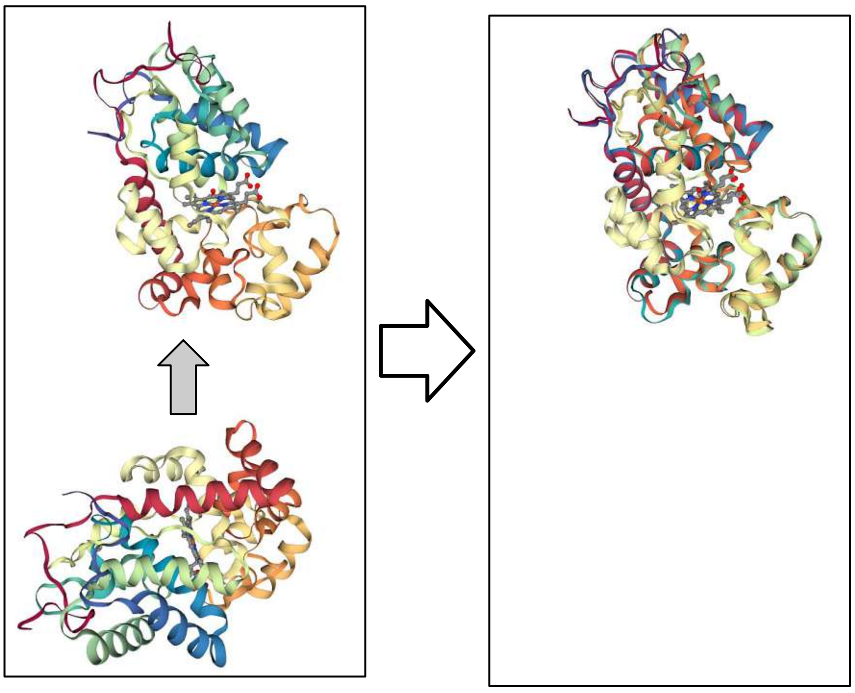


Figure 4.6: Structural superposition allows two nanostructures to be aligned/registered in space while taking into account their structural properties, such as their sequence. (left) In this case, the protein at the bottom is being aligned with the structure on top. They both have very similar amino acid sequences. (right) After alignment, the structures are registered in space, with closely matching positions and orientations. The structures were obtained from the PDB, with codes `1QO4` (bottom) and `1W4Y` (top).

overlap is left entirely to the user.

In MolCAD applications, however, this problem cannot be left for the user alone to solve. This is because molecular structures do not have a defined surface, but instead, their definition of collision depends on what measure is used (e. g., "Covalent" or "van der Waals"). On top of that, tens of thousands to millions of atoms may be present in a structure. This not only causes a very large number of occlusions (making it impossible to visually detect collision), but also it becomes extremely difficult to assess the depth of an atom (it may look like two atoms collide, where one is merely behind another. Lastly, molecular structures do not always display all the atoms, but often rely on alternative representations, such as the "ribbon"/"cartoon" representation for proteins. These issues are illustrated in Figures 4.7 and 4.8 respectively.

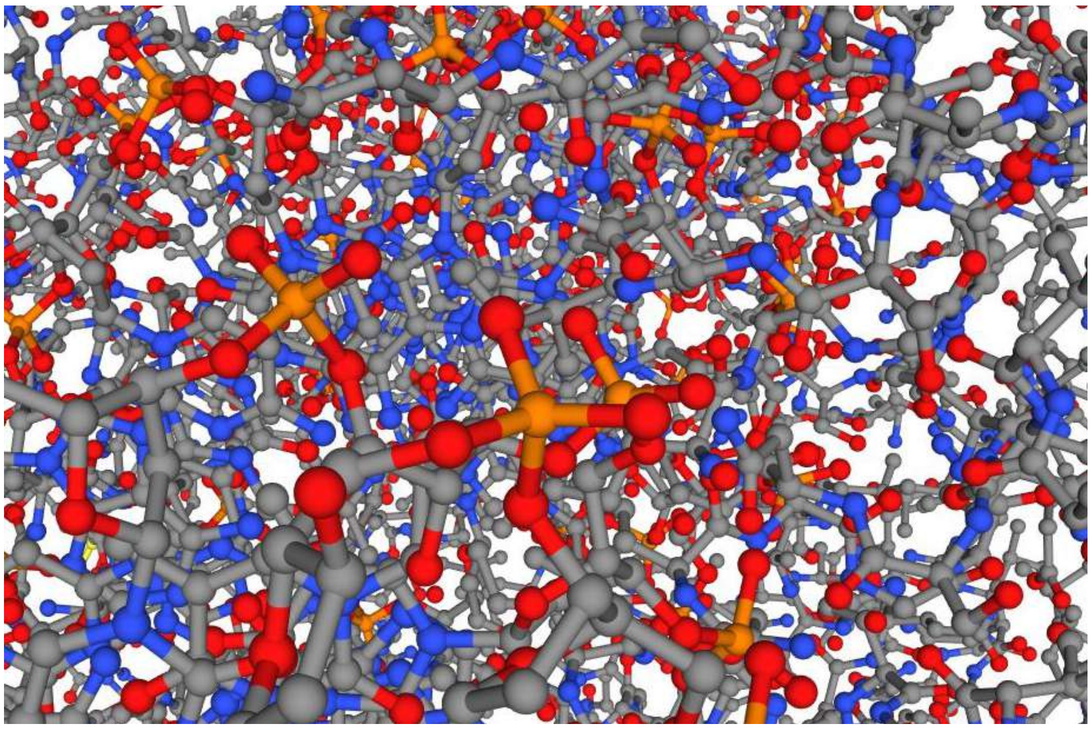


Figure 4.7: Visually determining whether a nanostructure contains atom collisions is unfeasible. Algorithms are needed to support this task. The structure was obtained from the PDB, with code 3UGM.

4.3 Survey on CAD

As a starting point for the short survey provided in this thesis, three extensive surveys are explored [JH12, MCG$^+$18, BYK$^+$21]. These surveys are essential in identifying the challenges of 3D interaction and the solutions proposed. The large body of literature around this topic further supports the emphasis this thesis puts on the need for careful consideration when developing CAD interactions.

General CAD software for architecture, engineering, and other applications benefits from decades of iterative development. AutoCAD [Auta], for instance, has been a staple in the industry since its inception in the 1980s. Its developer, Autodesk, provides a list of their publications [Autb], with a wide range of topics, from general interaction studies, to domain-specific techniques, to virtual reality experiments, since 1988. This shows the extensive amount of work over the decades to develop successful CAD software.

While general CAD software like AutoCAD has undergone decades of refinement, the literature available falls short in exploring and explaining the principles that contribute to its success and widespread use. The study by Sadeghi et al. [SDRP16] is an example of such a case. While it explores the intricacies of structured models and user interaction

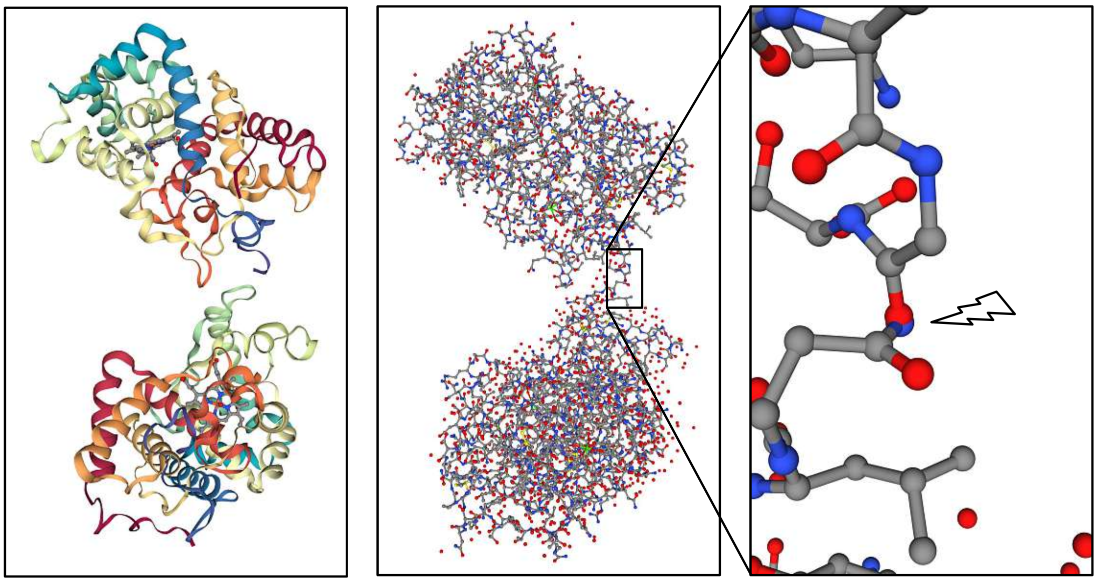


Figure 4.8: With the so-called “cartoon” representation (left), it appears that the two proteins do not collide. However, if a representation that displays the atoms is chosen, such as “ball-and-stick” (centre), it becomes clear that the two structures do collide (right). The structures were obtained from the PDB, with codes `1Q04` (bottom) and `1W4Y` (top).

in the context of a specific CAD software, it does not investigate broad aspects that contribute to making CAD software easy to use. This gap in the literature shows a need for comprehensive studies that aim to understand the user experience in CAD.

Earlier research, however, was more focused on dissecting 3D interaction methods. For example, Hand [Han97] presents a survey with an overview of techniques for object manipulation in 3D virtual environments. The paper also discusses the evolution from 2D mouse-based systems to more advanced 3D input devices, showing the need for interactions that bridge the gap between CAD’s 3D environments and the 2D displays they are mostly presented on.

The challenge in solving the problem of 2D interaction with 3D environments is also made evident by Wang et al. [WPP11]. In their work, they propose a bimanual hand tracking system for CAD. The system uses two consumer-grade webcams to track the user’s hands with 6 degrees of freedom, while explicitly emphasising the importance of exact constraints for precise alignment.

Alignment is also a central topic in the paper by Chaouch et al. [CVB09]. In it, a Principal Component Analysis (PCA)-based auto-alignment technique is discussed. The difficulties of 3D alignment are extensively discussed, further confirming the identification of alignment as a core task in this thesis (CT3.).

Constraint-based interactions have also been the target of studies. Stuerzlinger and Wingrave [SW10] provide an overview of the challenges for developing 3D user interfaces, offering a set of guidelines that promote the use of appropriate constraints for effective interaction in 3D environments. These constraints help users navigate and interact with the 3D environment.

4.4 Constraint-Based Molecular Interaction

Core tasks CT1. and CT2. did not pose a challenge to the users during the interviews. Both of them were found to already be well supported in MolCAD software, including Catana [KMS+22], which is the application that is used to implement the tools explored in this thesis. Core task CT3., however, was found to be especially difficult, and where users showed the most frustration in the interviews.

Observing this gap, alignment (CT3.) is considered to be where the design challenges lie in the context of this thesis. We also observe that the alignment task can be divided into two separate design challenges that are difficult to overcome with currently available tools:

DC1. Alignment

DC2. Collision detection

In order to solve the design challenges and answer RQ1., we first seek inspiration from academic work, as well as established CAD and MolCAD software. The following sections delve into the design challenges (DC1. and DC2.) in more detail and propose solutions that were inspired by the user interviews together with the extensive survey, applied to MolCAD.

4.4.1 Alignment

As discussed in Section 4.2, the constraint-based approach of snapping is a potential candidate for an effective alignment tool. The question remains, however, of whether it may be translated to MolCAD. One of the factors that enable snapping to often work in CAD applications is the existence of inherent geometries, such as lines, vertices, or faces. In MolCAD, these properties are not inherent, as nanostructures are typically not defined by a high-level geometry, but by their atoms.

One potential approach is to calculate a structure's principal axes and use those for alignment [CVB09]. Doing that gives us the axes of largest variance within a structure's atomic structure. For Scenarios S1. and S2., this is a potentially promising method to find a good alignment between the two structures.

The principal axes can be obtained by performing PCA, a technique that is most commonly used to reduce the dimensionality of a set of points. That is because PCA outputs a sequence of unit vectors, in order of best-fitting for the data (defined most often as minimizing the average squared distance from the input points to the output vector/line). In the proposed method, however, PCA is not used to reduce dimensionality, but to acquire the vectors along the axis of most variance.

Based on these two principles, two ideas for constraint-based alignment techniques in MolCAD emerge. They are listed below:

1. The first idea is based on the “snapping” paradigm, where the placement of objects in a scene takes into account other objects in the scene. [Bie] The precise position of the mouse cursor is merely a suggestion of the placement of an object, as the system will use it as a basis to suggest a nearby position that matches other objects (see Figure 4.4 for an example).

2. The second idea is to use the principal components computed from the atoms of a structure and enable not only automatic alignment based on them, but also constrained movement and orientation via the typical translation and rotation gizmos (as shown in Figure 3.1). Though less elaborate than the first alternative, it has the potential to enable a large set of new alignment possibilities (as shown by Chaouch and Verroust-Blondet [CVB09]), especially those presented in Scenarios S1. and S2..

Because of its simplicity and reliability, we decided to pursue Idea 2. Although Idea 1 is based on a very familiar concept present in numerous design software such as SketchUp [Ske], it can become less reliable depending on the complexity of the scene and its structures.

For instance, enabling snapping along the backbone of a protein can lead to a confusing experience for the user, as the backbone can wind seemingly arbitrarily due to the protein folds, as illustrated in Figure 4.9. Alternatively, snapping could be enabled only for simple lines, such as the principal axes of a structure (a concept which overlaps Idea 2). However, performing snapping onto the principal axes would be equivalent to performing an automatic PCA-based alignment, followed by translating in the direction of the principal axes, as illustrated in Figure 4.1.

Because of that, and for the sake of simplicity, we concluded that Idea 2 would be sufficient, and Idea 1 remains an open concept to be explored in future work (as mentioned in Chapter 7). Its implementation of Idea 2 is described in Algorithm 4.1.

The complexity of this algorithm depends on the complexity of the PCA implementation, which can be achieved in $O(n)$ for n atoms. This is discussed in detail in Section 5.1

The algorithm may result in the two structures colliding, since it ends with the position of structure A being assigned structure B's position. This effect is illustrated in Figure 4.11. For certain use cases, such as the one shown in Figure 4.12, this is desirable, as it is expected that structure B has reasonable space to accommodate structure A. In scenarios

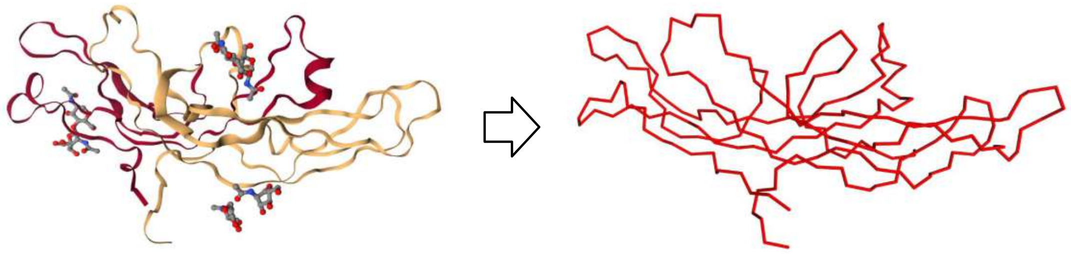


Figure 4.9: A protein (left), and its backbone highlighted (right). Employing snapping at the backbone of a protein could cause confusion, as it winds through space due to its folding. The structure was obtained from the PDB, with code 1HRP.

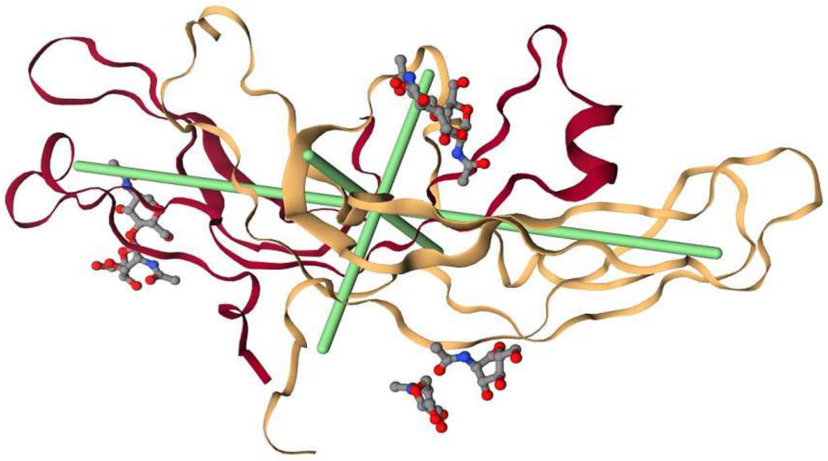


Figure 4.10: The principal axes of a nanostructure could be used for snapping, or as the axis of translation and rotation gizmos (see Figure 3.1) The structure was obtained from the PDB, with code 1HRP.

where this is not desirable, the PCA axes may continue to be used to orient translation and rotation tools, thus facilitating their alignment, as shown in Figure 4.13.

4.4.2 Collision Detection

To address the design challenge of detecting collisions on atomic structures, the trivial solution is the pairwise comparison of each atom, interpreted as a sphere. The radius r_i of an atom i can be defined by several different metrics, such as "van der Waals" or "Covalent". In this trivial method, for every atom i, every other atom j is checked for whether a collision occurs, as shown in Algorithm 4.2. This results in an algorithmic complexity of $O(n^2)$.

Algorithm 4.1: PCA-based alignment of two molecular structures A and B

```
Input: Two molecular structures $A$ and $B$, with $n_A$ and $n_B$ atoms respectively

// $pc$ are the principal components (three orthogonal 3D unit vectors), which can each be pre-computed in $O(n)$
$pc_A \leftarrow \mathrm{PCA}(A)$;
$pc_B \leftarrow \mathrm{PCA}(B)$;

// Rotation: Change basis of $A$ to match basis of $B$
$basis_{pc_A} \leftarrow \mathrm{MatrixFromColumnVectors}(pc_A)$;
$basis_{pc_B} \leftarrow \mathrm{MatrixFromColumnVectors}(pc_B)$;
$A.rotation \leftarrow B.rotation \cdot basis_{pc_B} \cdot inverse(basis_{pc_A})$;

// Translation: Assign $B'$s position to $A$
$A.position \leftarrow B.position$;
```

Algorithm 4.2: Trivial collision detection of n atoms (spheres)

```
Input: A list $atoms$ of $n$ atoms
Output: A list of booleans, each corresponding to an atom, where $true$ means
        that the respective atom collides with another, and $false$ means that
        the respective atom does not collide with any other atom
$collisions \leftarrow$ List of $n$ booleans initialized with $false$;
for $i \leftarrow 0$ to $n$ do
    for $j \leftarrow i + 1$ to $n$ do
        $d \leftarrow \mathrm{EuclideanDistanceBetween}(atoms[i].position, atoms[j].position)$;
        if $d < atoms[i].radius + atoms[j].radius$ then
            $collisions[i] \leftarrow true$;
            $collisions[j] \leftarrow true$;
        end
    end
end
return $collisions$;
```

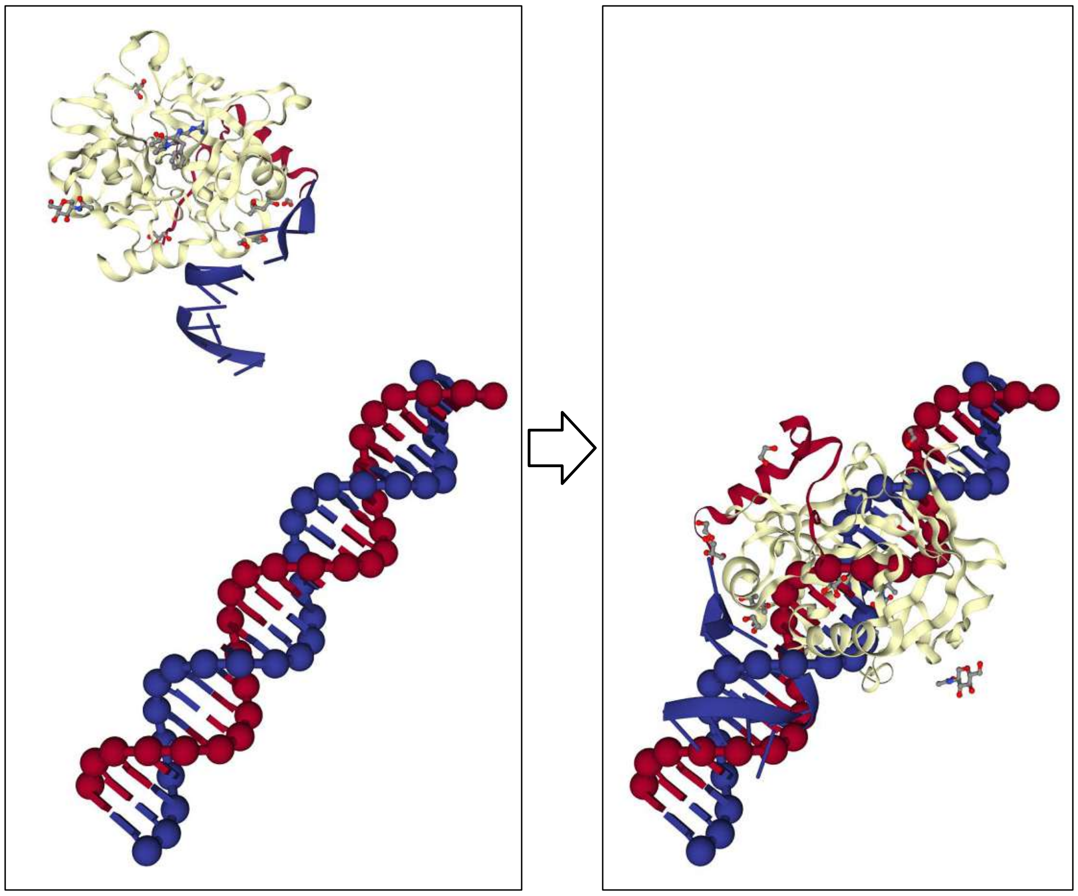


Figure 4.11: When using PCA auto-alignment to align the protein on the top to the DNA strand on the bottom (left), there is a heavy overlap of the two structures. The DNA strand was created using Catana [KMS+22], while the structure was obtained from the PDB, with code `5DO4`.

This trivial approach is already available in several MolCAD applications, such as PyMol [SD] and VMD [HDS96]. Because of the complexity of this approach, and the fact that the algorithm runs sequentially, it may take up to several seconds for the computation to finish. Even with a parallelised version (discussed and described in Section 5.2.2), it becomes unfeasible to perform collision detection in real-time (i. e., at least 20 times per second) for a large number of atoms.

More efficient implementations exist [KAK+18, Pau22] and are discussed in Section 5.2.1. These approaches present a considerable improvement, and prove the possibility of achieving real-time performance for a very large number of atoms (2 million [Pau22]), while also solving the collisions (i. e., when a collision occurs, the colliding objects move away from each other, hopefully closer to a state where no more collisions occur).

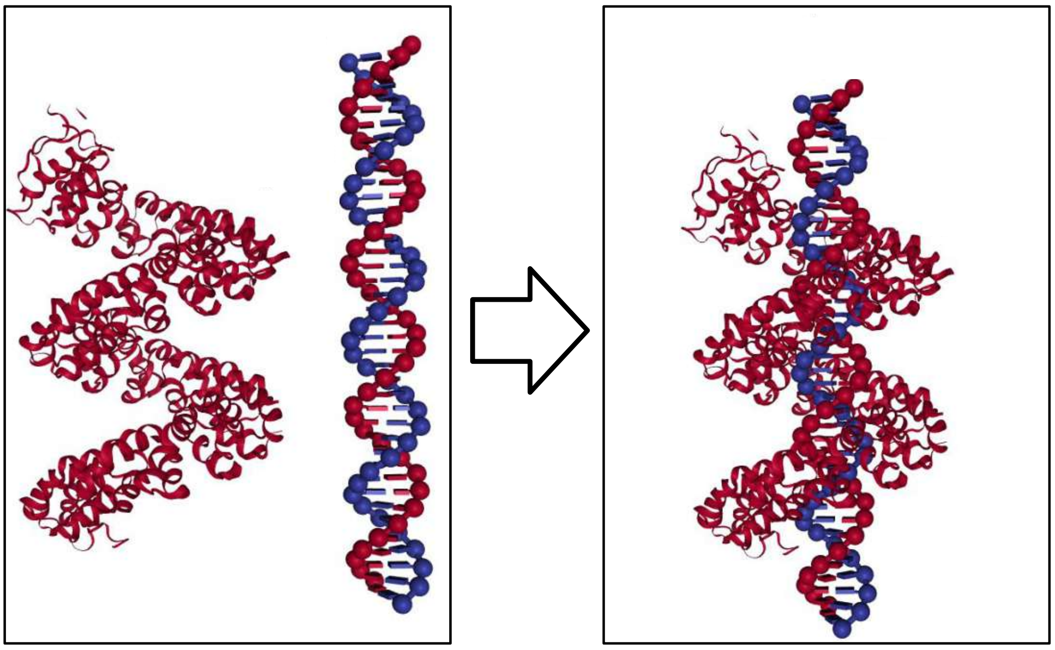


Figure 4.12: When using PCA auto-alignment to align the DNA strand with the protein (left), the structures do not overlap heavily (right), as there is space inside the protein's helix for the DNA to fit. The DNA strand was created using Catana [KMS$^+$22], while the structure was obtained from the PDB, with code `3UGM` and modified with Catana.

Although they are not implemented for the web (but instead for desktop applications), the concepts presented are directly translatable to a modern web environment where compute shaders are available (i. e., WebGPU [Webb]).

The algorithm used is the Fast Fixed-Radius Nearest Neighbor (FFRNN) algorithm and it is described in Algorithm 4.3. It improves on the trivial method (Algorithm 4.2) by checking for collisions only on atoms that are close to each other, rather than with every other atom.

The algorithm starts by subdividing the space into a uniform grid. Each atom is then assigned to a cell in the grid. Next, the atoms are sorted by the cell they belong to, this promoting cache locality (i. e., atoms close to each other in space are more likely to be close to each other in memory). Finally, the collisions can be determined by looking at the neighbourhood of an atom.

The concrete implementation of this algorithm is discussed further in Section 5.2, where a complexity analysis is also provided.

Algorithm 4.3: FFRNN collision detection of n atoms (spheres)

Input: A list *atoms* of n atoms, and the number of cells w, h, d (width, height, depth) for each dimension of the neighborhood grid

Output: A list of booleans, each corresponding to an atom, where *true* means that the respective atom collides with another, and *false* means that the respective atom does not collide with any other atom

```

$collisions \leftarrow$ List of $n$ booleans initialized with $false$;

// Assign atoms to a uniform grid (i.e., neighborhoods)
$atomCounts \leftarrow$ Array of $w \cdot h \cdot d$ unsigned integers, initialized with 0;
$gridCellIds \leftarrow$ Array of $n$ unsigned integers, initialize with $null$;
for $t \leftarrow 0$ to $n-1$ do in parallel
    $cellId \leftarrow$ Index3DTo1D($atoms[t].position$, $w$, $h$, $d$);
    $gridCellIds[t] \leftarrow cellId$;
    AtomicIncrement($atomCounts[cellId]$);
end

// Sort atoms by $cellId$ (for spatially-efficient memory access)
$prefixSum \leftarrow$ ParallelScan($atomCounts$);
$atomsSorted \leftarrow$ ParallelCountingSort($atoms$, $gridCellIds$, $prefixSum$);

// Check for collisions
for $t \leftarrow 0$ to $n-1$ do in parallel
    $atom \leftarrow atomsSorted[t]$;
    for each $neighborCellId$ in $GetNeighborCellIds(atom.position)$ do
        $firstNeighborAtomId \leftarrow prefixSum[neighborCellId]$;
        $lastNeighborAtomId \leftarrow prefixSum[neighborCellId + 1] - 1$;
        $\#neighborAtoms \leftarrow lastNeighborAtomId + 1 - firstNeighborAtomId$;
        for $i \leftarrow 0$ to $\#neighborAtoms - 1$ do
            $neighbor \leftarrow atomsSorted[firstNeighborAtomId + i]$;
            $d \leftarrow EuclideanDistanceBetween(atom.position, neighbor.position)$;
            if $d < atom.radius + neighbor.radius$ then
                $collisions[t] \leftarrow true$;
            end
        end
    end
end

return $collisions$;
```

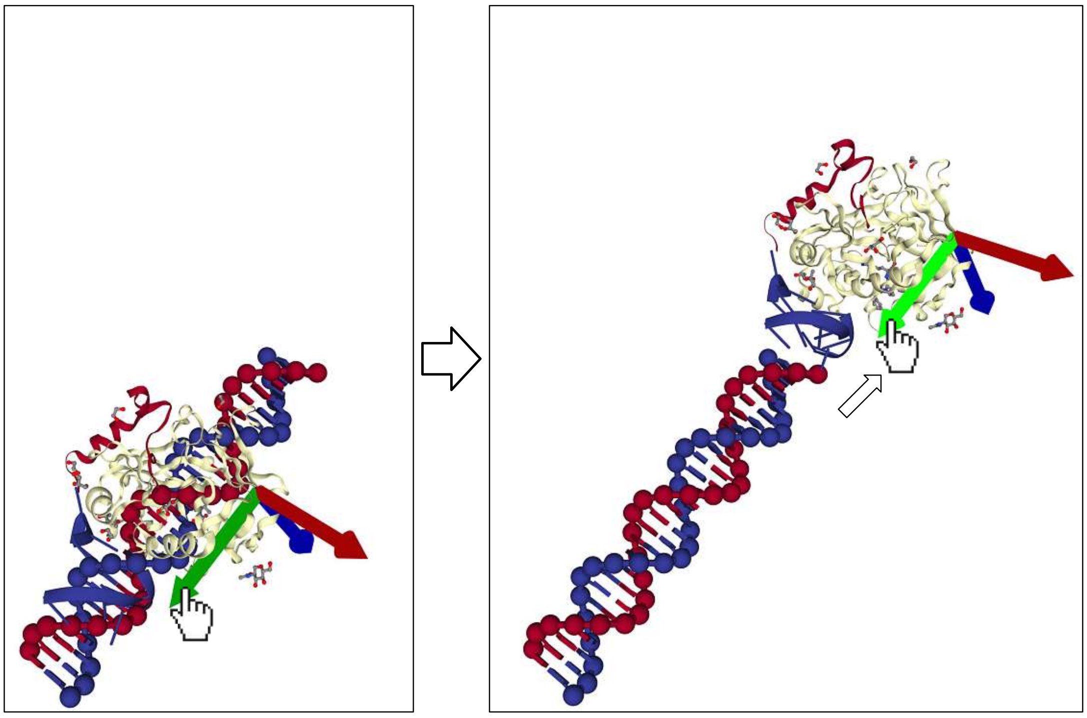


Figure 4.13: Heavy overlaps (as the one caused by PCA auto-alignment in Figure 4.11) can be solved by adjusting the position of the aligned structure. For this task, the principal components can be used as the basis for the translation gizmo. In this case, that allows the protein to be moved along its axis of highest variance, until the small DNA structure attached to the protein gets in close contact with the DNA strand. The DNA strand was created using Catana [KMS+22], while the structure was obtained from the PDB, with code `5DO4`.

4.5 Discussion

After interviewing domain experts, conducting a survey on both CAD software and academic work, and recognizing the main design challenges that emerged from this process, RQ1. can finally be answered: *Which CAD interaction techniques exist that are also applicable for MolCAD but are currently not in use yet, but could be useful?*. While we cannot determine all CAD interaction techniques that can be applied for MolCAD, we found that constraint-based alignment tools are in particular a weak point in established MolCAD software. Based on our interviews and survey, we decided to implement two concrete techniques that show promise in increasing user satisfaction and efficiency: a PCA-based automatic alignment tool and a real-time collision detection algorithm. Their implementation is explained and analysed in Chapter 5 (RQ2.) and their success in improving user experience is validated in Chapter RQ3..

CHAPTER 5

Technical Challenges

In previous chapters, we have analysed the design challenges (see Section 4.4 of efficient and intuitive MolCAD. With those in mind, we then gathered the algorithms that support them (Algorithms 4.1, 4.2 and 4.3). This chapter expands on that by going into detail about the concrete implementation challenges and their solutions. Section 5.4 concludes this chapter and answers RQ2. (*How can the novel MolCAD constraint-based interaction methods be implemented efficiently (in real-time) in a web-based environment?*).

The framework on which the novel MolCAD tools are being implemented is Catana [KMS+22]. Catana is based on NGL Viewer [RBV+18], which implements several molecule visualization features, such as a diverse set of visual representations and colouring options. NGL Viewer [RBV+18], in turn, uses Three.js [thr] internally to achieve that. As of the time of writing of this thesis, Three.js is predominantly based on WebGL (2.0), not yet having fully implemented the novel WebGPU API.

The algorithms required to perform collision detection, however, need general-purpose GPU capabilities (i. e., compute shaders). With rendering done in WebGL, and collision detection performed in WebGPU, one additional challenge arises for the implementation of the required MolCAD tools. That is: managing the interplay of these two APIs, which currently do not have any interoperability features.

5.1 PCA

For the computation of PCA, a sequential algorithm running on the CPU is provided by NGL Viewer [RBV+18] (a molecular visualization framework on which Catana [KMS+22] is based) via the class `PrincipalAxes`. This algorithm is described in Algorithm 5.1.

In the PCA algorithm, first, a 3-component vector *mean* is created, containing the mean $\hat{x}, \hat{y}, \hat{z}$ values of the positions of all n input atoms. The atoms are then centred by subtracting each of their x, y, z components by $\hat{x}, \hat{y}, \hat{z}$ respectively. Then, a 3×3

Algorithm 5.1: PCA of n points

```
Input: A $3 \times n$ matrix $points$ containing, for each of the $n$ rows, the position
       $x_i, y_i, z_i$ of an atom
Output: The three principal axes of $points$, normalized and sorted by largest
        variance first
1 $mean \leftarrow$ meanRows($points$);
2 $pointsCentered \leftarrow$ subRows($points$, $mean$);
3 $covariance \leftarrow pointsCentered' \cdot pointsCentered$;
4 $W, U, V' \leftarrow$ SingularValueDecomposition($covariance$);
5 $axisA \leftarrow U.columns[0]$;
6 $axisB \leftarrow U.columns[1]$;
7 $axisC \leftarrow U.columns[2]$;
8 return $axisA, axisB, axisC$;
```

covariance matrix A of the centred positions is created. The principal axes can then be acquired by decomposing the covariance matrix A into three matrices W, U, and V, such that $A = WUV'$ [Ger]. All the operations mentioned have a complexity of $O(n)$.

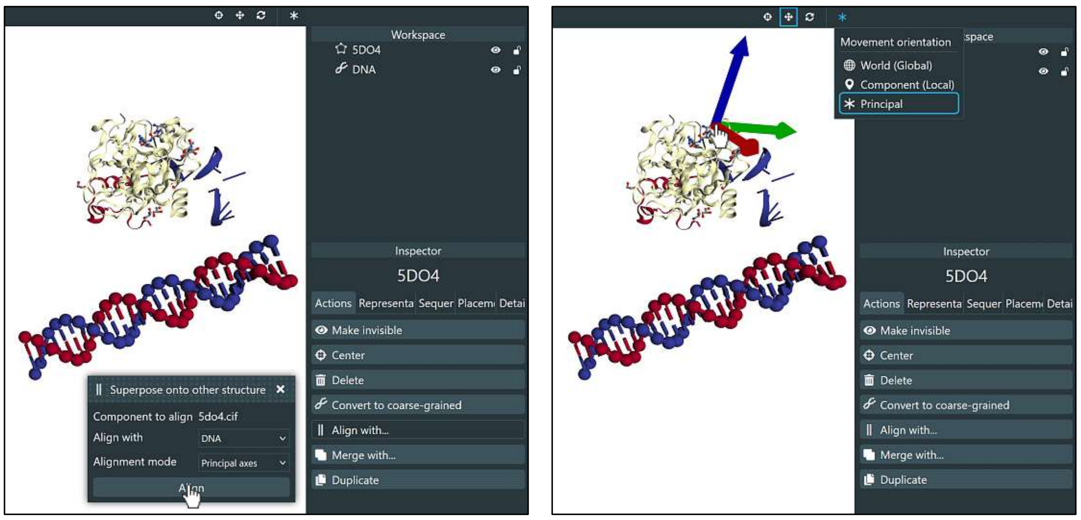
The implementation provided by NGL Viewer is sequential and takes place on the CPU. This is sufficient for our purposes, as this computation is not expected to take place every frame (i. e., there is no real-time constraint). Instead, the principal axes can be pre-computed when a structure is loaded, needing only to be recomputed when the structure changes (i. e., when atoms are added or removed).

Concretely, whenever an atom is added or removed from a structure, the structure's principal axes are marked invalid, signalling that they need to be recomputed. The PCA calculation itself is then triggered if the structure is marked invalid at the time when one of the following user interactions occur: (1) the user clicks the "auto-align" button, or (2) the user selects the translate tool or the rotate tool while in "Principal" mode and then clicks on the structure in the 3D view. These interactions are illustrated in Figure 5.1. After the principal axes are recalculated, they are marked as valid to prevent unnecessary recomputation.

5.2 Collision Detection Computation

Because of their high number of cores (the NVIDIA GeForce RTX 4090 has 16,384 cores [Nvi]) and their hardware architecture (Single Instruction Multiple Data, or SIMD), GPUs are ideal for the task of collision detection. This is because collision detection is a highly parallelisable task, where each atom can, independently from each other, perform a collision check.

That is crucial for the constraint-based interaction proposed in this thesis, as the real-time constraint is essential to guarantee user satisfaction [RR86]. Collision detection does



Component to align 5do4.cif
Align with DNA
Alignment mode Principal axes

Figure 5.1: The PCA implementation used in this work runs sequentially on the CPU. However, the (re)calculation of the principal axes is triggered on demand on two occasions: (left) When the "Principal axes" auto-alignment tool is used; and (right) when the "Principal" movement orientation is selected while using a translation/rotation gizmo. If the structure has not changed (i. e., no atoms were added or removed) since the last computation, the principal axes will not change either, thus their (re)calculation is averted. The DNA strand was created using Catana [KMS+22], while the structure was obtained from the PDB, with code `5DO4`.

not necessarily need to be performed every frame, but rather only when atom positions change. However, because moving atoms and aligning structures is a real-time task in MolCAD, the real-time requirement still applies to the collision detection computations and rendering.

For those reasons, and the requirement that the proposed tools must work on the web, the collision detection computation was implemented using WebGPU [Webb] compute shaders. WebGPU is an API for performing rendering and general-purpose computations on the GPU, in contrast to WebGL [Weba], which only provides rendering functionalities. General-purpose computations are essential for the implementation of a collision detection framework, as they enable direct and efficient access to GPU memory, as well as parallel computing features, such as atomic operations and synchronisation barriers.

These features are supported in WebGPU via compute shaders. A shader is a program written in its own programming language: WGSL in the case of WebGPU. The shader code is compiled and executed on the GPU. Two other types of shaders are also available, both used for rendering: vertex shaders and fragment shaders. Vertex shaders perform computations on vertex data. Three vertices are then used to rasterise a triangle, and each pixel is then processed by a fragment shader. These concepts are shared with all major contemporary graphics APIs (OpenGL [Ope], Vulkan [Vul], Metal [Met]), while WebGL only supports vertex and fragment shaders.

The FFRNN algorithm (discussed in Section 4.4.2 and described in Algorithm 4.3) was implemented. Implementation details are presented in Section 5.2.1. For comparison, the trivial (or "brute force") version of the collision detection algorithm (described in Algorithm 4.2) was modified to be efficiently parallelisable. The modified algorithm is described in Section 5.2.2. The rendering of collisions is then presented in Section 5.3, before finally analysing their performance and comparing it with previous work [Pau22] in Section 5.4.

5.2.1 FFRNN

As presented in previous work [Pau22], FFRNN is an efficient method to perform collision detection between spheres. Atoms are treated as spheres in the proposed framework. The covalent radius is used for computations, as it measures the size of an atom that forms part of one covalent bond. This means that, atoms that "collide" can be interpreted as atoms that share a bond.

Typically in nanostructures, all atoms share a bond with at least one other atom. This can be problematic for a collision detection algorithm that uses the covalent radius, as it would determine that all atoms collide. However, in Catana [KMS+22], the framework upon which the tools presented in this thesis are implemented, bond information is calculated a priori when importing or creating a nanostructure. With this information, the implemented algorithm can detect whether two atoms share a bond, and thus is able to skip atom pairs that share a bond while detecting collisions.

The FFRNN collision detection is performed in a multi-step pipeline of compute shaders. Section 4.4.2 provides a high-level description of the algorithm. Note that the FFRNN pipeline must perform a prefix sum (also known as scan), which may need to be repeated several times, as illustrated in Figure 5.2. This is because of the way that GPU computations are executed, and consequentially how WebGPU's "compute" functionality is defined.

When the execution of a compute shader is invoked, it is necessary to provide the number of workgroups that will be dispatched. A workgroup contains several threads. The number of threads per workgroup is defined by the programmer in the compute shader code. The optimal number of threads per workgroup depends on the specifications of the hardware, but it is general knowledge that a choice of 32 to 256 threads per workgroup leads to good performance.

A prefix sum is an array operation. For each element, the sum of all previous elements is calculated. This can be done sequentially in $O(n)$ for an array with n elements. This operation is not trivially parallelisable, as the result of one element depends on the results of all previous elements, and therefore cannot be performed independently.

With synchronisation functionality available, the prefix sum can be performed in parallel in $O(n)$ [Ngu07]. However, in WebGPU, synchronisation is not available globally across an entire array (or “buffer”), but only within a workgroup. To solve that, multiple prefix sum operations must be performed, and their results added recursively. Therefore, the number of scans and additions performed (and thus also the total prefix sum complexity) depends on both the number of elements and the workgroup size. This strategy is illustrated in Figure 5.2.

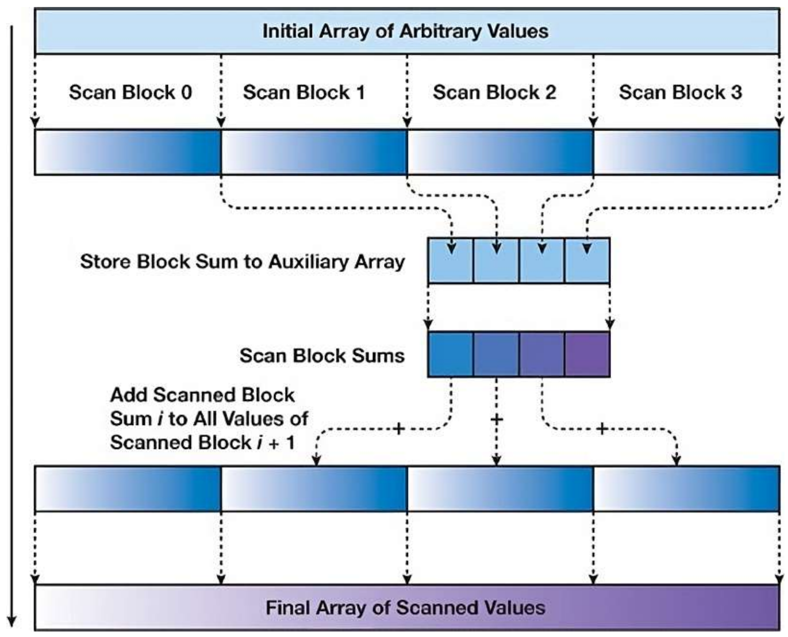

Initial Array of Arbitrary Values
Scan Block 0
Scan Block 1
Scan Block 2
Scan Block 3
Store Block Sum to Auxiliary Array
Scan Block Sums
Add Scanned Block Sum *i* to All Values of Scanned Block *i* + 1
+
+
+
Final Array of Scanned Values

Figure 5.2: The parallel scan algorithm implemented requires a barrier for all threads involved. Because WebGPU only offers barriers on the workgroup level (which, in our case, has a size of 256), the scan can only be performed on blocks of 256 elements. The solution is to take the largest element of a scan block (which is equivalent to the sum of all previous elements in the original data) and perform a scan on them. This process is repeated recursively until the resulting data has fewer elements than the workgroup size (256 in our case), and thus only one workgroup is needed. Once this point is reached, the result is added to the scan of the previous recursion, and so on. This recursion thus has a depth of $\lfloor log_w m \rfloor$ on the number m of grid cells and the workgroup size w. Image taken from `https://developer.nvidia.com/gpugems/gpugems3/part-vi-gpu-computing/chapter-39-parallel-prefix-sum-scan-cuda` (accessed 2023-09-04).

The list below describes all the steps of the WebGPU FFRNN pipeline. The algorithmic complexity is also analysed for n atoms, m grid cells, p processors/cores, and a workgroup size of w threads. Because the algorithm's performance also depends on the number of atoms contained in a grid cell and on a $3 \times 3 \times 3$ neighbourhood of cells, the number of atoms in a cell i is denoted by k_i. The largest number of atoms in any cell is thus denoted by $max(k)$. Note that, to ensure that a $3 \times 3 \times 3$ neighbourhood is sufficient, one cell must be at least the size of an atom, otherwise a larger neighbourhood would be necessary.

1. **Grid cell assignment** $O(m/p + n/p\ max(k))$
    a) **Fill-0:** For every grid cell, set its atom count to zero. $O(m/p)$
    b) **Insert:** For every atom, find which cell the atom is contained in. Assign the atom to the cell: the atom keeps a reference to the grid cell, and the cell atom count is incremented atomically. An atomic operation is generally considered sequential when performed on the same variable. $O(n/p\ max(k))$
2. **Sort atoms by the cell they are contained in** $O(m/p\ low_w m + n/p\ max(k))$
    a) **Scan:** Perform a prefix sum of the atom count of each grid cell. $O(m/p\ log_w m)$
    b) **Fill-`null`:** For every atom, prepare a separate atoms array/buffer that will contain the sorted atoms. Counting sort [Ngu07] will be performed next, which needs the output array to be initialized with `null`-type values. They are implemented in this work as the maximum value of a 32-bit unsigned integer (i. e., `0xffffffff`). $O(n/p)$
    c) **Sort:** For every atom, sort it by its grid cell using counting sort, thus assigning it to a position in an output array. $O(n/p\ max(k))$
    d) **Map:** The previous step outputs an array of sorted atom indices. In this step, the data of the atoms (e. g., the positions of atoms, or an atom's bonds) is arranged to follow the same sorted order. This is done for each atom. $O(n/p)$
3. **Detect collisions:** For each atom, find all atoms in its $3\times3\times3$ grid neighbourhood and check for collisions. $O(n/p\ max(k))$

The WebGPU implementation of the FFRNN algorithm thus has a complexity of

$$O\left(\frac{m\ log_w m + n\ max(k)}{p}\right).$$

When following the recommended workgroup size in the range from 32 to 256, w has a negligible influence on the complexity. What truly influences the efficiency of the algorithm is the number of grid cells m, the number of atoms n, and the density of the atoms $max(k)$ (i. e., the largest number of atoms in any grid cell).

In conclusion, the efficiency of the algorithm is highly dependent on the size of the grid cells. A cell size too small results in a very large number of grid cells m. On the other hand, a cell size too big will result in a large number of atoms being assigned the same grid cell, increasing the value of $max(k)$. Pauscheinwein [Pau22] found that a cell size matching the size of a single atom results in the best performance. Therefore, this cell size is used for performance benchmarks in Section 5.4.

5.2.2 Brute Force

For comparison purposes, the trivial (brute force) version of the collision detection algorithm (Algorithm 4.2) was implemented in WebGPU as well. As mentioned in Section 4.4.2, this algorithm can be parallelised. This can be trivially done by parallelising the outer for loop and having each thread perform the inner loop. However, the inner loop's variable j depends on the outer loop's variable i. Two ways to implement this are considered in this section.

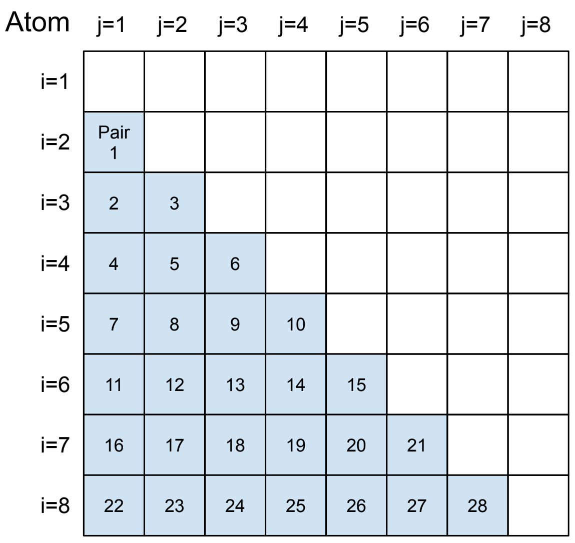
In the first alternative where the outer loop is parallelised, each thread performs a different amount of work (thread 0 performs $n-1$ checks, while thread $n-1$ performs no checks). This would lead to poor performance on GPU hardware, as GPUs follow a Single Instruction Multiple Data (SIMD) execution pattern. This means that computing cores that perform less work would still have to wait until cores that perform more work are finished in order to perform further computations.

The second alternative is to dispatch n^2 threads (one for each pair of atoms). This has the disadvantage that the algorithm must check whether two atoms being compared are the same, to avoid self-collisions. Despite performing more collision checks than the first alternative (n^2 rather than $\frac{n^2-n}{2}$), it would likely lead to faster execution on GPU hardware due to the SIMD model.

The ideal solution to parallelise the trivial collision detection algorithm would involve two properties: (1) an atom never checks for a collision with itself; and (2) one atom pair is only checked for collisions once (as their collision is commutative, i. e., determining the collision of atom a with atom b yields the same result as determining the collision of atom b with atom a).

Keeping these properties while parallelising the trivial collision detection algorithm leads to a pairwise comparison of atoms that can be expressed as the lower or upper triangle of a matrix, as illustrated in Figure 5.3. In such a matrix, each element represents one pair of atoms. An element in the diagonal represents the pair of an atom with itself. The lower triangle contains exactly the same pairs as the upper triangle (symmetry property). Checking collisions of only the atom pairs contained in either the lower or upper triangle of this matrix would fulfil both properties discussed.

In order to achieve that in parallel, we take advantage of the triangular numbers as follows: First, we number each element of the lower triangle of the n^2 matrix with incrementing values, as illustrated in Figure 5.3. Then, using the triangular numbers, we formulate the mapping of this number to the atom's position in the matrix as follows: Let a matrix element's



Atom	j=1	j=2	j=3	j=4	j=5	j=6	j=7	j=8
i=1								
i=2	Pair 1							
i=3	2	3						
i=4	4	5	6					
i=5	7	8	9	10				
i=6	11	12	13	14	15			
i=7	16	17	18	19	20	21		
i=8	22	23	24	25	26	27	28	

Figure 5.3: The unique computation and indexing of atom pairs can be visualised as a symmetric matrix. Each element of the matrix represents a pair of atoms, with the elements in the diagonal representing pairs of an atom with itself. For collisions, only unique pairs of atoms are necessary to compute, and self-collisions must be avoided. Such pairs can be described as the lower/upper triangle matrix, which has $\frac{n^2-n}{2}$ elements for a number n of atoms.

number be denoted by t, and the number of atoms be denoted by n. The matrix contains n^2 elements, and its lower and upper triangles contain $N = \frac{n^2-n}{2}$ elements. An element's position/coordinates i, j in the matrix is then defined by the following two equations:

$$i \leftarrow n - 2 - \left\lfloor \frac{\sqrt{-8t + 4n * (n-1) - 7}}{2} \right\rfloor - 0.5, \text{ and}$$

$$j \leftarrow t + i + 1 - N + \frac{(n-i)(n-i-1)}{2}$$

With that, a parallel version of the trivial collision detection algorithm that still performs $\frac{n^2-n}{2}$ checks is enabled. This version of the algorithm is shown in Algorithm 5.2. Its complexity for a p number of processors/cores is $O(n^2/p)$.

5.3 Collision Rendering

In order to perform rendering in both WebGL and WebGPU using HTML5 and Javascript, a `canvas` element must be created. Once created, this canvas must be given a context. For these two APIs, their respective contexts are `"webgl"`/`"webgl2"` and `"webgpu"`. This means that it is impossible to give one canvas two different contexts, and therefore

Algorithm 5.2: Parallel trivial collision detection of n atoms (spheres)

```
Input: A list atoms of n atoms
Output: A list of booleans, each corresponding to an atom, where true means
        that the respective atom collides with another, and false means that
        the respective atom does not collide with any other atom
1  $collisions \leftarrow$ List of $n$ booleans initialized with $false$;
2  $N \leftarrow \frac{(n^2-n)}{2}$;
3  for $t \leftarrow 0$ to $N-1$ do in parallel
4      // From thread ID $t$, compute two atom indices $i$ and $j$
5      $i \leftarrow n - 2 - \left\lfloor \frac{\sqrt{-8t+4n*(n-1)-7}}{2} \right\rfloor - 0.5$;
6      $j \leftarrow t + i + 1 - N + \frac{(n-i)(n-i-1)}{2}$;
7      $d \leftarrow \text{EuclideanDistanceBetween}(atoms[i].position, atoms[j].position)$;
8      if $d < atoms[i].radius + atoms[j].radius$ then
9          $collisions[i] \leftarrow true$;
10         $collisions[j] \leftarrow true$;
11     end
12 end
13 return $collisions$;
```

that the collision data calculated with WebGPU can only be rendered using WebGPU via a separate `canvas` element.

Therefore, a `canvas` element with a WebGPU context is created and overlaid onto the already existing WebGL canvas (set up as a part of Catana [KMS+22]). Each collision is then rendered as a two-dimensional marker, as shown in Figure 5.4. That results in an algorithmic complexity of $O(n/p)$ for n atoms and p processors/cores.

Although this method of displaying collisions results in real-time benchmarks (later shown in Section 5.4), its drawback is that it does not share the scene depth with the WebGL context. Because of that, the markers of all collisions are always displayed, even if the collision is occurring far away from the camera, where it would otherwise be occluded by other atoms which are closer to the camera. This effect is desirable when an overview of all collisions is required. However, when trying to determine the precise location of collisions, it becomes ineffective.

In order for both APIs to share the depth of the scene, sharing data between them is necessary. However, despite both technologies being developed for the web, their GPU assets (such as textures and buffers) are not compatible. A texture defined in WebGL, for instance, cannot be directly used in WebGPU. To pass any data between the APIs, it is necessary to first copy the data from one API to the CPU, then copy it from the CPU to the other API.

This approach was implemented by first copying the collision data calculated in WebGPU

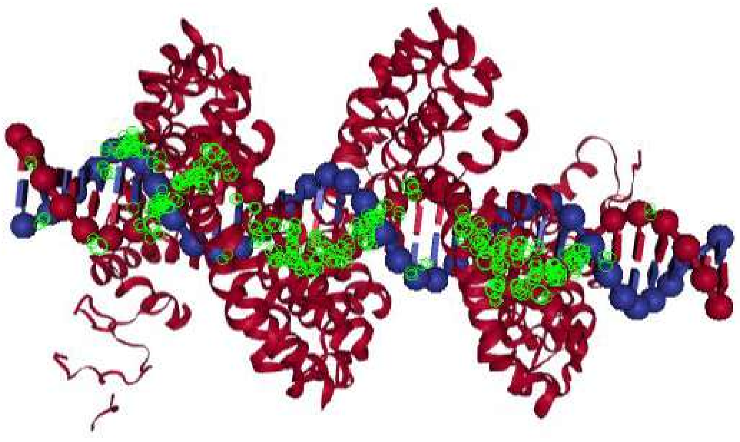


Figure 5.4: WebGPU collision markers, overlaid on the WebGL render of the molecular scene. The markers do not share the depth with the rest of the scene. Therefore, they are not occluded.

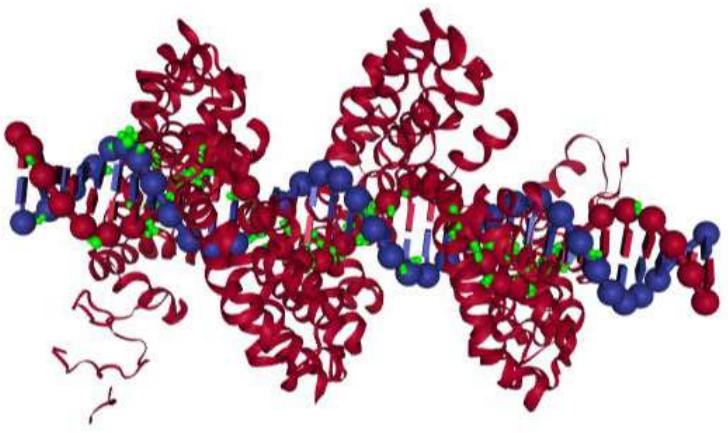


Figure 5.5: WebGL collision markers. Unlike their WebGPU variant, they are in the same molecular scene as the nanostructures, and are therefore subject to occlusions.

with FFRNN to the CPU. After that, the collision data is used to populate a WebGL vertex buffer in such a way that each collision results in a sphere. This buffer is then added to the same scene where all the nanostructures are rendered. By doing that, both the nanostructures and the collision spheres share the same scene depth, and can thus be occluded by each other. The result is shown in Figure 5.5.

5.4 Performance

With the two methods chosen to improve the MolCAD process (PCA for alignment and FFRNN for collision detection – answering RQ1.), and verified to be possible to achieve in a web-based environment, this section aims to verify that their implementation achieves sufficient performance (RQ2.). For the alignment challenge (DC1.), because there is no real-time requirement (as discussed in Section 5.1), a sufficiently fast performance is considered to not be more than 1 second, to avoid interrupting the user's flow of thought [Nie93]. For collision detection (DC2.), however, real-time performance is necessary (as discussed in Section 5.2).

There is no hardline definition for how many frames per second are required for an application to be considered real-time. At 10 frames per second, a system can be considered to be reacting instantaneously [Nie93]. For the purposes of this thesis, a framerate of at least 20 frames per second (or 50 milliseconds per frame) will be used as a threshold, as that is generally considered to be real-time in visualization applications.

The experiments were run on a laptop with an Intel(R) Core(TM) i7-11800H processor with 8 cores (16 logical processors) running at 2.30GHz, using 16GB of RAM, running Windows 11, with an NVIDIA GeForce RTX 3070 Laptop GPU. The Catana [KMS+22] source code, where the algorithms were implemented, is directly modified to include the time measurements. CPU times were measured with Javascript's `performance.now()` function, which returns a timestamp in milliseconds with up to microsecond precision. GPU times were measured with WebGPU's `TimestampQueries`, which provide timestamp values in nanoseconds, which are then converted to milliseconds for easier interpretation and comparison.

The data used for the experiments consists of two proteins downloaded from the PDB. Their codes are `1BNA`, `5ES5`, and `7A5J` and they are shown in Figure 5.6.

These proteins were chosen based on their atom counts: 566, 10,783, and 106,132 respectively. The structure with 566 atoms was chosen due to its low number of atoms, to analyse how the algorithms perform with small structures The remaining two values are convenient because of their proximity to 10,000 and 100,000 respectively. By duplicating these proteins in the scene, larger values are achieved. For ∼50,000 atom and ∼100,000 atom benchmards, 5 and 10 `5ES5` were used respectively. For ∼200,000 atoms up to ∼4,000,000 atoms, the `7A5J` protein was used (2 up to 40 of it).

These numbers go beyond the number of atoms that domain experts work with routinely. DNA origami nanostructures can reach the hundreds of thousands of atoms (as the one shown in Figure 2.2), and are considered unusually large. In the preliminary interviews conducted (see Section 4.1), users reported facing very slow framerates with structures of this size while using Adenita [dLMA+20], a modern tool tailored for DNA origami nanostructures. With older software, such as PyMol [SD], the participants reported that they were unable to load structures of this scale. Moreover, the PDB format only supports up to 99,999 atoms per file [PDB]. For those reasons, the number of atoms

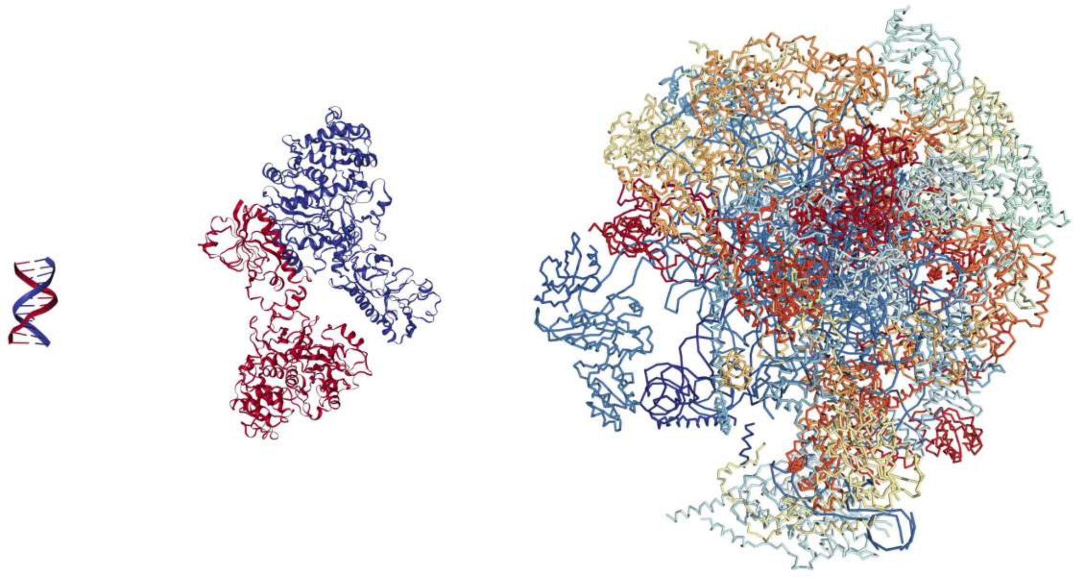


Figure 5.6: The PDB structures `1BNA` (left), `5ES5` (center), and `7A5J` used in the benchmarks. They contain 566, 10,783, and 106,132 atoms respectively.

chosen for the benchmarks are regarded as not only well within practical purposes, but beyond them.

5.4.1 PCA

For the PCA experiments, the loaded proteins were placed arbitrarily in space, as their position does not influence the efficiency of the algorithm. The results are presented in Table 5.1 and visualised in Figure 5.7. The values represent the average run time over 20 runs, after 5 warmup runs (25 runs per column in total). By increasing the number of atoms, the processing time increases linearly. This is in accordance with the algorithm's complexity of $O(n)$ mentioned in Section 4.4.1 and analysed in Section 5.1. The results of this CPU implementation are satisfactory, as they remain far from the defined maximum delay of 1 second, even for very large structures with over two million atoms. Because of this, a GPU implementation was not considered.

# Atoms	107,830	212,264	424,528	1,061,320	2,122,640
Time (ms)	3.61	6.40	12.42	29.90	61.12

Table 5.1: Computation times (in milliseconds) of the PCA algorithm. Averaged over 20 runs after 5 warmup runs.

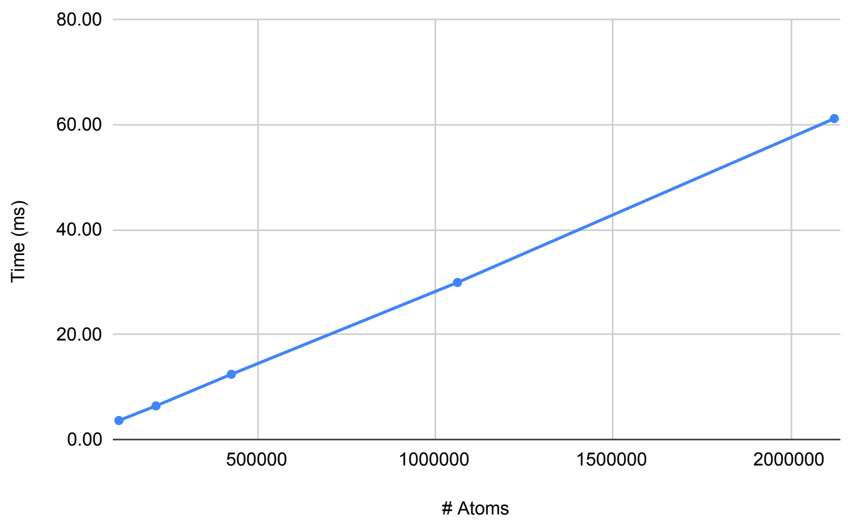


Figure 5.7: Results of the performance measurements of PCA algorithm, plotted according to Table 5.1.

5.4.2 Collision Detection

Both the FFRNN algorithm and the parallel trivial collision detection algorithm were implemented on the GPU and, for comparison purposes, sequentially on the CPU (Algorithms 5.2 and 4.3 respectively). Because they are both integrated in Catana [KMS+22], the CPU versions are implemented in Javascript, while the GPU ones are implemented using the WebGPU API [Webb]. On the CPU, the prefix sum algorithm does not use the scheme presented in Section 5.2.1, as it is implemented sequentially, making its implementation trivial (see Figure 5.2).

All times measured use a grid with fixed dimensions (i. e., number of cells), and each cell with a fixed size. This was done in order to facilitate comparison with previous work [Pau22], which only provides experiment results based on the number of atoms.

The size of the grid would otherwise be determined dynamically, and chosen in such a way that it contains all atoms in the scene. Each cell in the grid is initially defined as a cube where each edge has a length equivalent to the diameter of the largest atom in the scene [Pau22]. If the number of grid cells exceeds the maximum imposed by WebGPU limitations, the dimension of the grid's cells must be increased such that it spans all atoms in the scene.

WebGPU imposes a maximum number of 65,535 workgroups that can be dispatched in one single call. In turn, each workgroup has a maximum size of 256. Because of that, the maximum number of grid cells that can be processed in one call is $16,776,960 = 65,535 \times 256$. Therefore, the largest cubic grid possible while computing collisions with one single dispatch call is one where each side has $255 = \lfloor \sqrt[3]{16,776,960} \rfloor$ cells. With that, the fixed-size grid contains a total of $255 \times 255 \times 255 = 16,581,375$ cells.

A comparison of all implementations is provided in Table 5.2 and visualised in Figure 5.8. The values represent the average run time over 20 runs, after 5 warmup runs (25 runs per column in total). The CPU brute force variant was only executed for the smaller test cases, as it was quickly found to be excessively slower than the other algorithms. Measurements for GPU brute force could only be computed for up to the 1 061 320-atom test case, as larger test cases would require a number of threads to be dispatched (2,252,799,223,480 threads for 2,122,640 atoms) that is too large for the test hardware, leading the application to crash.

	CPU		WebGPU		
# Atoms	**BF**	**FFRNN**	**BF**	**FFRNN**	**Rendering**
566	1.65	122.59	0.02	2.58	0.004
53,915	11,439.56	127.23	10.24	2.56	0.042
107,030	47,655.94	219.74	30.98	2.66	0.078
212,264		458.53	99.95	2.78	0.155
424,528		807.01	322.37	2.91	0.305
1,061,320		1,660.64	2,024.98	4.85	0.754
2,122,640		2,902.69		9.75	1.505
3,183,960		3,863.94		12.55	2.884
4,245,280		4,639.43		18.91	3.198

Table 5.2: Computation times (in milliseconds) of the Brute Force (BF) algorithm, and the FFRNN algorithm. Averaged over 20 runs after 5 warmup runs.

Looking at the experiment results for 566 atoms, the overhead introduced by the fixed-size grid of FFRNN becomes clear, as their times are much larger than brute force. However, as the number of atoms grows, the difference in how they scale also becomes clear, with brute force values quickly reaching times above the real-time threshold (i. e., 50 milliseconds), while the FFRNN GPU implementation remains real-time for all test cases, even when adding the rendering times. The CPU version of FFRNN, although not delivering real-time results even for the smallest test case, shows the importance of an efficient collision detection algorithm when compared to CPU brute force, even if implemented sequentially.

The experiment results for each individual GPU FFRNN pipeline stage are also provided in Figure 5.9. Table 5.3 contains the data used in the plots.

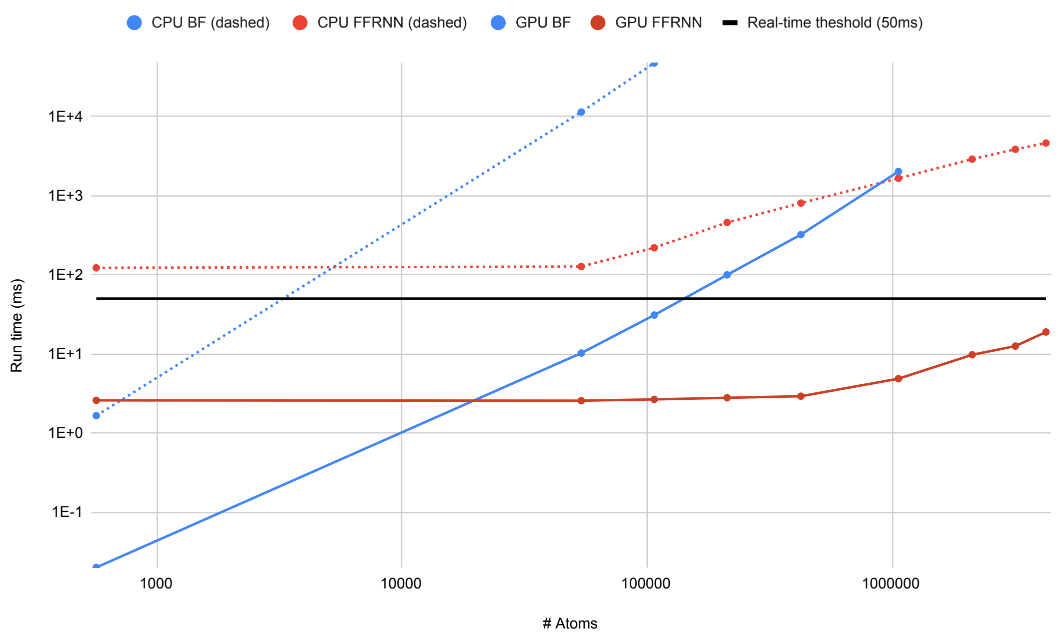


Figure 5.8: Results of the performance measurements of the Brute Force (BF) algorithm, and the FFRNN algorithm, plotted according to Table 5.2. Dashed lines are the CPU implementations, while solid lines are the GPU implementations. Both axes are logarithmic. Even for a very low number of atoms, the CPU implementations are not able to calculate collisions in real-time. The GPU BF variant loses that capability at a little over one million atoms. The GPU FFRNN variant, however, is able to remain below the real-time threshold even for a very large number of atoms.

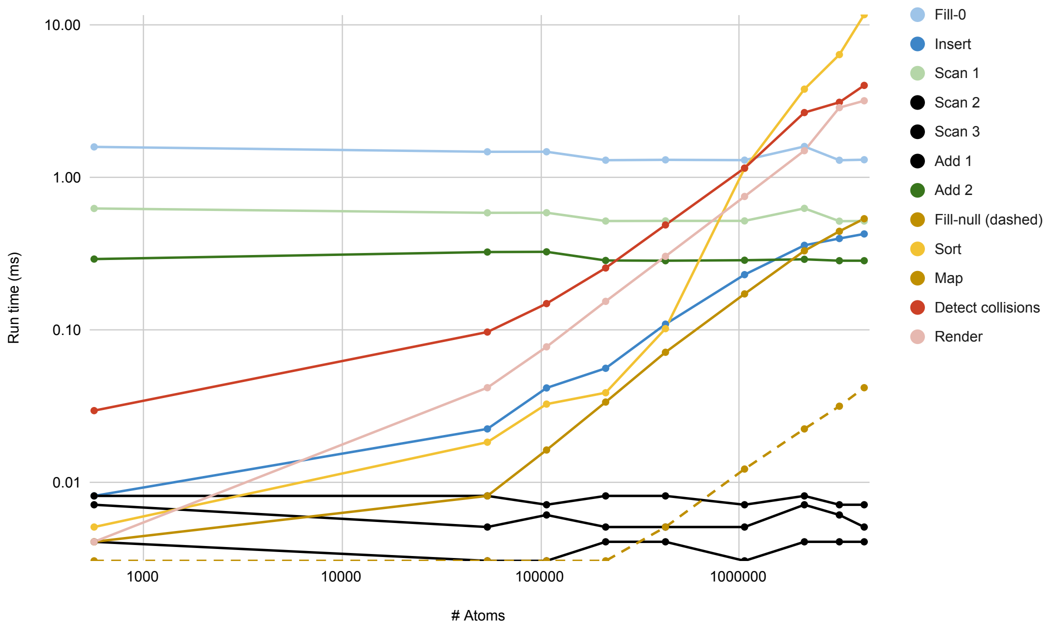


Figure 5.9: Results of the performance of each stage of the WebGPU FFRNN algorithm, plotted according to Table 5.3. Both axes are logarithmic.

	Inserting		Scanning (prefix sum)					Sorting			Collision	
# Atoms	Fill-0	Insert	Scan 1	Scan 2	Scan 3	Add 1	Add 2	Fill-`null`	Sort	Map	detection	Rendering
566	1.59	0.01	0.63	0.01	0.01	0.004	0.29	0.003	0.01	0.004	0.03	0.004
53915	1.48	0.02	0.59	0.01	0.01	0.003	0.33	0.003	0.02	0.008	0.10	0.042
107030	1.48	0.04	0.59	0.01	0.01	0.003	0.33	0.003	0.03	0.016	0.15	0.078
212264	1.30	0.06	0.52	0.01	0.01	0.004	0.29	0.003	0.04	0.034	0.26	0.155
424528	1.31	0.11	0.52	0.01	0.01	0.004	0.29	0.005	0.10	0.072	0.49	0.305
1061320	1.30	0.23	0.52	0.01	0.01	0.003	0.29	0.012	1.15	0.173	1.16	0.754
2122640	1.60	0.36	0.63	0.01	0.01	0.004	0.29	0.023	3.81	0.332	2.68	1.505
3183960	1.30	0.40	0.52	0.01	0.01	0.004	0.29	0.032	6.42	0.445	3.13	2.884
4245280	1.31	0.43	0.52	0.01	0.01	0.004	0.29	0.042	11.74	0.539	4.03	3.198

Table 5.3: Results of the performance of each stage of the WebGPU FFRNN algorithm. Values provided for each pipeline stage represent the run time in milliseconds.

For the measures with 566 atoms, a third nanostructure was used, with PDB code `1BNA`. Despite its small size, the stages that operate over each grid cell (*Fill-0*, *Scan*, and *Add*) take virtually the same amount of time to compute as with very large scenes, as a fixed grid size was used. This is noticeable in Figure 5.9, where these operations are represented by a flat line.

The operations with purely linear algorithmic complexity $O(n)$ on the number n of atoms $O(n)$ (i. e., *Fill-*`null`, *Map*, and *Render*), also scale as expected, with linear growth. The remaining operations (*Insert*, *Sort*, and *Detect collisions*) also follow the expected pattern, which additionally depends on the density $max(k)$ of atoms (i. e., the largest number of atoms contained in any single cell).

Since the number of cells in the grid and their size are kept at fixed values across the experiments, the density of atoms increases together with the number of atoms. For this reason, *Insert*, *Sort*, and *Detect collisions* are expected to follow a slight exponential growth. In the experiments conducted, this effect is not visible for *Insert* and *Detect collisions*, but very pronounced for *Sort*. This can be explained by the fact that, during sorting, a large number of atoms that are located in the same cell execute several atomic operations (`atomicExchange`) at the same time, on the same buffer location. This leads to a loss of efficient parallelism of this stage, as concurrent atomic operations on the same variable are sequential-like. *Insert* also relies on atomic operations (`atomicAdd`). The difference is that the grid cells being manipulated at this stage are more spread-out, due to the fact the atoms' spatial proximity does not correspond to their memory proximity yet.

The results obtained are comparable with those presented by Pauschenwein [Pau22], as is shown in Table 5.4. The implementations scale similarly, with the brute force approach scaling quadratically (quickly passing the real-time threshold), while FFRNN scales linearly. This work was able to achieve much lower run times overall. However, this can be attributed to the difference in GPU hardware,

	[Pau22]		This work	
∼ # Atoms	**BF**	**FFRNN**	**BF**	**FFRNN**
50,000	61.00	9.00	10.24	2.56
100,000	280.00	10.00	30.98	2.66
200,000	977.00	11.00	99.95	1.21
400,000	4,530.00	16.00	322.37	2.91
1,000,000	25,392.00	28.00	2,024.98	4.85
2,000,000	102,604.00	44.00		9.75
4,000,000	599,236.00	78.00		18.91

Table 5.4: Comparison of performance results (in milliseconds) of this work (NVIDIA GeForce RTX 3070 Laptop GPU) with [Pau22] (NVIDIA Geforce GTX 970).

5.4.3 Collision Rendering

As shown in Table 5.2, the collision rendering times are a small fraction of the total computation time. However, as mentioned in Section 5.3 and shown in Figure 5.4, the collision markers do not share the same `canvas` element as the rest of the molecular visualization. In order for them to share the same scene (and therefore the same depth information), the collisions calculated with WebGPU are transferred to WebGL and rendered, as shown in Figure 5.5.

This operation, however, is considerably slower than rendering the collisions directly in WebGPU. Table 5.5 shows a comparison of the runtimes of the two methods. Its values are plotted in Figure 5.10. Because of this, and in favour of user satisfaction [RR86], the WebGPU rendering is used while structures are being moved by the user. Once the movement (user interaction) stops, the WebGL rendering is triggered, and the delay of the WebGL approach becomes imperceptible.

# Atoms	WebGPU	WebGL
107,030	0.078	7.204
212,264	0.155	10.672
424,528	0.305	16.976
1,061,320	0.754	36.392
2,122,640	1.505	80.772
3,183,960	2.884	135.592
4,245,280	3.198	208.184

Table 5.5: Results of the performance of rendering atom collisions with WebGPU (directly from the implemented FFRNN pipeline) against rendering atom collisions using WebGL. The WebGL times include the time that it takes to download the collision data from the WebGPU API, process it in the CPU, and upload it to the WebGL API. This means that the rendering time of WebGL is excluded from this benchmark. Values provided represent the run time in milliseconds.

5.5 Discussion

As for RQ2. (*How can the novel MolCAD constraint-based interaction methods be implemented efficiently (in real-time) in a web-based environment?*), the benchmarks show that the research question can be answered as follows: The PCA-based alignment technique can be implemented efficiently on the CPU using singular value decomposition. Despite this implementation not resulting in real-time results, the results achieved are considered sufficient due to the fact that there is no real-time constraint in this operation, as it does not need to occur every frame, but it can be pre-computed on demand. The collision detection framework can also be implemented in real-time using the novel WebGPU API. Both solutions were shown to run efficiently in a web-based environment. All results were

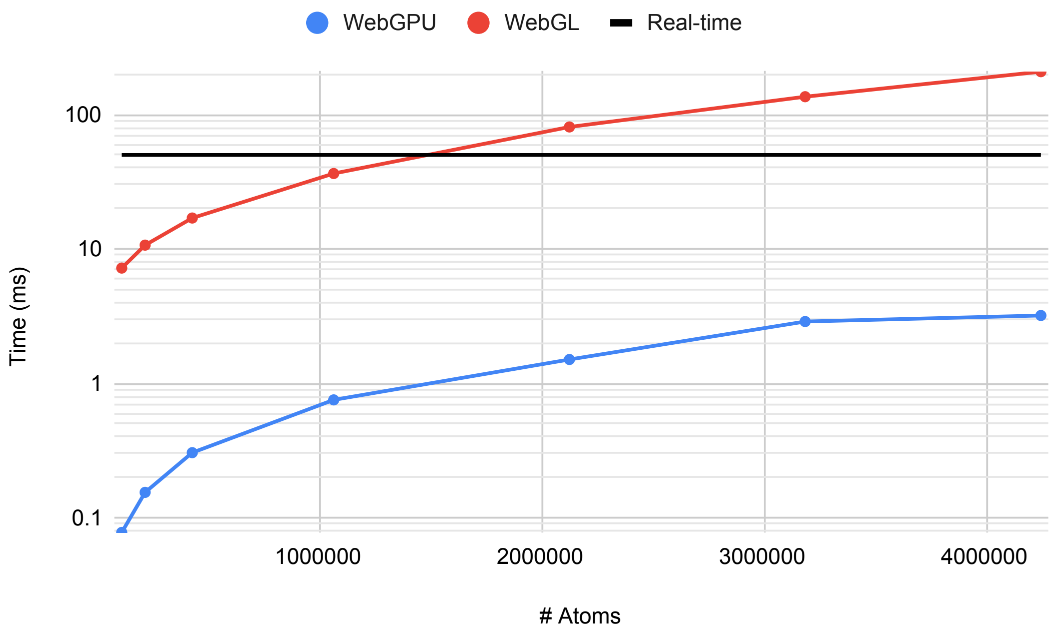


Figure 5.10: Results of the performance of rendering atom collisions with WebGPU against WebGL, plotted according to Table 5.5. The Y-axis is logarithmic.

determined based on a data scale that is relevant to the uses witnessed in the preliminary interviews.

CHAPTER 6

Case Studies

In order to be able to answer RQ3. (*Which of the novel MolCAD constraint-based interaction methods increases user satisfaction and efficiency for constraint-based MolCAD?*), we must first determine how to measure user satisfaction and efficiency. The objective of validating a system is to determine whether it benefits the domain experts. This can be achieved if users are able to perform tasks faster, more correctly, and/or with less workload, as well as when the tasks were previously impossible [Rot06].

This qualitative validation is achieved through case studies. The case study method consists of focusing on a select group of participants and observing them work in their routine tasks while conducting interviews and accompanied by automated logging. This method allows for a detailed insight into how domain experts use specific tools or interfaces, including what they are able to achieve and the challenges they face [SP06].

The interviews are conducted in a semi-structured manner. This style of interview combines a set of pre-defined open-ended questions which are aimed at initiating a discussion. With this method, participants are able to speak freely while being steered in a direction that enables validation of the software tools implemented. In the following sections, the entire process (from planning to executing) of the semi-structured user interviews is presented, as well as the user feedback.

6.1 Procedure, Setup, and Participants

As semi-structured interviews were conducted, a rough outline of the interview was planned out, with space planned for the participating users to fully express themselves and ask questions. The interviews are conducted in part only verbally, and in part while interacting with Catana [KMS$^+$22]. Because Catana is used, one point where special attention is needed is to steer users into the particular parts of Catana we are interested in (i. e., the design challenges presented in Section 4.4). Special focus is also given to the user's experience with previous tools and their interaction methods analogue to this thesis' requirements, how they compare to each other, while encouraging the interviewees to contribute with new ideas about what is still missing.

While users interact with Catana and solve their typical domain-specific tasks using their own data (both crucial requirements for the Design Study Methodology [Rot06]), we encourage them to speak aloud and describe what their goal is, how they are achieving it, and share any difficulties they may come across. Support is also given at any time in case users need assistance in finding a specific tool in the user interface.

Before each interview, the user is asked to prepare the data they currently use for their design/modelling tasks, or provide a list for the interviewer to prepare it. The semi-structured interview questions are listed in the Appendix.

The implemented PCA alignment and real-time collision detection with WebGPU were available for the users to interact with. The users had the knowledge that these tools existed, since the users accompanied the development process of both these methods. Because of this, the tools were not explained immediately prior to the interview.

The interviews were conducted in person using a laptop with a an Intel(R) Core(TM) i7-6500U processor with 2 cores (4 logical processors) running at 2.5 GHz, using 8GB of RAM, running Windows 10, with a AMD Radeon(TM) R5 M330 GPU. Internet connection was provided (important for PDB access), and screen capture, as well as audio recording for later reference, together with written notes.

The two participants of this case study are experienced in molecular modelling and tasks that require alignment and/or collision detection. That is because these two tasks were established in this thesis' work to be the most crucial points that are lacking in currently established software. This is crucial in order for them to compare the novel tools (and their experience with them) with tools that are already established, and thus be able to express whether and how the introduced tools are an improvement. Both domain experts also participated in the preliminary interviews presented in Section 4.1.

6.2 Interviews

Both participants were asked to perform a task they recently or routinely perform with their own data. The tasks and data are listed below.

Interview 1, Task 1 The task requires the following structures from the PDB: `3UGM` and `1W4W`. The `3UGM` structure is loaded and its DNA is removed. Next, a new DNA strand with a different sequence is created and aligned to the remaining protein of the `3UGM` structure. This DNA-protein hybrid structure is when duplicated. A new DNA strand is created to connect the DNA strands of each of the twin structured. Then, the `1W4W` structure is loaded and split into two parts. Finally, each part is connected to a terminus of each of the twin structures. While aligning the structures, the participant makes sure to avoid collisions between the structures. The result is shown in Figure 6.1 and contains 19,404 atoms.

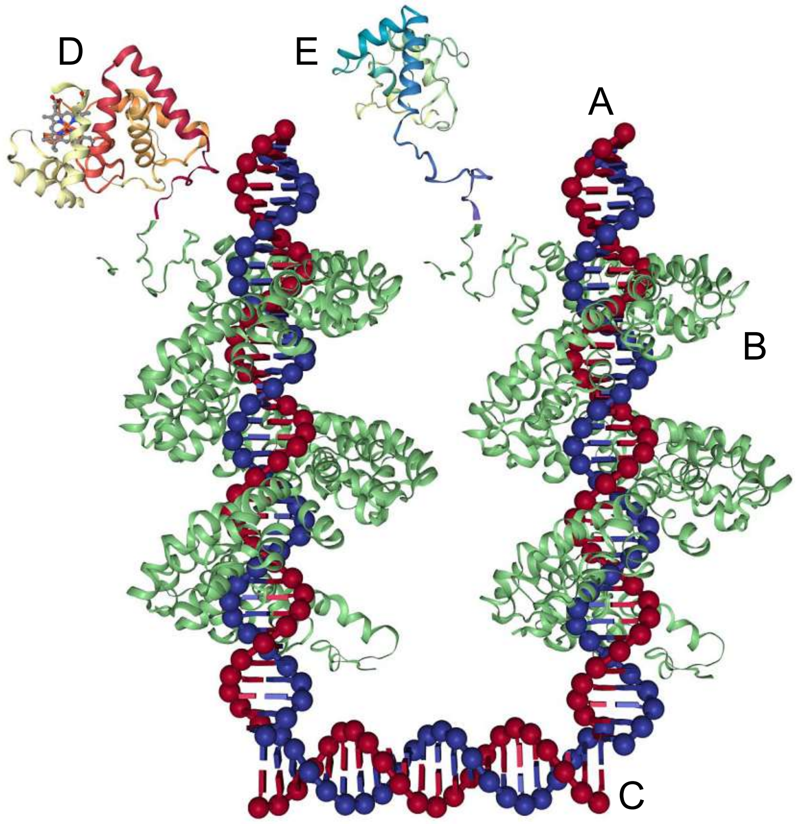


Figure 6.1: The first structure produced in Interview 1 (Section 6.2). Two identical DNA strands (A), each of them being wrapped/coiled by a protein (B) from the PDB structure `3UGM`. A DNA strand (C) connects the two identical DNA strands (A). Two split parts (D, E) of the protein `1W4W` from the PDB are connected to the termini of each of the identical `3UGM` proteins (B).

Interview 1, Task 2 The task requires the following structure from the PDB: `4TSZ`, as well as a large DNA-origami structure with 662,417 atoms. While aligning the `4TSZ` structure with the DNA-origami structure, the participant makes sure to avoid collisions between the structures. This task is illustrated in Figure 6.2.

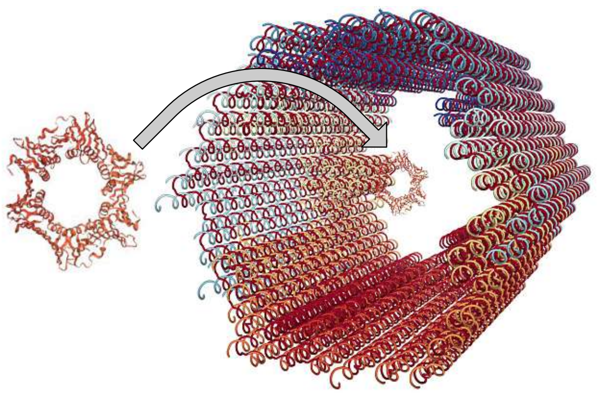


Figure 6.2: The second structure produced in Interview 1 (Section 6.2). A pore (structure `4TSZ` from the PDB) is aligned with a large hexagonal DNA-origami structure.

Interview 2 The task requires the following structure from the PDB: `5DO4`. Several DNA strands with different sequences are created in a brick-like structure. Then, the four `5DO4` proteins are loaded and connected to the ends of four of the DNA strands created previously. The result is shown in Figure 6.3 and contains 38,524 atoms.

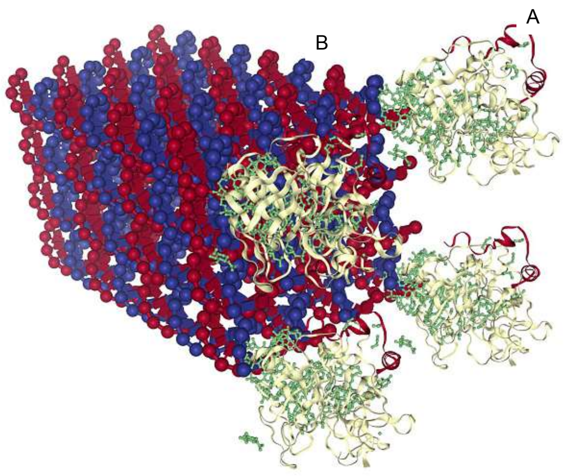


Figure 6.3: The structure produced in Interview 2 (Section 6.2). A DNA-origami structure resembling a brick (B) has several of its DNA strands connected with proteins (A) from the PDB structure `5DO4`.

The qualitative feedback collected from the two semi-structured interviews is categorized with open coding extracted from the transcribed feedback of the participants. The coded answers are listed in the Appendix. For both interviews, the feedback received was categorized using open coding. The five feedback categories are listed and described below.

- **Alignment in MolCAD tools**: How users solve DC1. (alignment) with other tools.
- **Collision detection in MolCAD tools**: How users solve DC2. (collision detection) with other tools.
- **PCA alignment**: Are users more satisfied and can they work more efficiently with PCA alignment?
- **WebGPU collision detection**: Are users more satisfied and can they work more efficiently with collision detection?
- **What is still missing**: What other possibilities users can think of to increase satisfaction and efficiency?.

Alignment in MolCAD Overall, we observed that the participants seemed to not feel confident with the alignment of structures in other tools. In Interview 1, the participant reported having used PyMol [SD] for alignment, but only its structural superposition feature (as introduced in Section 4.2). The participant also reported having used Catana [KMS+22] for alignment (both its translation/rotation tools and its structural superposing feature). In Interview 2, the participant reported having used SAMSON Connect [SAM] previously, despite its loss of real-time performance when large structures are added (such as the one shown in Figure 6.2). Like the participant of Interview 1, this participant also reported having used Catana.

Collision detection in MolCAD tools Overall, we observed that the participants do not have a clear idea of how to perform collision detection using established MolCAD software. In Interview 1, the participant expressed that their common method of estimating collisions is visual. The participant also expressed a likelihood that PyMol has this feature, but a lack of certainty that they ever tried it before, but confidence that it would not be a real-time feature. In Interview 2, the participant stated that they have previously only estimated collisions visually, and that in some scenarios it is impossible to do that (such as the one illustrated in Figure 4.7). The participant also shared their experience with domain experts of biophysics and that they have “a trick” to detect collisions.

PCA alignment Overall, we observed that the participants did not benefit greatly from the novel PCA-based alignment method. In Interview 1, the participant did not consider the PCA method when aligning structures, despite task 1 6.2 being an ideal case for it. However, when showed the capability of the tool, they admitted its usefulness in limited cases, such as when modelling DNA-protein hybrid structures. In Interview 2, the participant used the novel tool in the task, but we observed that they seemed confused about its use and capabilities. The user gave up after a few seconds and moved on to try the novel collision detection feature.

WebGPU collision detection Overall, we observed that the participants were very impressed with the novel collision detection method, and both saw potential in it contributing to increased user satisfaction and efficiency. In Interview 1, the participant seemed impressed with the tool and expressed their opinion about its potential use, especially when working with large proteins. The participant also added that, for estimating how close together two structures can get, MDS (a computationally expensive task) would be needed without a tool to calculate collisions. In Interview 2, the participant also seemed impressed. The participant also expressed how this tool has the potential to detect and prevent collisions early before sending a structure to MDS (increasing efficiency), and how the visual output facilitates this process (increasing user satisfaction).

What is still missing Overall, we observed three trends in the feedback from the participants regarding what is still missing in MolCAD: (1) an "undo" button, (2) the support of artificial intelligence methods, and (3) more or improved alignment tools. Both participants stressed the need for the "undo" feature, which allows users to undo previously performed commands (especially useful when the user makes a mistake). In Interview 1, the participant suggested a simple new alignment method that would potentially facilitate all tasks performed in the interview: by selecting two points, each on a separate structure, one structure would be moved/translated such as the points are in the same position (e. g., in Figure 4.3, the two selected points would be structure A's terminus and structure B's terminus, which would then be aligned). Additionally, the participants stated that protein structures predicted with AlphaFold [JEP+21] (an artificial intelligence tool that predicts how proteins fold based on their amino acid sequence) facilitated their work, as they showed to be accurate, and change very little in structure when sent to simulations. In Interview 2, the participant further stressed the rising need for artificial intelligence-based tools, giving an example of a novel system that, for a given protein, matches it with other proteins that can bind to it.

6.3 Discussion

Based on the findings of the interviews, PCA alignment did not leave a strong impact on users. Both interviewees did not understand how PCA could be useful to align structures, instead preferring existing alignment techniques previously available in other software, including Catana [KMS+22], such as structural superposition. This occurred despite prior communication regarding the tool's capabilities, making it evident that PCA poses challenges for user comprehension. However, after being shown cases where

PCA produces better results than structural superposition, users agreed that it can make their workflow less frustrating and more efficient in certain scenarios.

One such scenario is when aligning structures with very different compositions. The users highlighted that structural superposition (an alternative to PCA alignment) is only applicable if two structures are very similar, as it takes the structure's sequence into account. However, proteins and DNA have different types of sequences (amino acid and nucleotide, respectively). This may lead to structural superposition not supporting protein/DNA-hybrid MolCAD. The implemented PCA method, on the other hand, works universally, as it uses atom data, which is present regardless of the type of structure.

In contrast, collision detection emerged as a strong feature. Both users reported being satisfied with it. The capacity of this functionality to efficiently locate molecular collisions in a dynamic context was considered essential by interviewees. They emphasized its potential in evaluating the structural plausibility of their molecular models, a necessary step before moving on to MDS.

Finally, the insights gathered from the interviews allow us to address RQ3. (*Which of the novel MolCAD constraint-based interaction methods increases user satisfaction and efficiency for constraint-based MolCAD?*). As user feedback showed, PCA-based alignment has the potential to increase user satisfaction and efficiency in certain situations, while real-time collision detection showed strong signs to be an essential tool for increased satisfaction and efficiency.

CHAPTER 7

Discussion

In this thesis, an approach to MolCAD was introduced through the implementation of constraint-based interaction techniques on the web. These techniques were developed to overcome the complexities inherent in molecular modelling tasks and make them more usable and efficient. A review of existing methods in the field was conducted to develop this constraint-based interaction system, where established CAD and MolCAD interactions were surveyed, in order to find out which CAD aspects the MolCAD field is missing out on (RQ1.). These were then described as the design challenges of this thesis in Section 4.4: alignment DC1. and collision detection DC2..

Each design challenge then led to a technical challenge, and the PCA-based alignment was implemented, alongside a WebGPU-based collision detection system. With that, it was shown that their implementation is not only feasible in a web-based environment, but also efficient (RQ2.). Additionally, through case studies, this thesis demonstrated that these constraint-based interactions can enhance user experience and provide a more intuitive workflow for complex molecular design tasks (RQ3.). Moreover, the answers to the research questions can be summarised as follows:

1. RQ1. *Which CAD interaction techniques exist that are also applicable for MolCAD and are currently not in use yet, but could be useful?* Constraint-based alignment tools, commonly present in CAD solutions, hold great potential for MolCAD. Based on the interviews conducted, two such techniques are chosen to be implemented: a PCA-based alignment method, and a real-time collision detection algorithm.

2. RQ2. *How can the novel MolCAD constraint-based interaction methods be implemented efficiently (in real-time) in a web-based environment?* The PCA-based alignment method has no real-time constraint, as it can be pre-computed on demand. Its CPU implementation on a web-based environment yielded sufficient performance

for the user's flow of thought to not be interrupted [Nie93]. The collision detection algorithm achieved real-time performance in a web-based environment with a WebGPU [Webb] implementation.

3. RQ3. *Which of the novel MolCAD constraint-based interaction methods increases user satisfaction and efficiency for constraint-based MolCAD?* The PCA-based alignment tool received mixed feedback from users at first due to a lack of understanding of how it works and what it can achieve. When presented with examples of its use, users admitted its usefulness. In contrast, the interviewed users were pleased with the real-time collision detection tool and emphasized its practical utility. From the case studies, it can be concluded that PCA-based alignment holds potential in increasing user satisfaction and efficiency, (but requires explicit communication of its benefits), while collision detection shows strong signs of being an essential tool for increased satisfaction and efficiency.

While the results of this thesis show two promising MolCAD techniques, several aspects emerged that hold potential for future work and improvement in MolCAD tools. These are summarised below.

Alignment design challenge The alignment design challenge (DC1.) stems from the dichotomy between the 3D nature of molecular scenes and the 2D constraints of computer displays. This thesis explored PCA-based alignment as a potential solution. However, feedback from case studies indicates a mixed reception from users. They expressed confusion about its utility until they were shown examples of its efficacy in tasks that previous features could not address efficiently. As a future direction, other alignment methods, such as "snapping" [Bie], could be explored to find improved solutions to this challenge.

Collision handling The real-time collision detection tool was received well by users in the case studies. However, an addition to this system can be made that has the potential to further improve the workflow of users: automatically solving the collisions. Without collision handling, users must manually find all collisions and push atoms apart, a task that becomes unfeasible for structures with millions of atoms. This feature has been shown to work in real-time [Pau22] using OpenGL, and it has the potential to be implemented on the web as well, using WebGPU.

Undo button Feedback from the case studies highlighted the importance of an "undo" button in MolCAD. And "undo" button allows the user to revert actions, which is especially crucial in complex tasks.

Artificial intelligence methods Another aspect stressed by the case study participants is the growing use of artificial intelligence systems in MolCAD. The users expressed satisfaction with AlphaFold [JEP+21], a tool that is already integrated into

Catana [KMS$^+$22], and mentioned the potential to improve their workflow further with more such tools. This shows that not only there is potential for future work in developing these systems, but also that their integration in MolCAD software has the potential to improve user experience.

Larger and more elaborate case studies While the two case studies conducted for this thesis were sufficient to answer RQ3., there is potential for broader and more detailed studies in future work. Users of MolCAD can have very different backgrounds: molecular biologists, biophysicists, chemists, structural biologists, biotechnologists, etc. Because of that, they have diverse user needs. Understanding these needs gives MolCAD tools the potential to rival established CAD software in terms of user satisfaction and efficiency.

Formal comparison While the tools explored and implemented in this work have been shown to increase user satisfaction and efficiency on its own, there remains a need to compare these results with established CAD as well as MolCAD software. This step is crucial to understanding two aspects of the newly implemented methods: (1) whether they are comparable with well-established CAD tools, and (2) whether they outperform well-established MolCAD software. A comparison with existing virtual reality MolCAD software (such as [KSB$^+$21]) also holds potential for insights on the efficacy of the presented techniques.

List of Figures

List of Tables

List of Algorithms

Acronyms

CAD Computer-Aided Design. vii, ix, 1, 2, 8, 9, 13, 17–19, 21–23, 30, 59, 61, 63, 64

FFRNN Fast Fixed-Radius Nearest Neighbor. 28, 29, 34, 36, 40, 41, 43–49, 67, 69, 71

MDS Molecular Dynamics Simulations. 1, 2, 5, 8, 56, 57

MolCAD Computer-Aided Molecular Design. vii, ix, 1–3, 5, 8–14, 16, 18, 20, 23, 24, 27, 30, 31, 33, 41, 49, 51, 55–57, 59–61, 64, 81, 82

PCA Principal Component Analysis. vii, ix, 22–28, 30–33, 41–43, 49, 55–57, 59, 60, 65, 66, 69, 71, 82

PDB Protein Data Bank. 6, 12, 15–17, 20–22, 25, 27, 28, 30, 33, 41, 42, 48, 52–54, 63–67

SIMD Single Instruction Multiple Data. 32, 37

Bibliography

[ANW+20] Yasaman Ahmadi, Ashley L. Nord, Amanda J. Wilson, Christiane Hütter, Fabian Schroeder, Morgan Beeby, and Ivan Barišić. The brownian and flow-driven rotational dynamics of a multicomponent DNA origami-based rotor. *Small*, 16(22):2001855, may 2020.

[ASF+13] Murat Arikan, Michael Schwärzler, Simon Flöry, Michael Wimmer, and Stefan Maierhofer. O-snap. *ACM Transactions on Graphics*, 32(1):1–15, jan 2013.

[Auta] Autocad. `https://www.autodesk.com/products/autocad/overview`. Accessed: 2023-09-04.

[Autb] Autodesk research. `https://www.research.autodesk.com/publications`. Accessed: 2023-03-08.

[Bie] Eric A. Bier. Snap-dragging in three dimensions. In *Proceedings of the 1990 symposium on Interactive 3D graphics - SI3D '90*. ACM Press.

[Ble] Blender. `https://www.blender.org/`. Accessed: 2023-09-04.

[BYK+21] Lonni Besançon, Anders Ynnerman, Daniel F. Keefe, Lingyun Yu, and Tobias Isenberg. The state of the art of spatial interfaces for 3D visualization. *Computer Graphics Forum*, 40(1):293–326, jan 2021.

[CSH+] Brookshire D. Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In *Proceedings of the 1992 symposium on Interactive 3D graphics - SI3D '92*. ACM Press.

[CVB09] Mohamed Chaouch and Anne Verroust-Blondet. Alignment of 3D models. *Graphical Models*, 71(2):63–76, mar 2009.

[DDL+09] Shawn M. Douglas, Hendrik Dietz, Tim Liedl, Björn Högberg, Franziska Graf, and William M. Shih. Self-assembly of DNA into nanoscale three-dimensional shapes. *Nature*, 459(7245):414–418, may 2009.

[dLMA$^+$20] Elisa de Llano, Haichao Miao, Yasaman Ahmadi, Amanda J Wilson, Morgan Beeby, Ivan Viola, and Ivan Barisic. Adenita: interactive 3D modelling and visualization of DNA nanostructures. *Nucleic Acids Research*, 48(15):8269–8275, jul 2020.

[DLS20] David Doty, Benjamin L Lee, and Tristan Stérin. scadnano: A browser-based, scriptable tool for designing DNA nanostructures. In *26th International Conference on DNA Computing and Molecular Programming*. arXiv, 2020.

[DMT$^+$09] Shawn M. Douglas, Adam H. Marblestone, Surat Teerapittayanon, Alejandro Vazquez, George M. Church, and William M. Shih. Rapid prototyping of 3D DNA-origami shapes with caDNAno. *Nucleic Acids Research*, 37(15):5001–5006, jun 2009.

[Ger] Jan J. Gerbrands. On the relationships between SVD, KLT and PCA, year = 1981, issn = 0031-3203, note = 1980 Conference on Pattern Recognition, number = 1, pages = 375-381, volume = 14, doi = https://doi.org/10.1016/0031-3203(81)90082-0, keywords = Image processing, Statistical analysis, Statistical pattern recognition, Orthogonal image transforms, Singular value decomposition, Karhunen-Loeve transform, Principal components, url = https://www.sciencedirect.com/science/article/pii/0031320381900820,. *Pattern Recognition*.

[Han97] Chris Hand. A survey of 3D interaction techniques. *Comput. Graph. Forum*, 16:269–281, 12 1997.

[HDS96] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[HvDG94] Kenneth P. Herndon, Andries van Dam, and Michael Gleicher. The challenges of 3D interaction. *ACM SIGCHI Bulletin*, 26(4):36–43, oct 1994.

[JEP$^+$21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, Alex Bridgland, Clemens Meyer, Simon A. A. Kohl, Andrew J. Ballard, Andrew Cowie, Bernardino Romera-Paredes, Stanislav Nikolov, Rishub Jain, Jonas Adler, Trevor Back, Stig Petersen, David Reiman, Ellen Clancy, Michal Zielinski, Martin Steinegger, Michalina Pacholska, Tamas Berghammer, Sebastian Bodenstein, David Silver, Oriol Vinyals, Andrew W. Senior, Koray Kavukcuoglu, Pushmeet Kohli, and Demis Hassabis. Highly accurate protein structure prediction with AlphaFold. *Nature*, 596(7873):583–589, July 2021.

[JH12] Jacek Jankowski and Martin Hachet. A survey of interaction techniques for interactive 3D environments, 2012.

[KAK+18] Tobias Klein, Ludovic Autin, Barbora Kozlikova, David S. Goodsell, Arthur Olson, M. Eduard Groller, and Ivan Viola. Instant construction and visualization of crowded biological environments. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):862–872, jan 2018.

[KMS+22] David Kuťák, Lucas Melo, Fabian Schroeder, Zoe Jelic-Matošević, Natalie Mutter, Branimir Bertoša, and Ivan Barišić. CATANA: an online modelling environment for proteins and nucleic acid nanostructures. *Nucleic Acids Research*, may 2022.

[KSB+21] David Kutak, Matias Nicolas Selzer, Jan Byska, Maria Lujan Ganuza, Ivan Barisic, Barbora Kozlikova, and Haichao Miao. Vivern a virtual environment for multiscale visualization and modeling of DNA nanostructures. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021.

[MCG+18] D. Mendes, F. M. Caputo, A. Giachetti, A. Ferreira, and J. Jorge. A survey on 3D virtual object manipulation: From the desktop to immersive virtual environments. *Computer Graphics Forum*, 38(1):21–45, apr 2018.

[Met] Metal. `https://developer.apple.com/metal/`. Accessed: 2023-10-01.

[Ngu07] Hubert Nguyen. *Gpu Gems 3*. Addison-Wesley Professional, first edition, 2007.

[Nie93] Jakob Nielsen. Response times: the three important limits. *Usability Engineering*, 1993.

[Nvi] NVIDIA GeForce RTX 40 series. `https://www.nvidia.com/en-gb/geforce/graphics-cards/40-series/`. Accessed: 2023-09-01.

[Ope] Opengl. `https://www.opengl.org/`. Accessed: 2023-09-04.

[Pau22] Johannes Pauschenwein. *Real-Time Collision Detection and Handling of Molecules for Animation and Modelling*. Bachelor's thesis, Research Unit of Computer Graphics, Institute of Visual Computing and Human-Centered Technology, Faculty of Informatics, TU Wien, Favoritenstrasse 9-11/E193-02, A-1040 Vienna, Austria, January 2022.

[PDB] Protein data bank specificatios. `https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/tutorials/pdbintro.html`. Accessed: 2023-10-01.

[PGH+04] Eric F. Pettersen, Thomas D. Goddard, Conrad C. Huang, Gregory S. Couch, Daniel M. Greenblatt, Elaine C. Meng, and Thomas E. Ferrin. UCSF chimera – a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[RBV$^+$18] Alexander S Rose, Anthony R Bradley, Yana Valasatava, Jose M Duarte, Andreas Prlić, and Peter W Rose. NGL viewer: web-based molecular graphics for large complexes. *Bioinformatics*, 34(21):3755–3758, may 2018.

[Rot06] Paul W. K. Rothemund. Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302, mar 2006.

[RR86] Avi Rushinek and Sara F. Rushinek. What makes users happy? *Communications of the ACM*, 29(7):594–598, jul 1986.

[SAM] Samson connect. `https://www.samson-connect.net`. Accessed: 2023-09-04.

[SD] LLC Schrödinger and Warren DeLano. Pymol. `http://www.pymol.org/pymol`. Accessed: 2023-09-04.

[SDRP16] Samira Sadeghi, Thomas Dargon, Louis Rivest, and Jean-Philippe Pernot. Capturing and analysing how designers use cad software. *Tools and Methods for Competitive Engineering (TMCE'16)*, dec 2016.

[Ske] Sketchup. `https://www.sketchup.com/`. Accessed: 2023-09-04.

[SMM12] Michael Sedlmair, Miriah Meyer, and Tamara Munzner. Design study methodology: Reflections from the trenches and the stacks. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2431–2440, dec 2012.

[SMS$^+$16] B. Steiner, E. Mousavian, F. Mehdizadeh Saradj, M. Wimmer, and P. Musialski. Integrated structural-architectural design for interactive planning. *Computer Graphics Forum*, 36(8):80–94, oct 2016.

[SP06] Ben Shneiderman and Catherine Plaisant. Strategies for evaluating information visualization tools. In *Proceedings of the 2006 AVI workshop on BEyond time and errors: novel evaluation methods for information visualization*. ACM, May 2006.

[SPG18] Cornelia Johanna Franziska Scheitz, Lawrence J Peck, and Eli S Groban. Biotechnology software in the digital age: are you winning? *Journal of Industrial Microbiology and Biotechnology*, 45(7):529–534, jul 2018.

[SW10] Wolfgang Stuerzlinger and Chadwick A. Wingrave. The value of constraints for 3D user interfaces. In *Virtual Realities*, pages 203–223. Springer Vienna, oct 2010.

[thr] three.js. `https://threejs.org/`. Accessed: 2023-09-04.

[vRO$^+$12] Petr Šulc, Flavio Romano, Thomas E. Ouldridge, Lorenzo Rovigatti, Jonathan P. K. Doye, and Ard A. Louis. Sequence-dependent thermodynamics of a coarse-grained DNA model. *The Journal of Chemical Physics*, 137(13):135101, 2012.

[Vul] Vulkan. `https://www.vulkan.org/`. Accessed: 2023-10-01.

[Weba] Webgl. `https://www.khronos.org/api/webgl`. Accessed: 2023-09-01.

[Webb] Webgpu working draft. `https://www.w3.org/TR/webgpu/`. Accessed: 2023-09-04.

[WPP11] Robert Wang, Sylvain Paris, and Jovan Popović. 6d hands. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, oct 2011.

Appendix

Semi-structured interview questions

The following questions were used a a basis to conduct the final case studies presented in Chapter 6.

1. Understand interviewee's current work and experience:
   a) What are you currently working with that requires *in-silico* molecular design?
   b) What tools are you currently using? or in case they use Catana;
   c) Which tools did you use before Catana?
2. Understand interviewee's modelling/design workflow and wishes:
   a) What modelling/design tasks are you currently working on?
   b) How do you achieve that with established software? (not Catana)
   c) What could be improved? (encourage interviewee to contribute with ideas)
   d) Present our ideas, then ask: Do you think they could be helpful? How?
3. Observe interviewee's workflow:
   a) Could you please show us in Catana what your work is about? (encourage to explain what they're doing and what their goals are)
   b) Can you perform all required actions? Or is any external software needed?
   c) Are you having difficulties? (provide assistance whenever necessary)
4. Reflect on interview and implemented MolCAD tools:
   a) What challenges did you encounter?
   b) Which steps were particularly tedious, frustrating, and/or time-consuming?
   c) What do you think was lacking that could have made your experience easier?
   d) How would you fix/implement it? (encourage them to let their imagination run free, not constrained by what they think is technically possible/feasible)

e) What would you gain from it? (e. g., faster or easier interaction, new possibilities previously impossible)

5. Compare interview's experience with experience with other MolCAD software:

   a) Come back to beginning of the interview and the other software the interviewee used previously. How does it compare with Catana?
   b) What was harder?
   c) What was slower?
   d) What was easier?
   e) What was faster?

Open coding from case studies

The feedback gathered from the case studies presented in Chapter 6 were categorized with open coding. Below, the categories and the respective answers of the participants are listed.

1. **Alignment in MolCAD tools**: How users solve DC1. (alignment) with other tools

   a) "In PyMol I still have no clue how to do it. You can zoom in and stuff, but to move stuff, especially in relation to something else which you have positioned, I have no clue."
   b) "I think I used only structural superimposing because I had two proteins."

2. **Collision detection in MolCAD tools**: How users solve DC2. (collision detection) with other tools

   a) "I looked a little bit [and tried to make sure] that they were not clashing. But for a moment, you introduced [a feature that lets you] see where [the structure] is clashing. That, we didn't have to this point. [...] But that would be useful to see."
   b) "I think with PyMol you can calculate it as a seed, but I'm not sure if I ever tried it. [...] In real time for sure not. You have a structure and [the collision detection algorithm] is calculating something."
   c) "When you look at the all-atom [representation], you have no idea which ones are colliding. [...] The biophysicists have tricks to [fix], but it's much easier with the visual tool and then you can select individual atoms and adjust them a little bit and avoid it."

3. **PCA alignment**: Are users more satisfied and can they work more efficiently with PCA alignment?

   a) "[...] It depends on what use case you have: if you want to align DNA or if you want to align proteins, or maybe even if you would have a peptide, I guess this could be also useful because they often have also small helical structures."
   b) When asked about alignment, the user said, referring to structural superposition: "This is a very nice tool so...". Then, they try to align a protein with a DNA strand but fail. Then, the user selects the principal axes option and the result is unexpected to the user (protein goes inside DNA). Later, when queried, the user stated that they did not know what the principal axes tool means or how it works.

4. **WebGPU collision detection**: Are users more satisfied and can they work more efficiently with collision detection?
   a) "Oh yeah, now it's clashing. [...] That is cool. It's a little bit like playing games."
   b) "This way that you really move it and see where it's clashing. I'm sure that's not available. [...] That is quite fun. Especially if you have bigger proteins and want to do fusion and stuff."
   c) "Yeah, we would like to have it most probably closer together that we have less link and whatever. When doing here, we cannot assume how close it really can come together. For this, we would need the [simulations]."
   d) "I think it's cool. Because especially if you do this fusion, it is an issue that you do not overlap structures or anything."
   e) "[This is useful] because, when we were drawing the artificial systems, we couldn't go through all the molecular dynamics simulations, because when atoms were too close, they were exploding. And, especially in such a dense [system], you have no idea which atoms are colliding. [...] It's much easier with a visual tool."

5. **What is still missing**: What other possibilities users can think of to increase satisfaction and efficiency
   a) "Undo button [...] That would make life much easier because this way either you try to undo manually whatever you did, but often you start clicking and then whatever happened and... no clue. [...] And depending which changes and how you save people and that can be painful."
   b) "The new structures, which were predicted with AlphaFold, work better than what we do manually. If we add, for example, a HIS-tag [...] at one end, that was more accurate and for [the biophysicists it was] easier to do the model as the simulations on."
   c) "Sometimes I think a grid or something would be nice to see where you end up."

d) “If I have, for example, the ends as two points and I would put a line directly, the task or something on this, that could be… [...] For example, saying I select the two ends of the protein and they get more or less tracked close together or something. That would make a lot easier. You would need less moving around, but it’s...”