MolecularRift

A Gesture Based Interaction Tool for Controlling Molecules in 3-D

Magnus Norrby

2015

Master's Thesis

Department of Design Sciences
Lund University

MolecularRift, a Gesture Based Interaction Tool for Controlling Molecules in 3-D

Magnus Norrby
mange.norrby@gmail.com

June 11, 2015

Master's thesis work carried out at AstraZeneca.

Supervisors: Jonas Boström, Jonas.Bostrom@astrazeneca.com
Joakim Eriksson, joakim.eriksson@design.lth.se

Examiner: Jonas Borell, jonas.borell@design.lth.se

Abstract

Visualization of molecular models is a vital part in modern drug design. Improved visualization methods increases the conceptual understanding and enables faster and better decision making. The introduction of virtual reality goggles such as Oculus Rift has introduced new opportunities for the capabilities of such visualisations.

A new interactive visualization tool (MolecularRift), which lets the user experience molecular models in a virtual reality environment, was developed in collaboration with AstraZeneca.

In an attempt to create a more natural way to interact with the tool, users can steer and control molecules through hand gestures. The gestures are recorded using depth data from a Mircosoft Kinect v2 sensor and interpreted using per pixel algorithms, which only focus on the captured frames thus freeing the user from additional devices such as cursor, keyboard, touchpad or even piezoresistive gloves.

MolecularRift was developed from a usability perspective using an iterative developing process and test group evaluations. The iterations allowed an agile process where features easily could be evaluated to monitor behavior and performance, resulting in a user-optimized tool.

We conclude with reflections on virtual reality's capabilities in chemistry and possibilities for future projects.

Keywords: Kinect, Virtual Reality, Oculus Rift, Gesture Recognition, Visualization, Molecular modeling

Sammanfattning

Visualisering av molekylära modeller är en viktig del inom den moderna medicinutvecklingen. Förbättrade visualiseringsmetoder hjälper till att bygga upp den konceptuella bilden av strukturen vilket leder till snabbare och bättre beslut. Produkter som Oculus Rift har öppnat för nya möjligheter att visualisera sådana modeller i en virtuell verklighet.

Ett nytt interaktivt visualiseringsverktyg (MolecularRift), som låter användaren uppleva molekylära modeller i en virtual verklighet, har utvecklats i sammarbete med AstraZeneca.

För att uppnå en naturlig interageringsmetod låter vi användaren styra och kontrollera molekylerna med handgester. Gesterna avläses med hjälp av djupdata från Microsoft Kinect v2 och tolkas med per pixel-algortimer. Per pixel-algortimerna använder sig enbart av djupdatan från sensorn och frigör därmed användaren från ytterligare externa sensorer.

MolecularRift utvecklades från ett användbarhetsperspektiv med en iterativ process och utvärderingar i testgrupper. Iterationerna gav en agil process där förändringar lätt kunde utvärderas av testgruppen och där med skapa ett förbättrat verktyg.

Arbetet avrundas med reflektioner kring virtual verklighets möjligheter inom medicinutveckling och potentiella framtida projekt.

Acknowledgements

I would like to thank Jonas Boström at AstraZeneca for making this project possible and for contributing with numerous ideas through out. I also thank Joakim Eriksson at LTH for his support with user tests and developing the virtual reality environment. I'm also grateful for the help of Christoph Grebner at AstraZeneca for answering all my chemistry question and being my go-to test pilot. Lastly I want to thank everyone who took their time participating in the user tests, the feedback was key in the development process.

Contents

1 Introduction **1**
- 1.1 AstraZeneca . . . 1
- 1.2 Goals . . . 1

2 Theoretical Background **3**
- 2.1 Virtual reality . . . 3
- 2.2 3-D Stereo in Molecular Modeling . . . 4
  - 2.2.1 Passive Stereo 3-D . . . 5
  - 2.2.2 Active Stereo 3-D . . . 6
  - 2.2.3 Autostereoscopic 3-D . . . 6
- 2.3 Virtual Reality in Molecular Modeling . . . 6
- 2.4 Computer interaction . . . 7
  - 2.4.1 Gesture recognition . . . 8
  - 2.4.2 User Experience . . . 8

3 Technical Background **10**
- 3.1 Virtual Reality with Oculus Rift . . . 10
- 3.2 Gesture recognition . . . 11
  - 3.2.1 Gesture recognition with Kinect . . . 11

4 Implementation plan **14**
- 4.1 Development Process . . . 14
  - 4.1.1 Investigation Phase . . . 14
  - 4.1.2 Gesture Interaction . . . 14
  - 4.1.3 User Tests . . . 15
- 4.2 Development Tools . . . 16
- 4.3 Setup . . . 16
- 4.4 Minimum Requirements . . . 17

5 Investigation of the SDK:s **18**
5.1 Gesture Development using Kinect . . . 18
5.1.1 Hand tracking . . . 18
5.1.2 Finger tracking . . . 20
5.1.3 Thumb tracking . . . 22
5.1.4 Proof of concepts . . . 23
5.1.5 Intended Interaction . . . 25
5.2 Combining Kinect and Oculus Rift . . . 26
5.2.1 MolyRift . . . 27
5.3 Evaluation . . . 27
5.3.1 Gesture Interaction . . . 27
5.3.2 Voice recognition . . . 28
5.3.3 Oculus Rift . . . 29

6 MolecularRift **31**
6.1 Development . . . 31
6.2 GUI . . . 31
6.2.1 In-Game Menu . . . 32
6.3 Molecular Representations . . . 32
6.3.1 Input Files . . . 32
6.3.2 Primary Structures . . . 33
6.3.3 Secondary Structures . . . 35
6.3.4 Hydrogen Bonds . . . 37
6.3.5 Protein Labels . . . 38
6.4 Gestures . . . 38
6.4.1 Movement . . . 39
6.4.2 Rotation . . . 40
6.4.3 Translation . . . 40
6.4.4 Zoom . . . 40
6.4.5 Toggle Menu . . . 40
6.4.6 Cursor Movement & Cursor Click . . . 41
6.5 Focus Target - Multiple Users . . . 41
6.6 Features . . . 41

7 Iteration One **43**
7.1 MolecularRift Improvements . . . 43
7.1.1 Performance & Motion Sickness . . . 43
7.1.2 User Feedback . . . 44
7.2 User test . . . 45
7.2.1 Questionnaire Results . . . 46
7.2.2 False positives - Unintended Interactions . . . 47
7.3 Conclusions . . . 47

8 Iteration Two **48**
8.1 MolecularRift Improvements . . . 48
8.1.1 Menu Toggle . . . 48
8.1.2 Rotation . . . 49

8.1.3 Translation . . . 49
8.2 User Test . . . 49
8.2.1 Questionnaire . . . 49
8.2.2 Questionnaire Results . . . 50
8.2.3 Suggestions From Testers . . . 50
8.3 Conclusions . . . 51

9 Iteration Three **53**
9.1 MolecularRift Improvements . . . 53
9.1.1 Focus Target . . . 53
9.1.2 Toggle Menu . . . 54
9.1.3 Auto-Movement . . . 54
9.1.4 Gesture Recognition Sensitivity . . . 55
9.2 User Test . . . 55
9.2.1 Questionnaire . . . 56
9.2.2 Questionnaire Results . . . 56
9.2.3 Toggle Menu . . . 57
9.3 Conclusions . . . 58

10 Discussion **59**
10.1 MolecularRift as a Demonstration Tool . . . 59
10.1.1 Roller Coaster . . . 59
10.2 MolecularRift as an Analytic Tool . . . 60

11 Future Work **61**
11.1 Improved Parsing . . . 61
11.2 Molecular Sculpting . . . 61
11.3 Molecular Surfaces . . . 62
11.4 Alternative Interaction Methods . . . 62
11.4.1 Leap Motion . . . 62
11.4.2 Eye Tracking . . . 64
11.5 Mixed Reality . . . 64
11.5.1 HoloLens . . . 64

Bibliography **66**

Appendix A About this Project **72**

Appendix B Code **74**

Appendix C Keyboard Commands **75**

Appendix D Iteration One **76**
D.1 Gesture Guide . . . 77
D.2 Task . . . 81
D.3 Questionnaire . . . 82
D.4 Questionnaire Answers . . . 84

Appendix E Iteration Two **88**
E.1 Questionnaire 89

Appendix F Iteration Three **90**
F.1 Questionnaire 91

Chapter 1
Introduction

In modern drug design the analysis of protein-ligand complexes is an integral part of the process. 3-D visualizations of such complexes are one important factor in the design of new drugs. By introducing virtual reality components to such visualisations drug designers would be able to "step inside" the molecular models and be able to experience and interact with them in a whole new way.

The aim of the master thesis was to design and develop a new molecular visualizer with virtual reality as the core component and investigate if and how it can add value to the design of new improved molecules.

The project was performed as a collaboration of the department of interaction design at LTH in Lund and AstraZeneca.

1.1 AstraZeneca

AstraZeneca is a global, innovation-driven biopharmaceutical business that focuses on the discovery, development, and commercialisation of prescription medicines for some of the world's most serious diseases. The work was carried out at the department of Medicinal Chemistry at AstraZeneca's site in Mölndal.

1.2 Goals

There are numerous tools for visualizing molecules, all controlled by some artificial device (e.g. cursor, keyboard, touchpad). Moreover, with recent technology advances new ways for creating highly visual 3-D environments that simulate physical presence in the real world – a virtual reality – are at hand. Thus, we are looking to explore potential benefits of these new visualization methods and to develop new interaction methods that are meaningful in the new virtual environment.

The main goals are:

- Develop a molecular visualizer that allows drug designers to view molecules and protein-ligand[1] complexes in a virtual reality.
- Develop natural and intuitive gesture-based interaction methods that allows the user to control the visualizer.
- Improve the tool, and thus the user-experience, by applying user evaluations and an iterative design and development cycle.

[1] https://en.wikipedia.org/wiki/Protein_ligand

Chapter 2
Theoretical Background

This chapter describes the history and currently available methods for virtual reality, 3-D molecular visualization, and computer interaction.

2.1 Virtual reality

Virtual reality (VR) as we know it today started to emerge in the late 1950s and early 60s. One of the pioneers was Mort Heilig with his 4-D movie invention Sensorama in 1962. The viewer could watch movies in stereoscopic 3-D synchronized with chair movement, sound, and even smell [1].

The VR field grew and finally boomed in the 90s. One of the front runners then was Jonathan Waldern and his arcade gaming company, Virtuality. The games used big head mounted displays (HMD) and many different input devices such as gloves and joysticks, all running on the Commodor Amiga 3000 [2]. One of Virtuality's arcade machines from 1991 can be seen in Figure 2.1.

Figure 2.1: One of Virtuality's arcade machines, the 1000SD. (Source: `http://www.retro-vr.co.uk/test/Images/windpics/sd1000.jpg`)

Even though the technology was new and exciting most people found the game interaction complex and often quite expensive [3]. Business was bad and commercial products disappeared from the market [3]. It was not until almost 20 years later when companies like the kickstarted[1] Oculus Rift, later bought by Facebook, brought VR back into the limelight. This time VR is being marketed largely towards home users and games by being user friendly and affordable. Just in the recent year big companies as Sony (Project Morpheus) [4] and HTC in cooperation with Valve (HTC Vive) [5] joined the trend.

2.2 3-D Stereo in Molecular Modeling

The concept of molecules as 3-D objects started to emerge during the 19th century and was later independently proven in the 1870s by Le Bel and van't Hoff [6]. The 3-D representations of molecules improved with the X-ray technology and inspired constructions of physical models such as Kendret et al. brass models of myoglobin [7]. Physical representations aid in the analysis of the molecules and were key in many important discoveries such as Watson and Crick's determination of the DNA structure in 1953 [8].

Many different representation models were developed over the years. With the rise of computers, molecular modeling switched its main focus from physical representations to simulated models. Many years passed until these simulated models would be able to display stereoscopic 3-D (stereo). An early example is the TAMS (Teaching Aids for Macromolecular Structure) project from 1980. They used two polarized slide projectors mounted on a cardboard box. The setup can be seen in Figure 2.2.

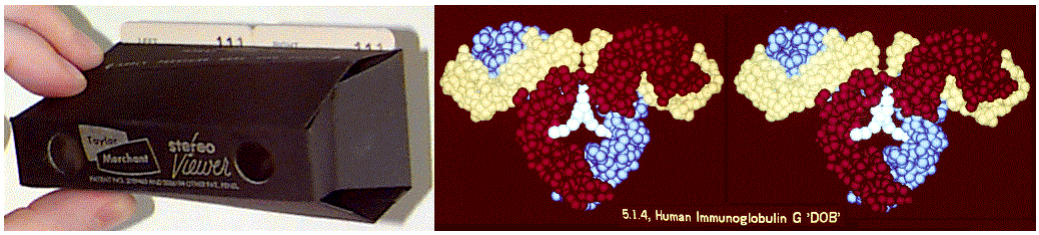


Figure 2.2: The TAMS viewer to the left and one of the slides to the right. (Source: `http://www.umass.edu/microbio/rasmol/history.htm#physical`)

In modern molecular modeling there exists a multitude of stand-alone computer tools such as PyMol [9], VIDA [10] and Chimera [11], which all support several different stereo effects. Lately light web-based systems such as AstexViewer [12], Jmol [13] and MDL Chime [14] have grown in popularity. An extensive list of today's available tools is maintained at Wikipedia [15].

The molecular stereoscopic models created by these tools utilize three main methods:

1. Passive Stereo 3-D
2. Active Stereo 3-D
3. Auto-Stereoscopic 3-D

[1] `http://en.wikipedia.org/wiki/Kickstarter`

Despite aiming for essentially the same stereoscopic effect the underlying theory for these methods vary significantly as well as their implementation, performance and user experience (UX).

2.2.1 Passive Stereo 3-D

A passive stereo effect is applied by wearing different types of polarized glasses. Two common methods are the anaglyph effect[2] and ChromaDepth[3].

The anaglyph 3-D effect is achieved by wearing glasses with lenses made of chromatic opposite colors such as red and cyan. The image is rendered twice, once in each color base, and with a slight shift to create the illusion of stereo. An example of a stereoscopic molecule using the anaglyph method can be seen in Figure 2.3.

The glasses are cheap compared to other methods making them popular when displaying stereo to large audiences such as in cinemas. One downside is that the molecular model loses some of its color properties thus making the analysis increasingly difficult.

ChromaDepth glasses utilizes a holographic film which chromatically distorts the image. The distortion applied by the film assigns different colors different convergence points in the spectacle lens, thus making them appear to differ in depth. Many color schemes exist but the most common one displays red objects as closer to the viewer while blue objects appears as furthest away. The depth is thus achieved by coloring the model in a spectrum from blue to red with a black background to simulate the stereo effect. This approach strips the model of all previous color properties which, as with the anaglyph effect, might hamper the analysis.

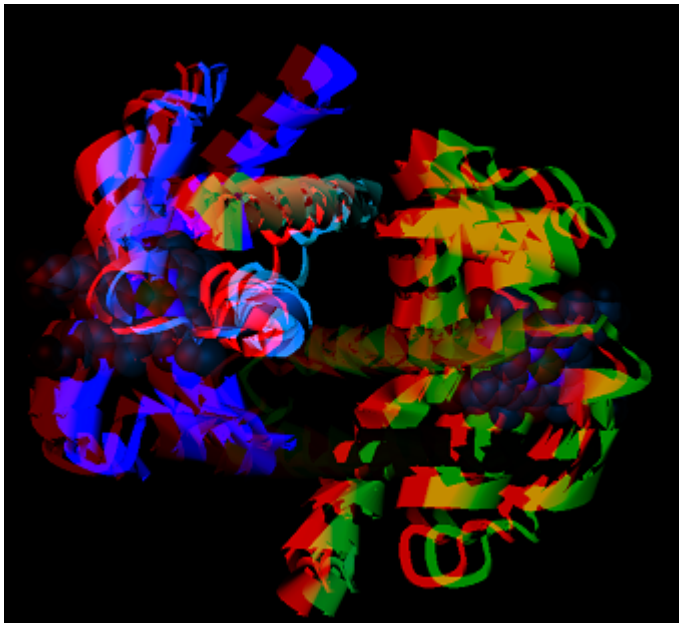


Figure 2.3: Horse deoxyhaemoglobin visualized in anaglyph 3-D.

[2] http://en.wikipedia.org/wiki/Anaglyph_3D
[3] http://en.wikipedia.org/wiki/ChromaDepth

2.2.2 Active Stereo 3-D

Here the stereo effect is achieved using active shutter 3-D systems[4] also known as alternate frame sequencing. The stereoscopic 3-D is rendered by shutter glasses which blocks out the view from the left and right eye in an alternating pattern. The displayed images are also alternated in the same pattern thus displaying separate images for the left and right eye. This method allows the images to be viewed in full size and thereby the full resolution is preserved, which is not the case for methods such as parallax barrier (2.2.3).

One drawback of active stereo is that the shutter glasses used to achieve the stereo effect are by far the most expensive appliance needed by any of the methods. The shutter visualization alternates between the images at clock rates around 100 Hz or more, thus potentially causing lag and rendering issues for large molecular models.

Volbracht et al. showed that the rendering quality of shutter glasses, while outperforming perspective methods, was almost equal to the effect offered by the anaglyph method, which is computationally cheaper to run [16]. Their assessment however focused only on the quality of details of the image and did not take the loss of color data in the anaglyph method into account.

2.2.3 Autostereoscopic 3-D

Autostereoscopic means that the stereo effect is applied without using external glasses or other headgear. The common method to achieve this in molecular modeling is called parallax barrier[5]. A parallax barrier, which can be seen in Figure 2.4, is placed between the viewer and the image source. The barrier consists of parallel lines which blocks part of the viewed scene. The image is rendered in two versions, one for each eye.The rendering is performed in such a way that the barrier blocks the left eye from viewing the right eye image and vice versa. When the barrier and the image are aligned correctly the stereoscopic effect is applied to the image.

While the method is in the same price range as the passive stereo, while preserving color, it performs worse with respect to resolution. Since the two images of the model are displayed simultaneously the resolution is effectively halved. This can lead to issues when attempting to display small or highly detailed structures, which both often are important in molecular modeling.

2.3 Virtual Reality in Molecular Modeling

The introduction of affordable VR development kits such as Oculus Rift has opened up new opportunities for creating stereoscopic 3-D visualizers. Functionalities for reconstructing regular stereo models to VR are also emerging. Many of the common 3-D libraries such as Javascript's three.js[6] now offers support for the Oculus Rift effect. One molecular visualizer that use this method is iview [17] which also offers the anaglyph effect, as well as the parallax barrier. These kind of programs only apply the effect as an overlay on the

[4] http://en.wikipedia.org/wiki/Active_shutter_3D_system
[5] http://en.wikipedia.org/wiki/Parallax_barrier
[6] http://threejs.org/

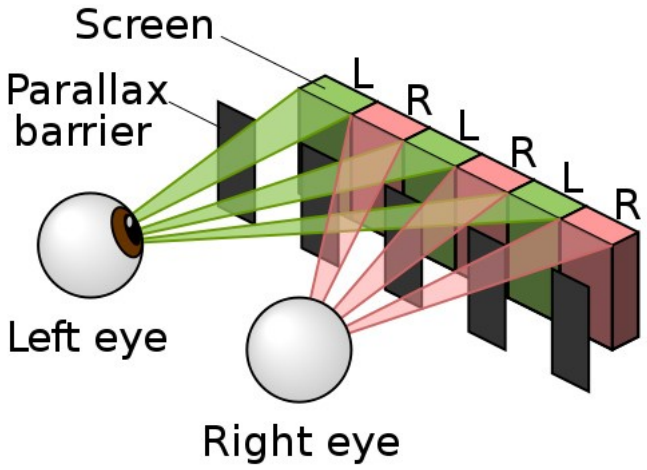


Figure 2.4: A parallax barrier displays a stereoscopic image by separating which part of the screen each eye sees. (Source: `http://www.blogcdn.com/www.engadget.com/media/2011/08/primed-parallax-barrier.jpg`)

regular stereo model without interacting with the Oculus Rift HMD itself. The correct visual effect is created but it looses all of the extra functionalities such as head tracking. This means that turning your head and looking around has no effect and the view has to be controlled by input sources such as a mouse and keyboard or a touchpad.

Many projects in the gaming area make use of Oculus Rift's full capabilities. One of the most common applications that show off the VR experience is roller coaster rides such as Rift Coaster HD[7]. Wearing the HMD grants the user a much more realistic sensation of going up and down the track. Even though the graphics quality reveal that you are in a game, many of the sensations from a real roller coaster ride, such as a surge in the stomach, are present.

Minor prototypes that render molecules using Oculus Rift's software development kit (SDK) and thus use its whole functionality have started to emerge. One example is OculusCrystal [18] which displays different crystal packings in an interactive environment. This project, while supporting the full VR effect, only displayed predefined scenes. We aim to combine the functionality supplied in tools such as iview with the real VR experience achieved in OculusCrystal.

2.4 Computer interaction

Human-computer interaction (HCI) is a field as old as the computer. Historically, typewriters were used as the standard input device. The typewriters were gradually converted into today's keyboards and around the 1960s computer terminals started to evolve. The mouse was introduced as a commercial input device together with the keyboard in the early 80s and combined they have been the standard HCI tool ever since [19]. With the rise of new cost-effective sensors alternative interaction methods have recently started to emerge. Wearing VR-glasses limits the use of many external input devices such as keyboards and

[7] `https://share.oculus.com/app/riftcoaster-hd`

touchpads. This has made alternative interaction methods, such as hand gestures, popular for VR projects.

2.4.1 Gesture recognition

Computer algorithms for hand recognition has previously been implemented using wearable gloves with colored markers [20]. This approach works well but is dependent on the lighting of the room and in some cases the color of your clothes and surroundings. An alternative solution is to use a glove made of piezoresistive sensors [21] to detect the hand movements. This method, while solving the issues of colors, still requires the user to wear gear to interact with the program. The resolution achieved using piezoresistive elements will always be limited by the size of the individual sensor while the maximum resolution for optical methods only is limited by the wavelength of light.

Gesture recognition using Kinect

Kinect is a motion sensing device developed by Microsoft for both the Windows and Xbox platforms. The first release came for Xbox 360 in 2010 and was later followed by a Windows version in 2012. The Kinect is equipped with both RGB color and infrared (IR) sensors which allows it to capture gestures and a multi-array microphone for voice interactions. Although it was first marketed for gaming, its competitive price and ease of access has lead to a spread into scientific fields. As an example Alnowami et al. showed that the Kinect sensors successfully could be used for gait tracking, an important indicator of health [22]. Another life science application is an exercise game for people suffering from Parkinson's disease developed by Galna et al. [23]. They construct stepping and reaching tasks for the users and track their progress using the Kinect. Instant user feedback enables the patients to learn the tasks, thus training and improving balance in their own homes.

Previously developed hand recognition projects using the Windows Kinect v1 showed promising results using per pixel algorithms. Even with the lower resolution (640x480 color, 320x240 IR) in Kinect v1, robust algorithms for over 14 distinct gestures could be defined utilizing the IR and color data [24]. The algorithms apply nearest neighbour matches[8] against a database of prerecorded gesture. The improved resolution (1920x1080 color, 512x424 IR) of the new Kinect v2 increases the detection capabilities of such algorithms. The computational load also increases with additional pixels which requires higher performance demands on the computer. The gesture recognition must be detailed enough to correctly interpret the gestures while still being fast enough to handle the data (30 FPS) in real time.

2.4.2 User Experience

UX emerged from the field of usability engineering. The term was brought to wide recognition by Norman et al. in their 1995 publication on human interface research [25]. The definition of the term user experience is still under some debate, but the general consensus

[8] `http://en.wikipedia.org/wiki/Nearest_neighbor_search`

is that it combines the usability aspect with the users feelings and relationship to a product. The goal is to extract the users point of view and not just focus on performance and response times.

One reason for the failure of VR in the 90s, described in Section 2.1, was most probably due to customers not being fully satisfied with the experience. Another example of why we need to evaluate the user experience and not just the user interface (UI) was made by Jakob Nielsen and Don Norman [26]:

"Consider a website with movie reviews. Even if the UI for finding a film is perfect, the UX will be poor for a user who wants information about a small independent release if the underlying database only contains movies from the major studios."

The UX evaluation of a product can be performed in many ways but all of them include user input. Some of the more common methods for gathering the UX data include questionnaires and focus groups [27].

Chapter 3
Technical Background

This chapter explores the methods used to create a VR effect in Oculus Rift and the techniques behind Kinect's body tracking and depth calculations.

3.1 Virtual Reality with Oculus Rift

Oculus Rift creates a stereoscopic 3-D using two lenses and a LCD display. The display applies a barrel distortion to the image which causes it to appear bulgy and spherical in the center. When viewed through the lenses the eyes are shown different parts of the screen, thus creating the 3-D effect. The lenses also apply a pincushion distortion thus reversing the distortion applied by the screen. Examples of pincushion and barrel distortion can be seen in Figure 3.1.

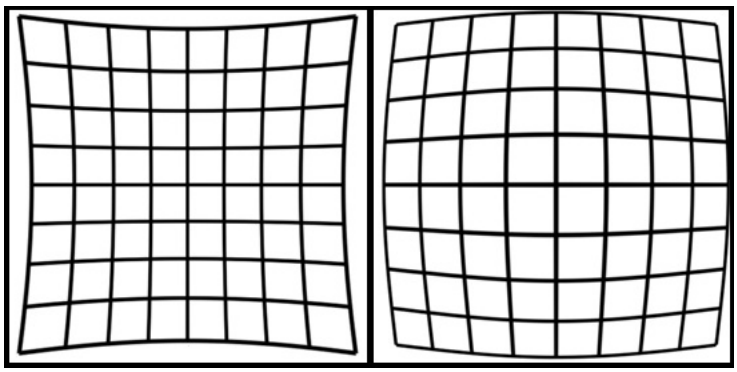


Figure 3.1: Pincushion distortion (left) and barrel distortion(right).

The VR experience is enhanced by displaying the scene with over a 90 degree field of view (FOV). The Oculus Rift HMD also comes with IR lights that communicates with an

external positional tracker, usually placed on the computer screen. The positional tracker combined with 3-axis accelerometers gives the Oculus Rift the direction of users head movements. The displayed images can thus be translated according to the direction a user is looking and thereby creating the VR experience.

3.2 Gesture recognition

Hand gestures have been studied extensively, especially in the field of sign language. For example Stokoe [28] divides hand gestures into four components: hand shape, location, orientation and movement. By accounting for all four factors when interpreting the gestures we can allow users to have a natural and intuitive interaction with our program.

3.2.1 Gesture recognition with Kinect

The Kinect SDK defines gestures using 24 points on the body called joints [29]. These joints are calculated from single depth images using a random forest algorithm as described by Shotton *et al.* [30]. In short, the algorithm maps the joints to an image frame using two components, depth filtering and a database of images of people with the joints correctly tagged.

Depth can be calculated using stereo triangulation[1] from two sensors as seen in Figure 3.2.

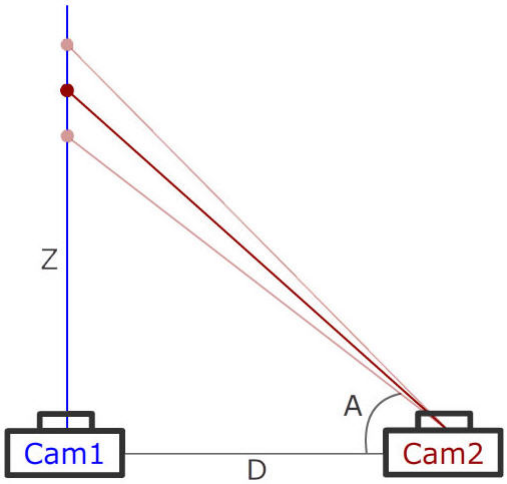


Figure 3.2: The depth of a point is calculated using the angular difference of that point between by two sensors. (Source: `http://static1.1.sqspcdn.com/static/f/513908/10402744/1295995364893/Stereo_Triangulation_Explained.jpg?token=xCQHygG1ZmHP2e%2BlUBNjJOtXGFA%3D`)

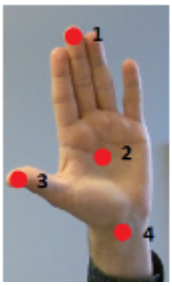
The stereo calculation requires the two sensors to correctly map all points to the other

[1] `http://en.wikipedia.org/wiki/Triangulation_%28computer_vision%29`

sensors points, this is known as the correspondence problem[2]. The Kinect solves this issue by combining the information from two IR sources, one IR-sensor and one IR emitter. The IR emitter covers the scene in a pattern of IR dots that mark a grid of unique boxes over its entire FOV. The dots are recorded by the sensor and the pattern allows it to calculate the dots location in the emitter's FOV, thus solving the correspondence problem. The approach using IR mapping allows the Kinect to perform as intended even in dim light and irrespective of two objects having the same color. One drawback is that IR light from other sources will destroy the pattern, thus an IR emitter from a second Kinect will cause the depth measurement to fail.

The generated depth map is compared to a database of pre-tagged images and matched using a random forest decision tree. This means that the algorithm attempts to place the 24 joints on the location that most resembles one of the predefined positions. The algorithm assigns each joint one of three tracking states "tracked", "inferred" and "not tracked". If the algorithm successfully identifies a position for the joint using the predefined positions the joint state is set to "tracked" [31]. If a joint position cannot be identified the algorithm tries to evaluate the position using neighboring joints, for instance the right wrist joint should always be close to the right palm joint. If neighbouring joints are tracked the algorithm guesses the joint position and sets its state to "inferred". If no position can be found the algorithm assumes that the body part is outside sensor range and sets the joint tracking state to "not tracked".

In this project we focus largely on hand gestures thus mainly utilizing the eight hand joints (four in each hand). The joints mapped to the right hand and their corresponding positions are shown in Figure 3.3.



1: HandTipRight

2: HandRight

3: ThumbRight

4: WristRight

Figure 3.3: The four hand joints from Kinect's SDK.

Developing Gestures with Kinect

There are two main methods available when developing a new gesture for Kinect, heuristic[3] and machine learning[4] [32]. In the heuristic approach the defining rules and characteristics of a gesture, such as inter-joint relations and gesture velocity, is specified manually in the code. When using machine learning users are recorded while performing the gesture and when doing other gestures. The recorded frames are then manually tagged with true or false, representing if the gesture is correctly presented or not. The information in the

[2] http://en.wikipedia.org/wiki/Correspondence_problem
[3] http://en.wikipedia.org/wiki/Heuristic_%28computer_science%29
[4] http://en.wikipedia.org/wiki/Machine_learning

tagged frames, such as joint placement, generates a model that can be used to evaluate future frames.

Chapter 4
Implementation plan

The implementation plan describes the process and tools used during development of MolecularRift as well as the requirements for building and running the project.

4.1 Development Process

Developing software for new technologies is not straight forward. The implementation is often unpredictable and requirements tend to change during the process [33, 34]. To handle these challenges we used an iterative development process where each iteration cycle consisted of design and development followed by a user test evaluation.

4.1.1 Investigation Phase

Since we were breaking new ground in terms of molecular VR it was decided to start the project with an investigation phase. The end goal of the investigation phase was to identify appropriate methods for developing the visualizer and gesture interpreter. These methods would then serve as a starting point for the following iterative development process.

Parnas and Clements have shown that many issues in the design will only become apparent during implementation [34]. Proof of concept prototypes will be implemented in the investigation phase to determine that the design is satisfiable. Identifying design issues early is critical since the time and cost for fixing them at a later stage increase with the progression of the project [35].

4.1.2 Gesture Interaction

The design of human-computer gesture interactions consist of two key principles, creating intuitive gestures and making the gesture interpretable by the computer's recognition

system [36]. These two principles do not always align and the design process will require a trade-off between the two. Since the computer interpretation limits the freedom to design gesture we must therefore first explore how well the sensor can discriminate between different hand poses. The investigation phase aims to evaluate different recognition techniques to identify the method that provides the best support for designing intuitive and natural gestures.

4.1.3 User Tests

The user tests, ending each development step, included potential end users at AstraZeneca. The tests were designed to measure the UX with emphasis on usability. The main goal when including UX in the project was to bridge issues that might arise from users being unfamiliar with the new gesture based interactions and the VR environment. By incorporating the end users in each iteration of the development the UX can be improved for each step thus hopefully creating a tool that is enjoyable as well as usable.

Research has shown that usability data is best gathered when a user is given the chance to explore the product with minimal interference from moderators [37]. User experience companies such as the Nielsen Norman Group thus recommend self-report methods such as questionnaires for usability studies [27].

Our UX evaluation consisted of observing the test pilots while they used MolecularRift to perform predefined tasks. The testers were also given the opportunity to share their experience by answering questionnaires after each session.

The UX of a product may be considered more difficult to assess than hard numbers such as response times and frame rate. It can therefore be difficult to evaluate the results of a usability test, especially if the reviews are mixed. Here the iterative process becomes the key to measure improvements in the product. While the UX evaluation of one product can be hard to quantify into interpretable data, the comparison between two similar products is easier to analyse [38]. When comparing the new version to the version from the previous iteration, one of the three cases below can occur:

1. The new version outperforms the old.

2. The old version outperforms the new.

3. The two versions perform equally.

If a feature in the new version tests better, the change is kept, if not it can be changed back or developed further. In either case the path forward is clearer compared to evaluation test results from only one version.

Small test groups are satisfactory for determining the most apparent usability issues while larger groups are required to achieve a wide coverage of less obvious complications [38]. Since our product was developed from scratch we started with test groups of less than ten people in total, and dealt with the most significant issues each iteration. Larger test groups were more relevant during later development stages, when the tool was more mature [38].

4.2 Development Tools

We used the Oculus Rift Developer Kit 2 (DK2) with the Oculus Rift 0.4.4 SDK and Windows Runtime [39]. The Kinect for Windows SDK 2.0 [40] and its application programming interface (API) was utilized to perform the gesture recognition. All development was done in Windows 8.1 with the game engine Unity 5 Personal Edition [41] and MonoDevelop as the primary editor. The code was mainly written in C# combined with a few C++ scripts. A full list of the employed tools and environment can be found in Appendix A.

4.3 Setup

The Kinect Sensor and the corresponding Oculus Position Tracker needs specific placement to work optimally. Microsoft states that the Kinect v2 correctly tracks humans at distances between 0.5 and 4.5 meters with its best performance in the 0.8 to 3.5 meter range [42]. The sensor has 70 degree horizontal and 60 degree vertical angle of vision. We decided to place the Kinect sensor above the computer screen at a height of approximately 1.8 meters and with a slight downwards tilt. This placement allowed the tracker to monitor the interacting users from the waist up when standing in the 0.8 to 3.5 meter range. By positioning the Kinect at the center of the screen we also made sure that users placed straight in front of the screen would be optimally tracked.

The Oculus positional tracker receives IR-signals from the HMD and thus need unobscured vision of it. We decided to place it at the top of our computer screen, the position of both sensors can be seen in Figure 4.1. The view from the sensor with a user at 1.2 meters distance can be seen in Figure 4.2.

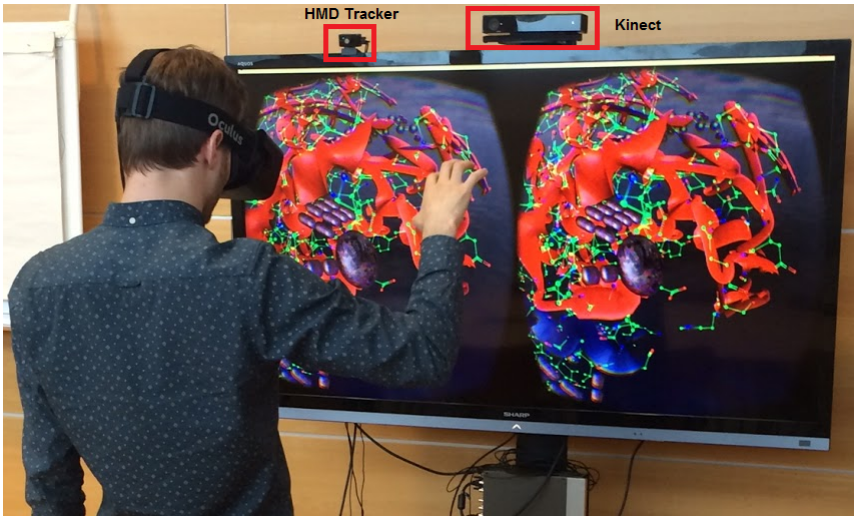


Figure 4.1: The sensor placement used during MolecularRift development.

Figure 4.2: A depth map created by the Kinect, illustrating the sensor perspective of the setup.

4.4 Minimum Requirements

While the tools and hardware described in Section 4.2 were used to create the project it will also successfully run on other systems. This section aims to describe the minimum requirements for running this project. The complete list of minimum requirements can be seen below:

- Microsoft Kinect v2 for Windows
- Oculus Rift Head Mount Display DK 2
- Oculus Positional Tracker DK 2
- PC with:
  - Windows 8 or 8.1 (x64)
  - USB 3.0 x1
  - USB 2.0 or higher x2 (at least one powered)
  - DVI-D or HDMI graphics output
  - DX11 capable graphics adapter
  - 4 GB Memory

Chapter 5
Investigation of the SDK:s

In this chapter we explored the capabilities of both the Oculus and the Kinect SDK's. Gesture recognition algorithms were developed using the Kinect SDK's API and Visual Studio Express 2013 as an editor. Proof of concept programs were developed in C# and C++ to test the performance of these algorithms and to investigate methods for combining the two SDK's into one program. The results were evaluated and the conclusions used as guidelines in the subsequent development of the first version of our tool, MolecularRift.

5.1 Gesture Development using Kinect

The Kinect comes with both color and IR cameras and an array of microphones, as can be seen in Figure 5.1. The SDK offers many pre-built programs that communicates with these sensors. For example, Depth Basics-WPF creates a depth map of the view and Audio Basics-WPF which allows the user to create voice commands from strings.

We focused on exploring two interaction methods, gesture recognition and voice commands. The goal was to create gestures for interactions such as movement and rotation of molecules while voice commands would be used for launching the menu and toggling the gesture recognition on and off.

5.1.1 Hand tracking

The first step when interpreting hand gestures is to successfully track the hand. This can be done using pixel algorithms on frames obtained from both the color and the IR sensors. We found that analysing both sources simultaneously caused performance issues, due to the large amounts of data to perform pixel analysis on. As a consequence, only use one sensor was used. It was decided that the IR sensor would suit this project best since the color data was sensitive to the light of the room and the colors of nearby objects. The

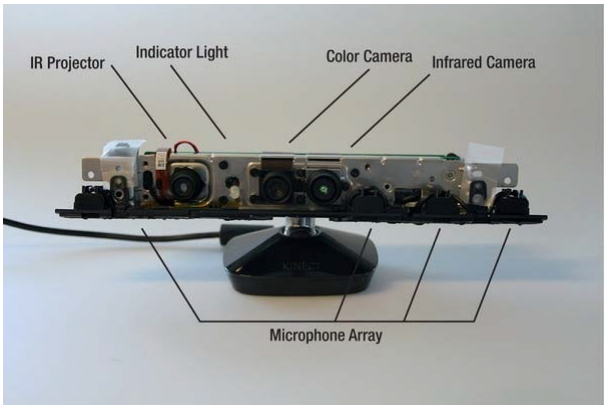


Figure 5.1: The sensors contained in the Kinect (Source: `http://fizzible.com/wp-content/uploads/2014/11/kinect1.jpg`).

IR sensor also had the advantage of being able to calculate depth which was utilized for filtering background pixels in the IR frames.

The hand was located using Kinect's native functions in Body Basics-WPF for mapping the palm joint. The palm location was combined with the depth map created from Depth Basic-WPF and was thus used to extract the distance between the hand and the sensor. A rectangle with the palm joint at its center was defined in an attempt to limit the amount of pixels that had to be search in order to locate the hand. The depth of each point contained in the rectangle was extracted and compared to the depth of the palm. A bitmap the size of our depth frame (512 x 424 pixels) was created in an attempt to display the output. A representation of the hand was achieved by coloring pixels in the bitmap green if their corresponding depth point had a value similar to our palm depth. An example of the representation can be seen in Figure 5.2. To allow interpretation of gestures such as finger pointing it was decided to add additional colors to represent if a pixel was closer or further from the sensor compared to the reference depth of the palm. Red was chosen to represent pixels closer to the sensor while blue was used for those further away.

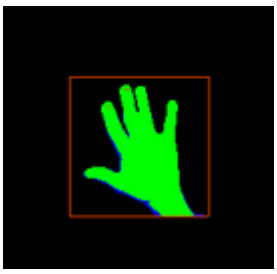


Figure 5.2: The tracked hand and the searched rectangle.

The algorithm successfully tracked the hand and required low computational resources since the number of pixel traversings could be limited to approximately 1% of the original. One downside was that all objects close to the hand and at a similar depth also were selected. For instance if a user scratched his head the whole head was also interpreted as part of the hand. We focused on hand gestures alone, thus such cases would not arise while performing the intended interactions. Since the focus of this part was finger tracking, these

issues were ignored.

5.1.2 Finger tracking

The next step was locating and identifying single fingers, which would allow us to define a larger number of unique hand gestures. Optical finger tracking has previously been performed through nearest neighbor algorithm against predefined hand shapes [24] and with gesture models built through machine learning [32]. While both methods can achieve robust recognition they both depend on databases with prerecorded gestures. Redesigning gestures can therefore be cumbersome since new training data has to be produced. We needed a more flexible method since the iterative development process constantly evaluates and potentially redesign the gestures. We therefore developed a per-pixel algorithm to define gestures through heuristic rules.

In an attempt to limit the number of traversed pixels the direction of the hand was located, thus allowing the algorithm to guess where the fingers would be positioned. Once again Kinect's predefined joints were utilized. This time the palm joint was combined with the wrist joint. An approximate hand direction was created by drawing a straight line from the wrist to the palm joint.

Another search rectangle was created by drawing a new line from the tip of the direction vector (at the palm joint position) in a perpendicular direction. The depth pixels from each position on the new line were traversed in the direction defined by our direction line. When the depth difference exceeded a predefined cutoff, e.g. the pixel did not belong to the hand, the traversing was stopped and restarted at the next pixel on the line, thus creating a outline of the tracked hand. In an attempt to create the search rectangle containing the fingers another line was drawn at the point furthest from the start line. This box and the direction line can be seen in Figure 5.3.

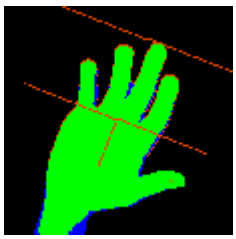


Figure 5.3: The direction line and two parallel lines defining the search area.

The approach was quickly found to be flawed since it would stop each time an edge was reached, thus failing to identify the whole finger if it was bent. The issue was resolved by reversing the transversing direction. The starting position was instead chosen at a distance far enough from the palm to never miss a finger, thus always finding the furthest tip at the cost of a larger search area. A comparison of the methods can be seen in Figure 5.4.

The first attempt to define a hand gesture was to evaluate if the hand was closed. By utilized the average distance of all edge points we could evaluate if any fingers were raised. The average distance for both an open and a closed hand can be seen in Figure 5.5. As can be seen the difference between the furthest tip and the average point shifts significantly

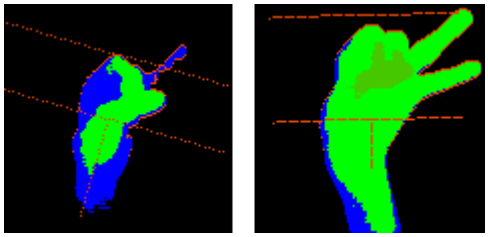


Figure 5.4: The palm out method (left) fails to find fingers at large angles while the reverse direction (right) succeeds.

depending on the number of raised fingers. By calculating the ratio between the two points the state of the hand (open or closed) could be calculated.

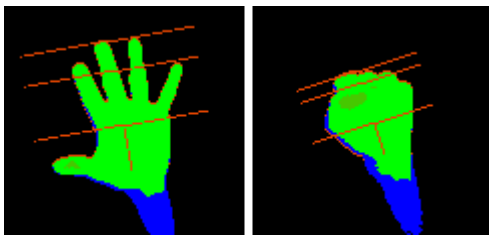


Figure 5.5: The average edge distance for an open (left) and a closed (right) hand, represented by the middle line.

Having established a method that could differentiate between an open and a closed hand the next step was to develop a method that could interpret gestures performed by an open hand. By identifying the finger tips, information was obtained regarding the number of raised fingers.

For this analysis the assumption was made that the tip of a finger in an open hand would be further away from the hand then the rest of the finger. It was also assumed that such a finger tip would exceed a certain ratio of the average edge distance. This definition allowed us to define the first finger tip, previously identified by the algorithm, as the furthest edge point of the hand. The location of the tip was marked by rendering a red circle, in our hand representation, with the finger tip point in its center, thus allowing us to visually evaluate the results. It was attempted to identify a second finger by removing the furthest point from our set of edge points. The second furthest point could be identified, thus potentially giving us the location of another finger. Since that point possibly could belong to the previously found finger we chose to not only discard the top value but also points in close proximity to the already defined tip. This decreased the chance of defining two tips on one finger. The algorithm was designed to search for additional fingers as long as edge points with a distance above average existed or four fingers had been found (the thumb was not included in the search rectangle due to its angle).

This approach worked well when all fingers were straight but bent fingers could still be marked twice as seen in Figure 5.6. To amend this we extended the deletion of edge points to also include all points between a defined finger tip and the middle point (the palm position). This approach relied on the algorithms ability to identify the fingers in a specific order (middle finger > middle & ring fingers > pinkie) since for example finding the pinkie first would erase the points for the index finger. The resulting finger tip mapping, which be

seen in Figure 5.7, correctly handled some of the previously double marked finger issues but not all.

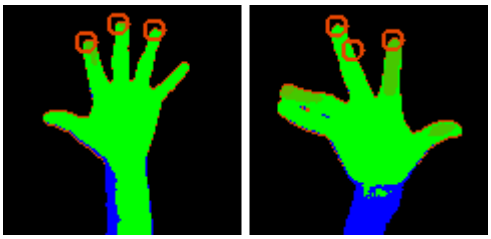


Figure 5.6: Attempts to identify finger tips using the furthest tip method.

The results prompted us to revisit the previous version of the algorithm, where only the nearby edge points were deleted if a finger tip was identified, with the modification of an increased deletion area. This removed most of the cases where fingers were marked twice but instead occasionally caused two fingers close together to be counted as one. Since the method performed well for frequently used gestures such as pointing and the victory sign[1] it was selected as the best of the approaches.

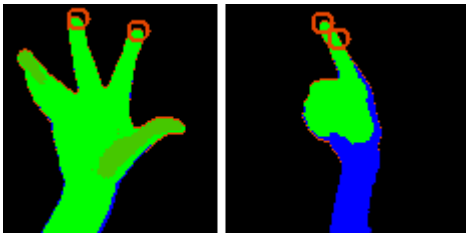


Figure 5.7: The deletion of edge points solves previous double mappings (left) while still incorrectly labeling fingers close to the middle point (right).

Some issues still remained with tracking the pinkie finger. As can be seen in Figure 5.5 the average distance could be greater than the size of the pinkie if all fingers were stretched. This was caused partly due to its shorter length but mainly because it often had a different angle than the other fingers. It could therefore be compared more favorably by calculating the Euclidean distance between the edge points and the palm, thus causing bent fingers to contribute equally to the average distance. This approach was not adequate and cases still occurred where the pinkie was to short to be identified as a raised finger. To amend this the points from a identified finger were removed and the average distance was recalculated. This lowered the limit for subsequent fingers to qualify as raised while keeping the limits for a closed hand intact.

5.1.3 Thumb tracking

The thumb was tracked separately partly due to its independent location and direction but also because its role in gestures often differ from the other fingers. Many of our natural

[1] `http://en.wikipedia.org/wiki/V_sign`

gestures such as the "A-ok sign"[2] use a combination of the thumb and the other fingers. Using a different algorithm for the thumb would therefore allow the program to differentiate between the thumb and the other fingers. This allowed us to define gestures where they are combined.

The thumb tracking was performed using the furthest tip and average distance methods described previously (5.1.2) but with a search rectangle with an orthogonal angle to the hand direction, facing inward towards the body. The decision to only search in the inward direction would result in only locating the thumb if the palm was facing the camera. If the hand was rotated 180 degrees the algorithm would still search the inward side and in some cases incorrectly identify the pinkie as the thumb. Since most gestures are performed with the palm facing the direction of the intended interaction target the method was kept. The algorithm was more stable and faced few of the previous issues with double marked fingers since the algorithm terminated after identifying its first finger tip.

5.1.4 Proof of concepts

To test the robustness of the hand tracking algorithms and to evaluate if it would be feasible to use them in the development, a smaller prototype was built as a proof of concept. The program, called HandMap, was designed to visualize the tracked hands on screen, as well as allowing the user to control cursor movements and cursor clicks using gestures defined by the hand tracking algorithm.

Another program, called MyRemote, was also developed using Kinect's own predefined gesture and voice recognition to remotely control existing molecular 3-D viewers. The aim of this program was to evaluate if the existing API would suffice for developing our tool and to give us a benchmark to evaluate our own algorithm against.

The source code for HandMap and MyRemote can be downloaded from GitHub. The URL-link to each program and a description of their associated SDK packages can be found in Appendix B.

HandMap

The aim was to combine the position data from Body Basics-WPF with depth data from the IR-sensor. To accomplish this we added functions from Depth Basics-WPF as well. The hand position was located, once again using Body Basics-WPF and our previously developed hand tracking algorithm. Functions for extracting and displaying color images were added from Color Basics-WPF. This allowed the created hand representation to be shown next to a color image of the real scene, see Figure 5.8.

Finger tip detection was applied using the method described in Section 5.1.2. The finger tip detection allowed us to count the number of fingers and thereby enabling some simple gestures to be defined.

In an attempt to give users visual feedback of the algorithm output the finger tips were marked with circles, blue for the thumb and red for the rest of the fingers, and the trajectories between the finger tips and the palm location were drawn to represent the fingers. An example of the visual feedback with finger tip marks and finger trajectories can be seen in Figure 5.9

[2] http://en.wikipedia.org/wiki/A-ok

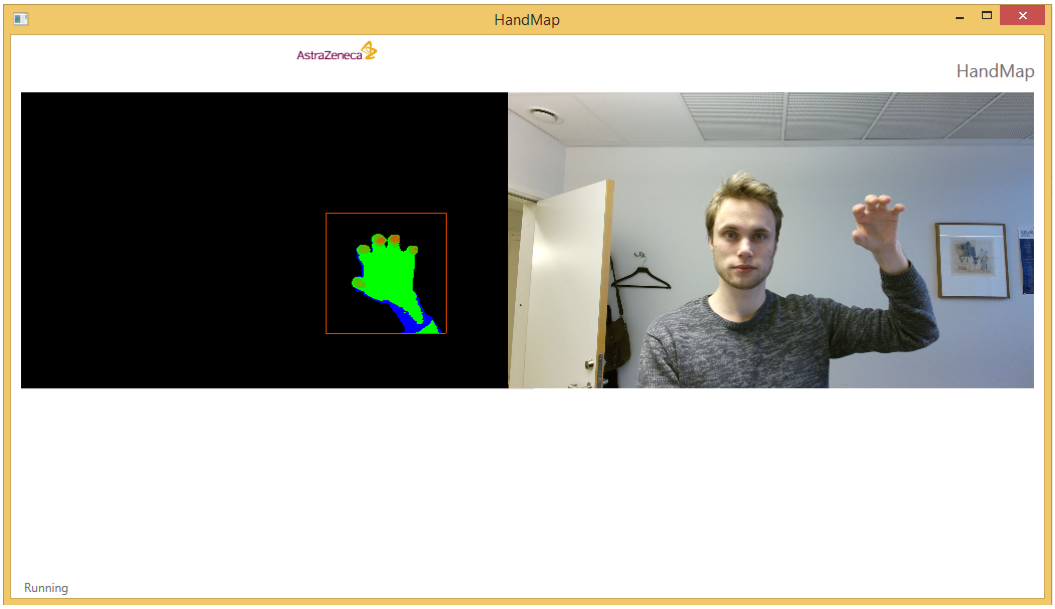


Figure 5.8: Hand tracking using depth filtering.

Gestures for steering cursor movement and perform cursor clicks were defined. A pointing gesture, which would allow the user to control cursor movements by moving one finger up and down, was developed using the finger count. The gestures were defined as a hand where only one finger was raised, not counting the thumb. By monitoring the location of the raised fingers tip for each frame and comparing the new coordinates with the previous, its movement could be calculated. The direction of the movement was supplied as input to the on-screen cursor arrow using functions from Visual Studio's interop assemblies[3], thus allowing the user to move it with a pointed finger.

A gesture was developed to allow users to trigger cursor clicks. In an attempt to replicate the movement performed when pushing a button the gesture was defined as a single pointed finger quickly doing a tap motion in the air. This gesture was implemented using the depth value of the finger tip. If the tip of the raised finger was moving towards the sensor (Z-axis) while not moving in the X and Y directions it was defined as a tap. To reduce unintended taps the gesture was required to be performed during a minimum number of frames, thus not triggering a cursor click for small movements.

MyRemote

This program was developed in an attempt to control three existing molecular visualization systems utilizing the Kinect sensors as our only input device. The selected systems were PyMOL [9], GLmol [43] and C-Lab [44], all commonly used at AstraZeneca. We utilized two of Kinects SDK packages, Body Basics-WPF and Audio Basics-WPF. Body Basics-WPF tracks body positions and supplies functions for three predefined hand gestures seen in Figure 5.10. Audio Basics-WPF contains a library for speech recognition and simple methods for building grammar and adding voice commands.

The speech recognition used the standard speech.dll in Windows 8.1 Enterprise and the

[3] `https://msdn.microsoft.com/en-us/library/bb164686.aspx`

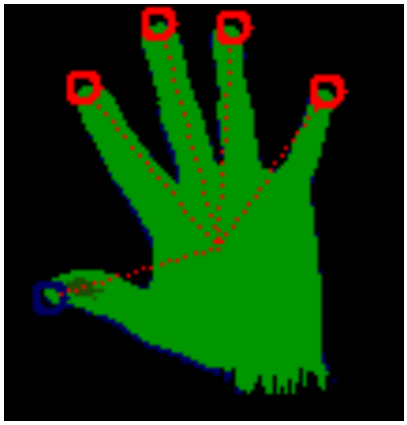


Figure 5.9: Finger tip detection using depth filtering.

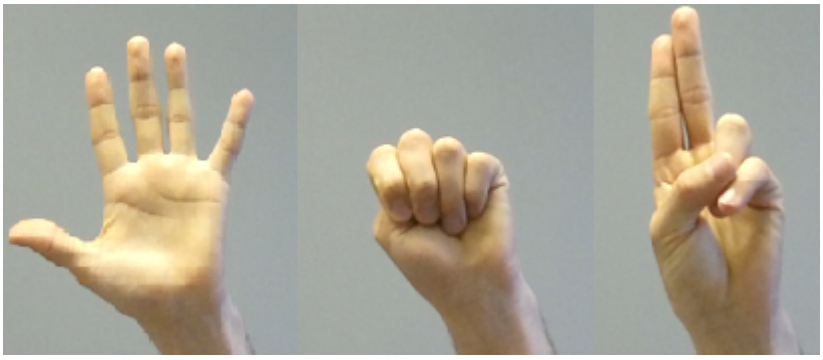


Figure 5.10: The three predefined gestures in the Kinect SDK: Open (left), Closed (middle) and Lasso (right).

US-en language pack to interpret the sound input. The voice interaction was intended to supplement hand gestures when, for instance, launching a program or toggling the remote control on and off.

The users needed to control both cursor and keyboard inputs in order to interact with the visualizers, thus requiring more than three distinct gestures. The gestures, defined in Body Basics-WPF, could be performed with each hand independently, thus allowing gestures to be interpreted differently for the left and the right hand. It was decided not to use the open hand gesture since it was considered the natural state of the hand, thus allowing the user have open hands without causing unintended interaction. A complete list of the gestures and voice commands can be seen i Table 5.1 below:

5.1.5 Intended Interaction

A challenge when working with HCI is to define accurate boundaries to evaluate if the input was an intended interaction. This issue was encountered during the development of MyRemote (Section 5.1.4). The goal was to steer the on-screen cursor by mapping it to the users hand movement. This generally worked well, but when the user attempted to end the interaction problems occurred. The natural reaction when finishing the interaction was to simply lower the hand, often quite rapidly. This caused the cursor, which followed the hand movements, to rush to the bottom of the screen, which was not the users intention.

Table 5.1: Fetures MyRemote

Gesture	**Action**
Closed Right	Cursor Movement
Lasso Right	Zoom
Lasso Left	Cursor Click
Voice Command	**Action**
Activate	Activate Gesture Recognition
Break	Deactivate Gesture Recognition
Open Graphics	Launch GLmol
Calculated Lab	Launch C-Lab
Python Mol	Launch PyMOL
Shut Down	Shut Down
Are You Alive?	Status Check
Help Me	Print Commands to Console

This issue relates strongly to Stokoe's location component, which states that humans interpret hand gestures differently depending on the hands location in relation to the rest of the body [28]. For instance a raised open hand would by many be interpreted as a greeting. An open hand positioned beneath the waist would, contrary to the raised hand, not be seen as any gesture at all. To improve our HCI such conditions was added to the interpretation model, thus allowing the user to have a more relaxed and natural interaction.

As a remedy, it was decided to use an approach called "engaging" in the Kinect community. An engaging gesture allows a user to signal the start of the HCI. Many different engaging gestures have been tested, such as the 'Wave to engage' used in Kinect for XBox One. A user study by Schwarz et al. found that the fastest and most accurate of the methods was to simply raise a opened hand towards the sensors [45].

An implementation of a method for signaling the start and end of HCI was created using the palm and wrist joints available in Kinect's SDK. A raised hand, intending to interact, has its palm joint located above its wrist joint. By extracting the height coordinates of both joints all interactions where the hand was not raised could be filtered out. This method solved previous interactions issues and allowed users to lower their hands without being interpreted as a cursor movement.

5.2 Combining Kinect and Oculus Rift

In the first part of the investigation phase we familiarised ourselves with the comprehensive SDK of Kinect. The Oculus Rift SDK, on the other hand, was much less extensive but contained three demo scenes which were assembled in C++. It contained a basic example on how to generate in-game objects from code and how the HMD input was handled. The possibilities for dynamic generation of molecular representations using the SDK were explored together with ways for combining the Oculus Rift SDK with the SDK of the Kinect. As a final proof of concept we chose to develop a program which would render a simple molecule in VR and let the user interact with it using hand gestures interpreted by

the Kinect.

The developed program, called MolyRift, can be downloaded from GitHub. The link and the SDK packages used can be found in Appendix B.

5.2.1 MolyRift

This program combines the SDK:s for Kinect and Oculus Rift into a shared application. The program reads x,y,z-coordinates from a text file and uses them to render atoms to the scene and lets the user interact with the atoms using hand gestures. It was built using functions from OculusTinyRoom demo in the Oculus Rift SDK, which was written in C++. Functionalities from Kinect's Body Basics-D2D, which is the C++ version of the previously used Body Basics-WPF (C#), were utilized to define and interpret gestures.

A screenshot from MolyRift where both the left and right eye image is shown can be seen in Figure 5.11.

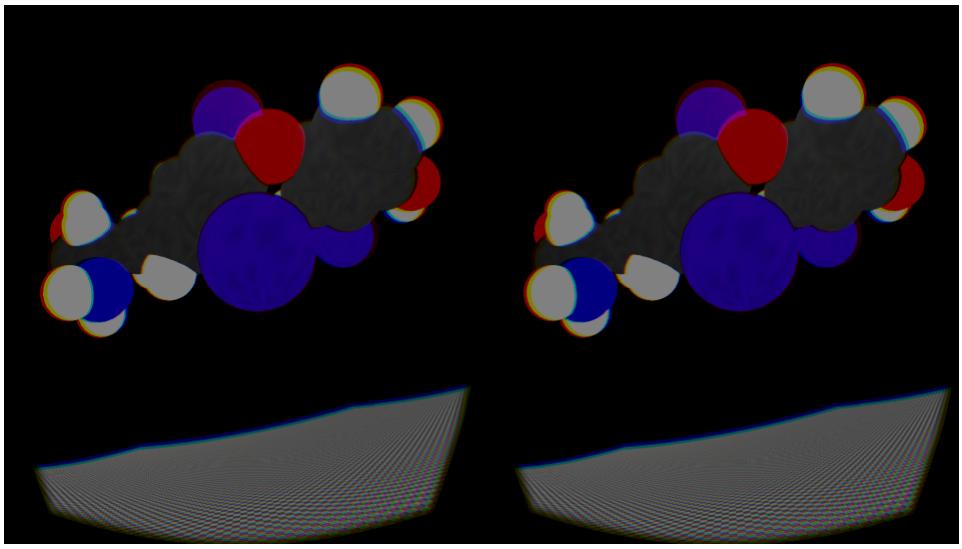


Figure 5.11: The view in MolyRift as seen on the computer screen.

5.3 Evaluation

This section aims to evaluate the developed algorithms, the explored SDK-packages and the proof of concept tools created during the investigation phase, thus creating a platform for the development of the first version of the main tool - MolecularRift.

5.3.1 Gesture Interaction

Practice was needed for the predefined gestures from Kinect's SDK. For example the Lasso gesture was interpreted as Open if the thumb could be detected. However, after practicing the gestures the performance showed to be robust.

In our view the developed hand and finger tracking algorithms performed on par with Kinect's when the variables such as furthest tip to average distance ratios were tuned for the user. Issues occurred when switching between users with different hand proportions and the cut-off values, such as the ratio exceeding the average distance needed to qualify a finger as raised, had to be updated. This could potentially be solved by allowing user profiles with different configurations but since Kinect's gestures performed well we saw no reason to develop this algorithm further. The gesture recognition of the Kinect, while robust in the gesture detection, limits the scope of gestures to those that can be differentiated through the joint points. This reduces the design options but provided gesture detection based on training sets with sizes far greater then what could be collected in the scope of this project.

The method for determining intended interactions (Section 5.1.5) improved both Kinect's own gesture interaction and the developed finger tracking algorithm. The method allowed users to have a more relaxed interaction where the HCI could be signaled on and off. The first version of MolecularRift was therefore built using Kinect's pre-defined gestures with the addition of the developed method for identifying intended interactions.

5.3.2 Voice recognition

Voice interaction was explored in the development of MyRemote (Section 5.1.4). The voice commands used included software names such as GLmol and C-Lab. Since these words did not exist in the US-language dictionary, used for the speech recognition, they were specified phonetically to be interpreted properly. For example C-Lab was natively interpreted as being pronounced 'klab' and had to be changed into two real words as 'see Lab', which solved the issue.

The voice interaction worked well in a quite room but it was easily disturbed by other sounds, such as conversations in adjacent rooms. Since the voice recognition attempts to map all sounds to a defined list of words it quickly became apparent that short words were prone to be registered as false positives[4]. For example the 'see lab' voice command was frequently appearing as a false positive. This was due to the word 'see' which often was present in regular conversation. As a workaround, voice commands that were less frequently occurring were selected.

Using short words also caused issues for the program. For example the words 'stop' and 'help', which were used as commands in early development, both had a large emphasis on the ending 'p' when pronounced. When using confidence cut-offs around 70 - 80 % (default 30%) such words would match both voice commands, thus always mapping to which ever was on top of the list. Increasing the confidence would instead cause false negatives where correct voice commands were not recognised. The voice commands were therefore changed to 'Shut Down' and 'Help Me', thus distinguishing between the two with confidence cut-off's low enough for efficient speech recognition.

The Kinect has the ability to determine the direction of a sound using its microphone array. By comparing the sound registered in each microphone the Kinect evaluates which microphone position that is closest to the source. The direction of a sound source combined with positional data from the body tracking can be used to filter which sounds that should be interpreted. While possibly solving some of the issues mentioned above, the voice

[4] `http://en.wikipedia.org/wiki/False_positives_and_false_negatives`

recognition is currently not mature enough to meet our quality criteria. It was therefore decided not to use voice interaction in the first version of the program.

5.3.3 Oculus Rift

The Oculus SDK used in MolyRift (5.2.1) showed that integration of the Kinect SDK was possible but creating and rendering molecules had to be done from scratch. It was expected that molecules in real-life project would be much larger and rendering them in real time, through script, would require implementation of many basic functions, such as drawing standard geometrical objects. The quality of the graphics was not satisfiable when compared to other Oculus Rift programs such as Tuscany, seen in Figure 5.12, which was included in the Oculus Rift SDK. The first version of the tool was therefore developed using the Unity game engine, which has an extensive API for drawing graphics. Unity also had a associated Oculus Rift integration plug-in which allowed for communication with the HMD through C# scripts.

Figure 5.12: An example view from the Oculus Tuscany Demo (Source: `https://share.oculus.com/app/oculus-tuscany-demo`)

Motion Sickness

Oculus VR is currently working on improving the motion sickness experienced by users. The phenomenon is common enough to have spawned its own term, virtual reality sickness[5]. Even short sessions with the HMD could, in our own experience, cause quite a bit of nausea. The cause for virtual reality sickness is not yet fully understood but the refresh rates of the display could be one component. Our experience while developing was that the issues mostly occurred when the HMD did not update directional changes fast enough and when users could not control their own movement. It was determined that reducing

[5] `http://en.wikipedia.org/wiki/Virtual_reality_sickness`

the motion sickness would be key in achieving a desirable UX and that these areas would receive attention in later development.

Chapter 6
MolecularRift

The first version in the development of a molecular visualizer that allows users to view and interact with molecules in a VR environment was created using the guidelines established in the investigation phase. The development was divided into two main areas, construction of molecular representations and creating definitions for input gestures. A graphical user interface (GUI) was also added to give users a more familiar starting point.

6.1 Development

The development was moved to Unity 5 Personal Edition [41], which is a free version of Unity supporting plug-ins allowing improved graphic and modeling capabilities. Unity (.NET Framework 4.5) was not compatible with the dynamic-link libraries of Kinect's SDK (.NET Framework 3.5). As a consequence the code written using the Kinect v2 SDK could not be used in the Unity scripts. To solve this issue a Unity wrapper [46] was used. The Oculus Rift Unity integration plug-in was used to connect Unity with the HMD.

6.2 GUI

A GUI was created as the start menu. The GUI (Figure 6.1) allows users several options such as configuring custom atom colors and input 3-D models from the net. Upon click, the launch button parses the input file and creates a molecular representation. At the same time, the HMD starts (and the GUI closes) and a new scene containing the molecular model is shown on both the screen and in the HMD. This allows the user wearing the HMD to view the scene in VR while others can watch the same 3-D representation on the computer screen. Simultaneously, the Kinect is enabled, thus initiating body tracking and allowing the user to switch from the keyboard to hand gestures.

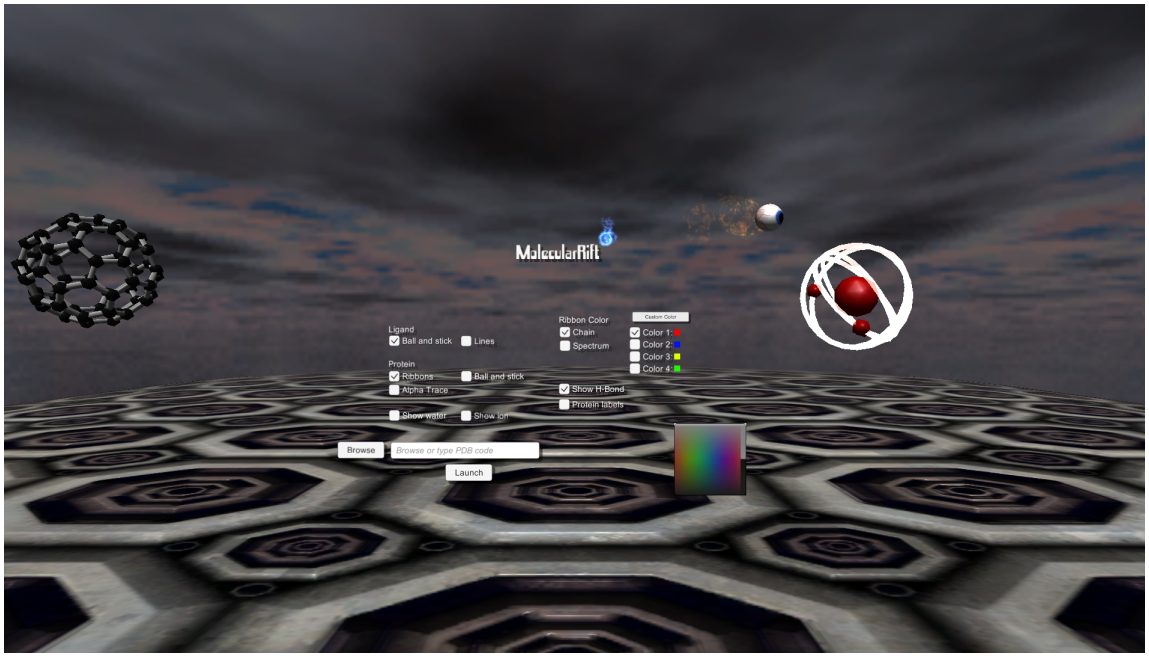


Figure 6.1: The start screen of MolecularRift.

6.2.1 In-Game Menu

A slimmed down in-game menu was developed and added to the VR mode to allow users to change display options in the GUI. This in-game menu (Figure 6.2) launched a cursor that could be steered with hand gestures to select options. In order to reuse gestures from the standard interaction, in an attempt to keep the number of gestures to a minimum, the menu was designed as a separate mode. This meant that users could toggle the menu on and off, thus hiding it from view. When the in-game menu was active all regular interaction were halted and MolecularRift only interprets menu interactions. As a consequence the in-game movement is deactivated while in menu mode.

6.3 Molecular Representations

To render and visualize molecules from files the information needs to be interpreted and represented in a meaningful way. In protein-ligand complexes the primary structures, such as ligands, and the secondary structures, such as protein helices, are traditionally represented in different ways. This approach was chosen in an attempt to make drug designers more familiar with our representations.

6.3.1 Input Files

The molecular representations were created from text data and x,y,z-coordinates as defined by the protein data bank, commonly called PDB-files[1]. The input format was chosen for

[1] `https://www.cgl.ucsf.edu/chimera/docs/UsersGuide/tutorials/pdbintro.html`

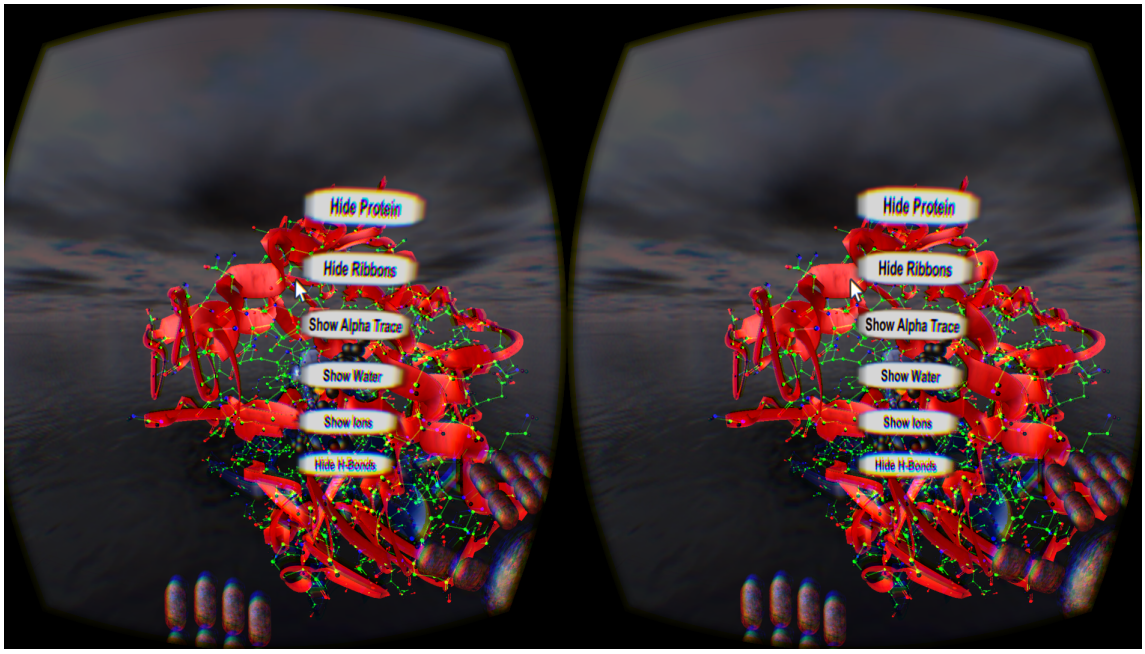


Figure 6.2: The in-game menu of MolecularRift.

its regular use both at AstraZeneca and in the drug discovery community as a whole. The Protein Data Bank[2] allows users to access molecules through PDB-id's. As an example the PDB-file describing horse deoxyhaemoglobin with the PDB-id 2DHB can be retrieved using the URL: `http://www.rcsb.org/pdb/files/2dhb.pdb`. This allowed us to combine the GUI (Section 6.2) with a remote download that retrieved PDB-files online using the PDB-ID.

The coordinates in PDB-files can be generated from experiments such as X-ray crystallography[3] or NMR spectroscopy[4] thus creating models of the molecular complexes.

6.3.2 Primary Structures

Primary structures such as ligands are traditionally represented in four ways, namely: lines, stick, ball-and-stick and sphere [17]. Lines and ball-and-stick were implemented in the first version of the program since they are commonly used.

Lines

The line representation, which only renders the bonds between atoms and not the atoms themselves, was created using Unity's standard cube object. The cube was selected since the 3-D representation of a line is a thin rectangle which was achieved by rescaling a cube object.

[2] `http://www.rcsb.org/pdb/home/home.do`
[3] `http://en.wikipedia.org/wiki/X-ray_crystallography`
[4] `http://en.wikipedia.org/wiki/Nuclear_magnetic_resonance_spectroscopy_of_proteins`

Bond information is generally explicitly defined for primary structures ("HETATM") in PDB-files. This allowed the coordinates for both atoms in the bond to be extracted using their numeric id's and string matching. A cube was created in the centre of the two atom locations. The cube was scaled to adopt a length equal to the distance between the atoms. The bond was aligned correctly by facing the cube transform[5] towards one of the atom positions, thus rotating the bond around its center and placing each end point at one of the atom positions. The entire bond creation process can be seen in Figure 6.3.

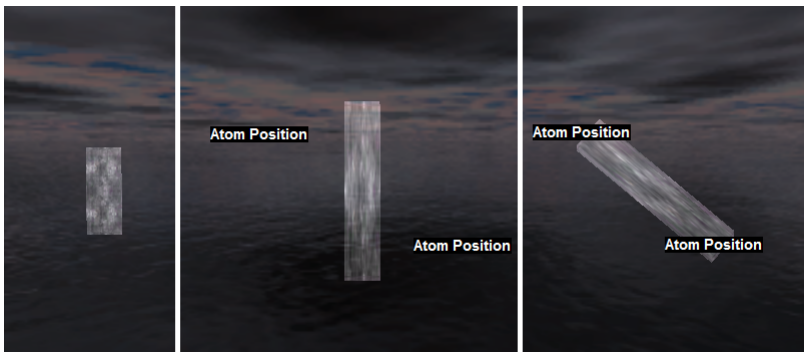


Figure 6.3: The three steps included in bond creation: Rendering the basic bond (left), scaling the bond according to atom distance (middle) and rotation towards one of the atom positions (right).

Ball & Stick

For the ball-and-stick representation the primary structure atoms were handled differently if they were part of the ligand or the protein, thus allowing the user to differentiate between the two. For ligands the line representation was used for bond while atoms were represented by Unity's standard sphere object. The spheres were colored according to atom type and scaled to match Bondi's calculations of atom radii [47]. The ligand representation can be seen in Figure 6.4 and a list of atom colors and radii is shown in Table 6.1.

Table 6.1: Atom Properties of the Ligand Representation

Atom Type	Radius Ratio	Color
Carbon	1.0	Black
Oxygen	0.715294	Red
Nitrogen	0.729412	Blue
Iron	0.912941	Orange
Hydrogen	0.564706	White
Default	0.7	Dark Green

The protein representation, while using the same atom type colors except for carbons which were rendered with the default color green, used a constant radius scale of 0.3. Since

[5] `http://docs.unity3d.com/ScriptReference/Transform.html`

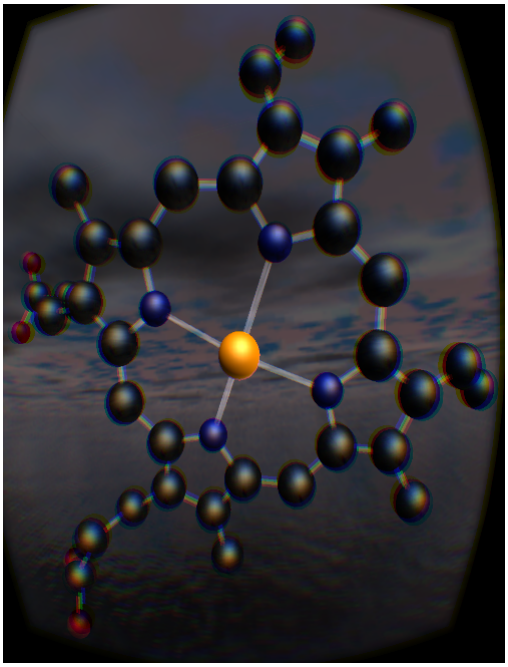


Figure 6.4: A heme ligand represented with ball-and-stick.

bonds between atoms ("ATOM") in proteins are not provided in the standard PDB-files a different method had to be utilized. Instead, bond information was retrieved by calculating atom distances in Euclidean space. We also wanted to be able to represent different bond types. Supported types were single bond, double bond and aromatic rings (triple bond in the future). By extracting the atom types the distances between the atoms was used to determine if a bond was present and what type such a bond would have. The distance 1.6 Ångström was used as the upper limit of where a bond was present for all atoms. The only exception was Sulfur which was allowed a distance up to 2.0 Ångström due to its bigger size. The bond orders were evaluated using the atom types and the distance.

One challenge when rendering bonds is to keep the bond type visible independent of which direction it is viewed. Double bonds, commonly represented with two parallel lines, would for instance look like single bonds if viewed from the side. This was solved using Unity's Line Renderer[6] which is a 2-D line that always faces the viewer, thus creating a 3-D effect and allowing different bond types to be represented indifferent of the viewers position. An example representation using this technique is displayed in Figure 6.5. As can be seen the dotted line of the aromatic ring is not always facing inwards. This limitation is caused by the rotating lines.

6.3.3 Secondary Structures

Protein helices and other secondary structure refers to general 3-D forms of the local environment. The five standard representation modes for secondary structures are: ribbon, strand, cylinder & plate, C alpha trace and B factor tube [17]. We started of by implementing the ribbon and C alpha trace representations since they are commonly used.

[6] `http://docs.unity3d.com/Manual/class-LineRenderer.html`

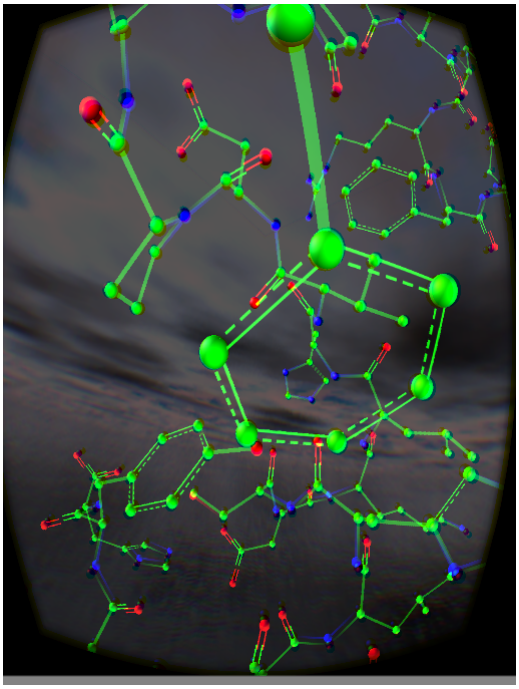


Figure 6.5: An aromatic protein ring represented with ball-and-stick.

Ribbons

Protein chains are often represented by ribbon like structures to ease the visualization of secondary structures and protein folding. If the protein contains an alpha helix[7] MolecularRift renders it as a spiral.

To render the ribbon shapes custom mesh objects[8] had to be created. These meshes were rendered in a way that interpolated the atom positions in a ribbon like fashion.

The interpolation was done using mirrored Bézier splines[9]. By placing the Bézier control points at each atom position a custom curved line that correctly interpolated the protein structure was created. These points were modified to mirror each other which created a serpentine like shape. It was decided to only include the alpha carbons[10] as control points for the alpha helices, to create a smooth spiral, as compared to the standard protein depiction.

The ribbon mesh was created using a modified version of StarManta's TubeRenderer [48]. To obtain an elliptic, almost flat, shape the original tube was modified by dividing the y-axis by a factor of 10.

A screenshot from MolecularRift in Figure 6.6 shows both types of protein ribbons.

[7] http://en.wikipedia.org/wiki/Alpha_helix
[8] http://docs.unity3d.com/ScriptReference/Mesh.html
[9] http://en.wikipedia.org/wiki/B%C3%A9zier_curve
[10] http://en.wikipedia.org/wiki/Alpha_and_beta_carbon

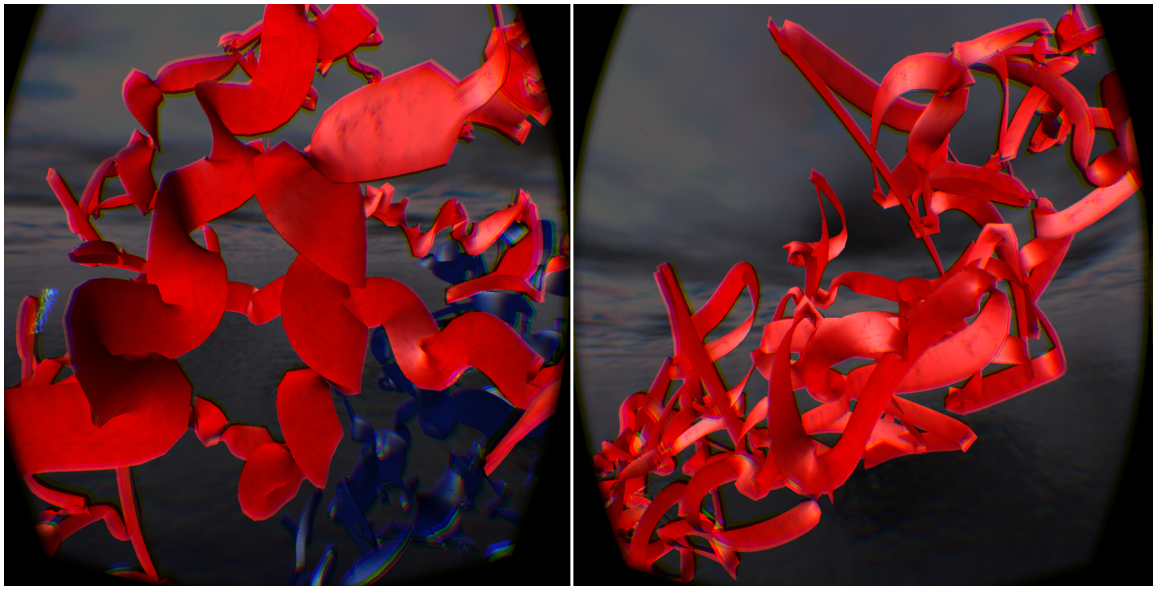


Figure 6.6: Two different protein structures rendered as ribbons to display the alpha helix (left) and standard (right) representations.

C Alpha Trace

The C alpha trace represents the backbone of the protein structure and is defined by alpha carbon positions. It was decided to represent the C alpha trace with the same approach as for the alpha helix (Section 6.3.3) but with a smaller radius than that of the alpha helices, thus making it appear more like a line.

6.3.4 Hydrogen Bonds

Hydrogen bonds[11] are electrostatic interactions that typically occur between hydrogen atoms and nearby electronegative atoms, such as oxygen and nitrogen. Thus molecular models often include options to highlight hydrogen bonds between proteins and ligands since they mark the interaction points between the two.

A function was written to identify intermolecular ligand-protein hydrogen bonds. All atoms in the PDB-file that were located in protein chains were iterated over and all oxygen and nitrogen atoms were recorded as potential hydrogen bond candidates. The heteroatoms[12], labeled "HETATM" in the PDB-file, were selected as candidates if their atom types were nitrogen or oxygen, as they are the most commonly present atoms capable of forming hydrogen bonds. The candidate atoms were inspected and all labeled water ("HOH") were removed. When all candidates were identified protein-ligand atom pairs with a distance of 3.5 Ångström or less were selected, since those atoms were considered close enough in space to form a hydrogen bond.

The Unity's particle systems[13] was used to simulate the hydrogen bond. The particle system creates a random emission of small objects in a specified shape and direction. By

[11] `http://en.wikipedia.org/wiki/Hydrogen_bond`
[12] `http://en.wikipedia.org/wiki/Heteroatom`
[13] `http://docs.unity3d.com/ScriptReference/ParticleSystem.html`

applying one cylindrical particle system to each of the atoms and facing their emission direction towards the other atoms position an animation moving over the bond, as seen in Figure 6.7, was created.

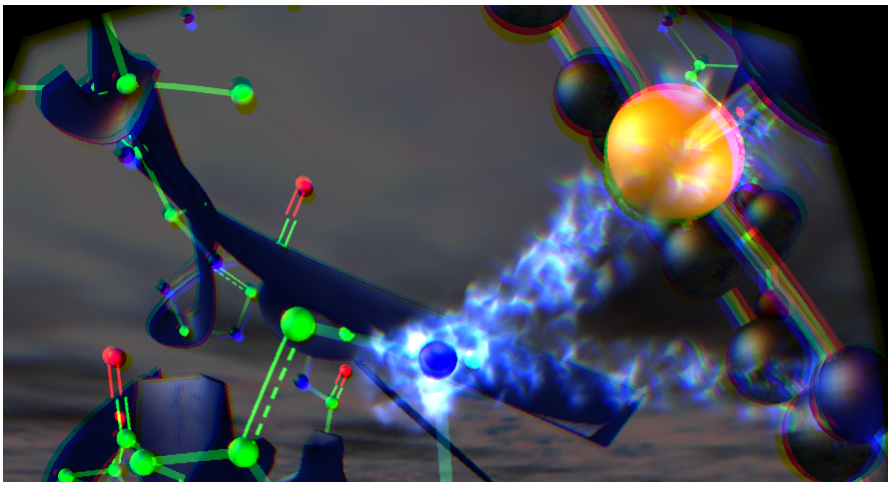


Figure 6.7: Intermolecular ligand-protein hydrogen bonds.

6.3.5 Protein Labels

Protein labels, seen in Figure 6.8, are used to display information about the protein residues. The labels were implemented using Unity's TextMesh class[14] which included functions for displaying 2-D text in a 3-D environment.

Each atom is labeled with its residual name, a three letter code describing the amino acid[15] type , and a sequence number, an integer that represents the amino acids position in the protein chain, separated by a dash (residue name - residual sequencing number). All rows labeled "ATOM" in the PDB-file were selected. In the syntax for PDB-files "ATOM"-rows contain the residue names on column 18-20 and the sequence number on column 23-26, thus allowing the extraction to be done using standard string operations.

6.4 Gestures

The standard gestures described in Section 5.1.4 were used when defining the gestures. Focus when implementing the gestures was moved towards learnability, since the design capabilities were limited by the sensors ability to differentiate between discrete gestures. Many of the gestures, such as rotation and movement, required user defined direction. To limit the amounts of gestures a user had to learn the direction component was therefore designed to be the same in all gestures which required it.

A visual representation of the gestures can be seen in Appendix D.1.

[14] `http://docs.unity3d.com/ScriptReference/TextMesh.html`
[15] `http://en.wikipedia.org/wiki/Amino_acid`

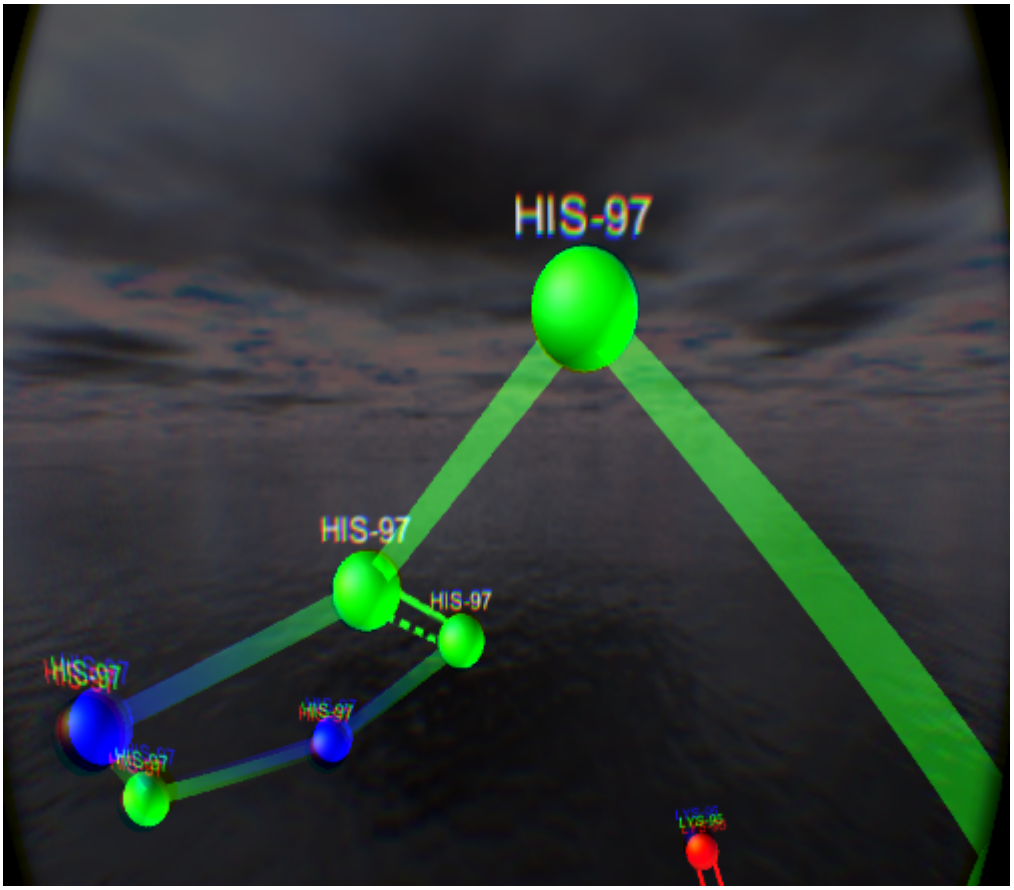


Figure 6.8: Protein labels containing residue name and sequence number.

The gestures were translated into in-game actions using the OVRPlayerController class[16] defined by the Oculus Rift Unity integration plug-in. The OVRPlayerController was also used to define keyboard shortcuts (see Appendix C) for all gestures, thus allowing usage without the Kinect sensor.

6.4.1 Movement

The gesture for movement was not only used for navigation but also included as a component in other interactions. For instance, it is also used in the gesture for translation, where the user relocates a molecule using the movement gesture. The gesture for controlling the cursor (closed right hand), used in MyRemote (Section 5.1.4), was modified slightly. This time, depth tracking was added to allow additional movement in the forward and backward directions. The Kinect would thus track the right hand and move the point of view camera in the scene according to the direction and speed of the hand movement. A maximum movement distance per frame was implemented to ensure that the camera view would not leap on fast movements. This allowed users to travel upwards in the scene by closing their hand and raising it. If a user wanted to move up additionally, when the hand reached its

[16] https://developer.oculus.com/doc/0.6.0.0-unity/class_o_v_r_player_controller.html

maximum height, the user could lower the hand while keeping it opened, thus not causing the camera to travel. By repeating the gesture the user could continue moving upwards.

6.4.2 Rotation

In order to allow users to rotate molecules, a center of rotation was required. This was performed by creating a simple function that summed up the coordinates of all atoms and divided by the total number of atoms. The gesture interaction for movement was used since the rotation required a user defined direction. To differentiate between movement and rotation the addition of a left hand lasso gesture (Figure 5.10) was added to signal rotation. This allowed the user to control the rotation by moving a closed right hand if the left hand gestured a lasso. The movement of the right hand was tracked and the molecule's transform (the position, rotation and scale of the object), included in its mesh object, could be rotated using the tracked directions, thus allowing rotation around all three axes simultaneously.

6.4.3 Translation

Two previously described methods were combined to translate (move) molecular structures in the scene. The movement was controlled with a closed right hand. Additionally a closed left hand differentiated the translation gesture from movement and rotation. The translation was achieved by tracking the direction of the right hand and updating the position of the molecule according to that direction.

6.4.4 Zoom

Zooming could not be implemented by moving the camera closer to the zoomed object, as it is done in a ordinary 3-D viewer, since the VR view is defined in 360 degrees. The reason being, moving the camera forward would only zoom in one direction and if the user turned its head the effect would be eliminated. The zoom was instead implemented as increasing (zoom out) and decreasing (zoom in) the field of view, thus creating the desired zoom on the whole scene.

The zoom gesture was defined as moving a right hand lasso in the horizontal direction. Moving the hand to the left zoomed in and moving it to the right zoomed out.

6.4.5 Toggle Menu

In order to toggle the in-game menu described in Section 6.2.1 an uncommon gesture, that would not interfere with the regular interaction, was designed. Stokoe's location aspect was used to differentiate it from other gestures [28]. By tracking the coordinates of the users head, using the head joint defined by Kinect's SDK, it was possible to determine if the gesture was made below or above the head. The toggle menu gesture was defined as a sequence of the open and closed right hand, going from open to closed. In addition the gesture had to be performed above the users head. This allowed regular gestures to be performed below the head without accidentally launching the menu.

6.4.6 Cursor Movement & Cursor Click

The cursor movement was done in 2-D and could therefore be implemented as described in MyRemote (Section 5.1.4) with a closed right hand. The cursor click was launched by performing a lasso with the left hand, allowing the user to move the cursor and click at the same time.

It was decided to implement an in-game cursor due to conflicting .NET frameworks in Unity and the interop assemblies[17] from Visual Studio used previously for cursor input. The in-game cursor consisted of a regular cursor arrow which location was updated each frame according to the hand gestures. When a user performed a cursor click the location coordinates of the arrow were compared to the location of each menu buttons. If the arrow position superimposed any button that buttons function was called, thus simulating a cursor click. The cursor click was also implemented to include a 0.5 second lock-out, thus preventing it from launching cursor clicks every frame.

6.5 Focus Target - Multiple Users

The focus target is the user that the Kinect sensor is currently tracking. To allow more than one person to stand in front of the sensor, without interfering with the interaction, such a focus target was defined. Kinect can track up to 6 persons simultaneously, using random forest algorithms. All tracked bodies are given unique id's which are stored in a list. We defined the focus target as the first person in the list of tracked bodies. To allow users to take turns, the focus target was changed if the tracked user no longer was detected by the sensor. A new focus target was identified from the top of the list when the old focus target exited the sensors range. If the list was empty the algorithm would continue to search until a new user could be detected.

6.6 Features

This section contains a summary of the visualization options implemented in Molecular-Rift. The features are presented in Table 6.2 below to simplify comparisons with available 3-D viewers such as iview and PyMol.

[17] `http://en.wikipedia.org/wiki/COM_Interop`

Table 6.2: Features of MolecularRift.

Category	**Feature**
File input format	PDB
Structure coloring	Chain Spectrum
Primary structure	Ball-and-Stick Lines
Secondary structure	Ribbons Alpha Trace Ball-and-Stick
Residue labeling	Yes
Protein surfaces	No
Intermolecular ligand-protein hydrogen bonds	Yes
Ions & water	Ball-and-Stick Ball-and-Stick
Virtual reality effect	Oculus Rift
Environment manipulation	Hand gesture Keyboard
Manipulation mode	Rotation Translation Zooming

Chapter 7
IterationOne

Having completed a first working version of MolecularRift, the focus was shifted towards improving its usability and the overall UX through iterative development.

7.1 MolecularRift Improvements

Additional gesture feedback was added to make the learning easier for new users. Methods for reducing motion sickness were also explored to allow users to run longer sessions with the tool, thus enabling us to do more extensive user tests while simultaneously increasing the UX.

A user test was created in an attempted to evaluate the developed interaction gestures and visualization implementations but also to gather data about the testers previous VR experiences and other possible suggestions for improvement.

7.1.1 Performance & Motion Sickness

Nausea was experienced when developing with the HMD. The cause was not completely understood but multiple factors such as latency and stuttering certainly play a part. From the experiences in the previous development it was believed that reduction of in-game stuttering[1] and improvements of the HMD's response times would help.

The reason for the in-game stuttering was identified to be caused by Unity's DirectX[2] settings. Unity natively launched applications with DirectX 11 mode disabled, thus causing the program to run older versions. By creating a shortcut to the executable file and adding the line "-force-d3d11" the application could be forced to run with DirectX 11, thus removing the stuttering.

[1] http://en.wikipedia.org/wiki/Micro_stuttering
[2] http://en.wikipedia.org/wiki/DirectX

When developing it was noticed that the HMD response times were very uneven. This behavior was believed to be linked to large CPU spikes that occurred frequently during run time. The cause could successfully be identified using Unity's built-in profiler[3] that visualized the CPU usage and draw calls. For each time period the CPU usage could be divided into its core components such as rendering, scripts and physics. The physics engine was found to be the main source of performance issues in MolecularRift. Unity uses its built-in physics engine for tasks such as object collisions and in-game gravity. These features were not utilized by MolecularRift since all objects were stationary, unless moved by an in-game script. In addition all objects allowed users to move through them, thus making the native collision detection redundant. The CPU usage spikes could be reduced by disabling native physics components natively added to Unity objects. Overall, improved performance was obtained and the in-game frames per second (fps) doubled to values between 20 and 30 fps, depending on the size of the molecular structure.

We still experienced slight motion sickness after the improvements but its effect was reduced enough to allow long sessions without discomfort.

7.1.2 User Feedback

Two types of feedback exists in this project. The first is feedback gathered from user tests, such as questionnaire responses, that is used to improve future development. The second is feedback that a user receives from the program to improve the interaction. This section will discuss the later version of feedback.

Grant Wiggins defines the conditions for effective feedback as [49]:

"Effective feedback requires that a person has a goal, takes action to achieve the goal, and receives goal-related information about his or her actions."

In this case the goal is a successfully interpreted gesture, the action is to successfully perform the gesture and the feedback is an in-game event (such as movement). The in-game event might not be feedback enough for a user to efficiently learn new gestures, especially if the gesture is performed in the wrong way and no event is triggered. To provide user feedback, systems for both gesture and sound feedback were implemented in the VR mode.

Gesture feedback

While wearing the HMD it is helpful for the user to receive feedback on the performed gesture. Two in-game hands were created in Unity for this purpose. These hands, which can be seen in Figure 7.1, are moved through animations to mimic the gesture interpreted by the system.

An animation for hands which are not tracked by the sensor was added since the HMD obscures the users vision. This animation moved the in-game hand to the bottom of the screen. In this way the user is made aware of when their gestures are being tracked and when they are not.

Unity's animation state machine was used to control the animations by giving each gesture a separate state. This allows the state machine to launch transition animations spe-

[3] `http://docs.unity3d.com/460/Documentation/Manual/ProfilerWindow.html`

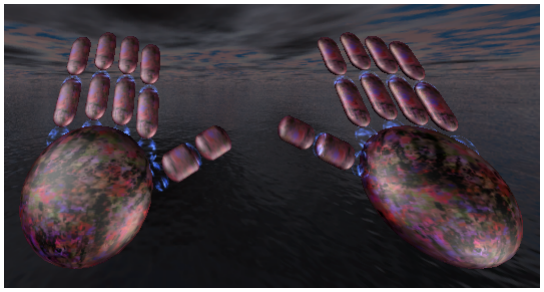


Figure 7.1: The animated hands that provide gesture feedback.

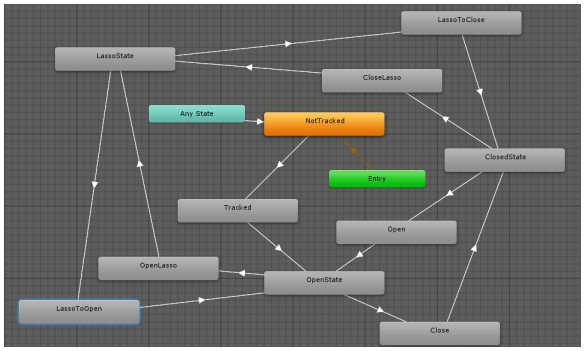


Figure 7.2: The animation states used during preliminary development.

cific for each transition, thus allowing smooth animation movements. The state machine, containing animations for Kinect's standard gestures, can be seen in Figure 7.2.

Sound feedback

Sound effects were added to functions where the visual feedback was expected but not adequate, such as the in-game menu toggle. The menu position was designed to be static, since it had to be drawn on a 2-D plane, and users might not immediately see it. The visual feedback of a menu appearing can thus be missed. By utilizing Unity's AudioSource class[4] a sound could be played upon menu activation/deactivation, thus telling users that the interaction was successful. Additional sound effects were added to menu selections since some of the options could lack visual effect. For example, selecting "show ions" in a model without ions would not render any changes in the scene, thus not giving the user feedback that the option was selected.

7.2 User test

User tests were setup to gather feedback of the UX and to evaluate the gestures from a usability perspective. As shown by the Nielsen Norman Group, usability data is best gathered when the users are moderated minimally but a script for the usage can still be supplied to narrow the focus to specific usage aspects[37]. A usage script was therefore developed

[4] `http://docs.unity3d.com/ScriptReference/AudioSource.html`

to make sure that the users tried out all the different gestures. Additionally a questionnaire was developed to gather self-report feedback after the test.

The selected users were 9 AstraZeneca employees at the site in Mölndal. Most of the test users had a background in drug discovery due to the test being performed at AstraZeneca.

The users were initially given a Gesture Guide (Appendix D.1) and a 5 minute demo of the program. They practiced the gestures on their own for about 10 minutes. After practicing they received a list of tasks (Appendix D.2) to perform. The test was concluded with a questionnaire (Appendix D.3) aimed at evaluating the UX and identifying improvement areas for the next development cycle.

7.2.1 Questionnaire Results

Answers from all test users were collected and the result can be seen in Appendix D.4.

The questions regarding previous experience revealed that only two people had previously tried Microsoft Kinect and none had used Oculus Rift. If the result is representative for future end-users care should be taken to make MolecularRift easy to learn, since most users are new to VR.

The users ranked all gestures from 1 - 10, where 1 is best, according to usability. In addition to the ranking they were also asked to name their favorite and the worst gesture. The results can be seen in Table 7.1.

Table 7.1: The usability scores for all gestures in iteration one.

Gesture	Average Score	Favorite	Worst
Movement	4.0	5	2
Zoom	4.8	2	2
Rotation	5.7	1	1
Translation	6.0	0	0
Toggle Menu	6.0	1	4
Cursor Movement	3.9	0	0
Cursor Click	2.6	0	0

The results were mixed and no unanimous favorite gesture could be identified. Movement was voted as the favorite by 5/9 but was also voted as the worst by 2/9 and did not get the best usability score. The cursor click got the best score and was ranked as easy to perform by all users but no one ranked it as their favorite. Part of the explanation for this might be that it was performed less frequently compared to other gestures.

On the other side of the scale toggle menu was voted worst by 4/9 and with only one user voting it as the favorite gesture. Rotation, translation and toggle menu were all scored as hard to use. From the free text answers it became clear that the sensitivity was an issue for both translation and rotation. Some users remarked that the direction of the rotation was counter intuitive and that it would be easier if only one axis was rotated at a time.

The toggle menu was the only gesture that had to be performed in sequence. The gesture had to be done rather quick and therefore had a slight learning curve. Most users

struggled the first time but once they got it right they could easily repeat it. It was also the only gesture that some users failed to perform entirely.

When asked about the overall experience, most users wrote that the experience was mostly positive, and the most frequently used word was "cool". When compared to standard visualizers many users stated that the 3-D view itself was better but that the program needed improved control and more visualisation options.

7.2.2 False positives - Unintended Interactions

Some unintended interactions were observed during the user test. The most common was accidental launches of the in-game menu. Since the menu was toggled by users closing the right hand quickly above their head it could be triggered while intending to move upwards.

Another observed unintended interaction was that some users tended to have their left hands slightly raised when only wanting to use the right. This caused errors since the left hand signaled movement to be interpreted as translation or rotation.

7.3 Conclusions

The user test (7.2) results revealed that the users had low or no previous experience with VR applications. This would require a low learning threshold for the final version of MolecularRift.

It was also concluded that even though many users had a positive UX they found the interaction difficult. Many stated that interactions such as movement felt natural but they had trouble remembering the gestures for most of the other commands. Many of the difficulties encountered during interaction could be improved by practice. It was decided that additional steps would be taken to improve the worst performing gestures (Rotation, Translation and Menu Toggle).

The unintended interactions (7.2.2) was partly due to inexperience with the interaction method. When users were made aware of a hand being tracked unintentionally, either by the test supervisor or the gesture feedback, the users quickly adapted. The accidental launching of the menu was worse since that was an intended interaction that the program interpreted differently from what the user expected. One solution was to replace one of the two overlapping gestures, thus eliminating the issue. When comparing the two conflicting gestures usability scores toggle menu did much worse it was therefore selected for refactorization in iteration two.

No complaint concerning motion sickness were mentioned during the tests. The previous performance improvements and reduction of in-game stuttering could therefore be considered successful and no further actions were scheduled for iteration two.

Chapter 8
Iteration Two

The focus of this second development iteration was to improve existing gesture interactions according to feedback gathered from the initial user tests.

The applied modifications were evaluated by a new test group, this time at a science symposium. MolecularRift was showcased at AstraZeneca's IMED Science Retreat which presented the opportunity to reach a much larger and diverse audience compared to the test group in the previous iteration.

8.1 MolecularRift Improvements

The gesture for menu toggle was redesigned to avoid interfering with the movement gesture. In addition, the rotation and translation gestures were modified in an attempt to give users more control over the in-game response.

8.1.1 Menu Toggle

The gesture for movement could potentially be interpreted as the gesture for toggling the in-game menu if the user had its right hand above the head. This could be avoided by simply shifting the gesture from the right hand to the left hand. The order of sequence (from open to closed hand) was kept as well as the location aspect. The decision to switch hand was motivated by the fact that the left hand was only used for signaling. The signaling interactions did not require any hand movements which meant that the left hand should not be located above the users head during regular interaction.

A keyboard shortcut ("N") was also added for menu toggling to solve cases where users could not perform the gesture. This was not an optimal solution and more attention to improve that gesture was given later in development.

8.1.2 Rotation

The rotation interaction was criticized in the survey for its high sensitivity and that the rotation direction was hard to control. These two points are related since high sensitivity would amplify issues with rotation control.

The sensitivity could be reduced by decreasing the angle at which the object was rotated. Different angles were tested during development to identify the best trade-off between rotation speed and control. It was decided that a factor 2 of the original angle (0.018 degrees) was a good trade-off.

In an attempt to improve the control of rotation direction it was modified to only spin around one axis at a time. When the rotation gesture was performed the horizontal and vertical change would be compared and the highest value would determine the rotation axis.

8.1.3 Translation

The translation gesture was designed very similarly to the movement gesture, which scored better during evaluation. Thus, the gesture was optimized by aligning its sensitivity to resemble that of the movement gesture. The reduced sensitivity was achieved by decreasing the translation speed with a factor of 3.

8.2 User Test

MolecularRift was demonstrated at AstraZeneca's Science Reatreat in Mölndal the 6th of May 2015 (Figure 8.1). Attendees were given the opportunity to test the program. Since the user test was performed at a symposium the tests had to be much shorter compared to the previous iteration. Users would only receive a brief introductory lesson which prompted us to only include four basic gestures (movement, rotation, translation and zoom) in the test. The users lined up to test MolecularRift and were given verbal instructions while they used the program. After the test run each users were asked to answer a short questionnaire, which can be seen in Appendix E.1.

8.2.1 Questionnaire

The questionnaire for the previous iteration had revealed that users found the survey scoring scale difficult to understand. The order of the integers used to grade the gesture usability was not obvious. As a consequence some testers, that did not read the instructions carefully, misinterpreted the direction of the scale. As a solution we therefore changed the questions to use a visual analogue scale[1] (Very Easy, Easy, Medium, Hard, Very Hard) which had a obvious order.

[1] `http://en.wikipedia.org/wiki/Visual_analogue_scale`

Figure 8.1: Demonstration of MolecularRift at AstraZeneca's Science Reatreat.

8.2.2 Questionnaire Results

We gathered questionnaires from as many as 31 users during the Science Retreat. The results, which can be seen in Figure 8.2, were overall positive. Comparison of the tests from iteration one with the current user test is non-trivial since they differed both in execution and measurement scales but similar tendencies in gesture performance could be identified. For instance the rotation and translation gestures remain more difficult compared to Zoom and Movement, which were assessed as easy by most users.

8.2.3 Suggestions From Testers

During the symposium we also gathered ideas and remarks from users about their experience and improvement ideas. The suggestions we deemed feasible to implement during the project scope are described below.

Customizable Sensitivity

One improvement mentioned was personalized sensitivity for interactions such as rotation and translation. This could potentially solve issues regarding the sensitivity tuning and at the same time make the tool more versatile.

Center of Rotation

Suggestions to allow the user to defined the center of rotation were also made. The current implementation always rotates around a center point which is calculated as the mean position of all atoms. Since a ligand-protein complex contains mostly protein atoms the rotation is effectively done according to the protein chain positions. Many chemists would instead like the option to rotate around the ligands, which often is the point of interest.

Figure 8.2: The questionnaire results from iteration two.

Gesture Recognition Sensitivity

One user noted that the model stuttered back and forth when he intended to hold the gesture still. This was caused by the sensitivity of the gesture recognition. A cut-off for the smallest recognized movement was suggested to amend this. Such a cut-off would allow users to maintain a gesture without interacting with the model, whereas the current version requires that the user stops performing the gesture.

Movement

An issue that was often raised was that the movement gesture had to be repeated when traveling far in the scene. A suggestions that the movement would continue without having to repeat the gesture was put forward.

8.3 Conclusions

Given the size of this second iteration test group quantitative data could be gathered. In general, these larger test groups has an increased chance of reporting a wider range of usability problems [38]. This can be compared to the smaller test group used in the first iteration, which mainly finds the major issues.

In the previous test (iteration one), all users got the same introduction to the program. At the symposium some users were thrown straight into the program without any previous experience while others had the opportunity to watch several users before trying it them-

selves. This probably made the experience easier for some users but the preparation was still less extensive then the lessons offered to testers in iteration one. The questionnaire results showed that most user rated the gestures as easy or medium to perform. Taking into account the shorter practice times for these users the results suggests that the modifications applied to the rotation and translation interactions were improvements. Since users at a science fair were able to successfully perform the gestures without previous practice suggested that the learning threshold seemed to be rather low. It was concluded that the gesture interactions usability was likely to satisfy end users.

The larger test group allowed the collection of a wider scope of opinions. The questionnaire evaluated the usability of the gestures but many users also gave suggestions verbally during the test. The discussions with the testers generated a list of suggestion (Section 8.2.3) that could potentially improve the UX and was therefore added as possible improvements for the next iteration.

For none of the 31 testers the in-game menu was launched unintentionally, suggesting that moving the gesture to the left hand was an improvement. Since the toggle menu gesture itself was not tested in this iteration the issues users experienced with its complexity to perform was not evaluated. Adding a keyboard shortcut for menu toggle solved the immediate problem but refactoring of the gesture sequence, to make it easier to perform, was one of the top priorities for the next iteration.

The symposium environment revealed previously unknown challenges for MolecularRift. During previous testing there had been at most three people moving around in the test room. At the symposium there were larger crowds and more background movement for the sensor to handle. Issues with selecting the correct focus target, as described in section 6.5, became apparent almost every time a user tried to hand the control over to a new user. The sensor often selected a person from the audience as focus target instead of the intended user. It was often a non-trivial task to determine which person in the crowd represented the new focus target, thus forcing all spectators to move from the sensor's FOV or restarting the program.

A possible solution would be to evaluate each tracked person using attributes such as location and orientation when selecting the focus target. Another method would be to add functionality for switching focus target, such as a keyboard command, that would not require the current focus target to leave the sensor's FOV.

Chapter 9
Iteration Three

Iteration two (Section 8) gathered valuable development information thanks to many enthusiastic testers that gave relevant feedback. The most urgent issue that was revealed was the focus target. Implementing a method for selecting new focus targets would allow the tests to run smoother, thus improving the testers experience while also allowing more time to be spent on actual testing. In an attempt to further increase the UX some of the user suggestions from the previous iteration were also explored.

The user test were scheduled for a new symposium, this time the CVMD Medicinal Chemistry Department Day at AstraZeneca.

9.1 MolecularRift Improvements

As a result of previous feedback, a new gesture to toggle the in-game menu was developed using machine learning and a redesign of the focus target algorithm was implemented. The redesign of in-game movement and gesture recognition was performed in an attempt to meet user suggestions.

9.1.1 Focus Target

Two solutions had been proposed (Section 8.3) to improve the focus target. First, by using the keyboard to iterate through all tracked bodies and secondly by evaluating the most likely target using location and orientation. It was decided that a function for manually switching focus target would be more versatile, as compared to an evaluation algorithm, since it would allow switching users regardless of their position.

The function to manually switch the focus was implemented using Kinect's list of identified bodies. If the keyboard hotkey ("P") was pushed the algorithm was designed to iterate through the list of bodies until the focus target was identified (using the tracking id). With the focus target identified the algorithm continued one more step and selected the

next tracked body as the new focus target. If the end of the list was reached the algorithm would start over from the top. This made sure that old focus targets could be reselected.

9.1.2 Toggle Menu

As described previously (Section 3.2.1) there are two main methods available when developing new gesture for Kinect, machine learning and heuristic rules. Kinect's development team currently recommends machine learning as the implementation approach [32]. Support for machine learning was available through the Kinect SDK's Visual Gesture Builder[1]. A gesture where a closed left hand with the thumb raised (Figure 9.1) was designed due to its unique joint positions.

Figure 9.1: The new toggle menu gesture.

Visual Studio v2.0 (included in the Kinect SDK) was used to record IR, depth, body index, body frame and telemetry[2] data while users performed gestures. Three users were recorded to increase the diversity of the model. The frames were displayed as IR and body index (Figure 9.2) which allowed them to be tagged manually. Using the tagged frames a model for the toggle menu gesture was generated using Visual Gesture Builder.

The additional requirement that the left hand had to be at the same height as the head was added manually to improve the model.

9.1.3 Auto-Movement

Users had previously commented on that they had to repeat the movement gesture (Section 8.2.3). In an attempt to reduce the number of performed gestures an auto-movement gesture was implemented. That is, if a user performing the movement gesture stopped the right hand but kept the gesture (closed right hand) the program would continue movement in the previous direction. This meant that the user did not have to repeat the gesture unless a change in direction was desired. An acceleration of 1% per frame was also applied in an attempt to perform long distance movement more efficient. The acceleration component was only applied to the auto-movement gesture and was reset to 0% between each usage.

The new implementation also applied to rotation and translation which both had movement as a key component.

[1] `https://msdn.microsoft.com/en-us/library/dn785304.aspx`
[2] `http://en.wikipedia.org/wiki/Telemetry`

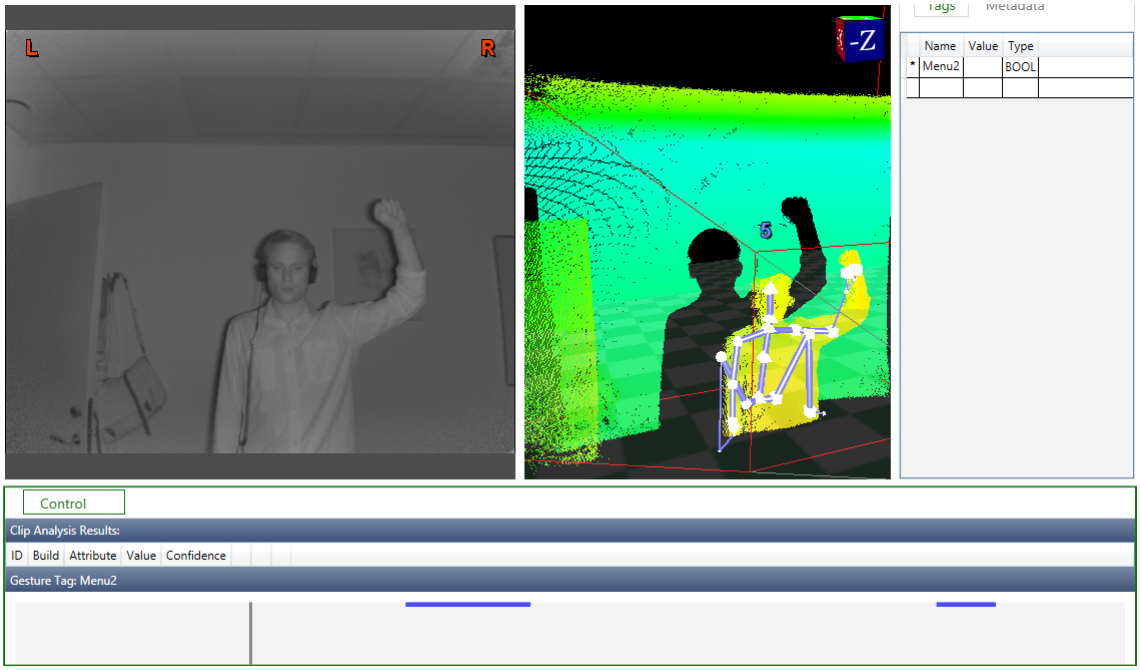


Figure 9.2: Gesture tagging in Visual Gesture Builder.

9.1.4 Gesture Recognition Sensitivity

As suggested by users in iteration two (Section 8.2.3) a cut-off limit was added to define the smallest recognized movement of a gesture. The cut-off value was defined so that the intended interaction was not effected. For gesture movement in vertical and horizontal direction the cut-off could be set to 2 pixels, without interfering with regular interaction. The depth (Z-axis) had to be defined separately as it used a different measurement scale. Here the cut-off was instead set to 5 depth points.

9.2 User Test

The user tests took place at AstraZeneca's CVMD[3] Medicinal Chemistry Department Day in the 28th of May 2015. The symposium had less attendants than the previously visited Science Retreat and less quantitative data could be gathered. In addition, the users were all chemists with experience in visualizing molecules as opposed to previous tests. The tests were performed in a similar fashion to the tests performed in iteration two (Section 8.2) where users lined up for the demonstration and were given verbal instructions through out. The aim was to evaluate the implemented user suggestions (from the second iteration) and the new menu toggle gesture.

[3] `http://astrazenecamolndal.com/cardiovascular-imed.php`

9.2.1 Questionnaire

The questionnaire used in the user test (Appendix F.1) included the same gesture evaluation as the questionnaire used in iteration two, thus making the result comparable. Since this was the last iteration within the framework of this master thesis additional information regarding MolecularRift's future potential and overall UX was desired. The information was gathered using four statements that users were asked to grade using a five level Likert scale[4] (Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree), which is a technique commonly used in customer satisfaction studies [50].

9.2.2 Questionnaire Results

The questionnaire was filled out by 14 testers, all chemists at the CVMD department. Due to technical difficulties, most likely due to the another setup in the demonstration room compared to the development room, the toggle menu gesture caused several unintended interactions and was turned off for the remaining tests. The toggle menu gesture was thus removed from the questionnaire results.

The test result for the remaining gestures (Figure 9.3) showed a slight decrease (-8%) in usability for translation and movement, when compared to the results from iteration two (Figure 8.2).

Figure 9.3: The gesture evaluation results from iteration three.

It was observed during testing that the auto-movement confused users. Many users would keep a gesture static while intending to stand still in-game but unexpectedly continued to move. This erroneous behavior continued even after having informed about the

[4] http://en.wikipedia.org/wiki/Likert_scale

details of the gesture. It can be seen that all gestures including the auto-movement feature got worse scores except rotation. This is believed to be due to the fact that rotation did not change the users in-game location.

The statements in the survey received a positive grading (Figure 9.4). 9 out of 14 users strongly agreed with the statement that they enjoyed using MolecularRift which points to a positive UX. Additionally most users saw potential in the use of VR tools in drug discovery.

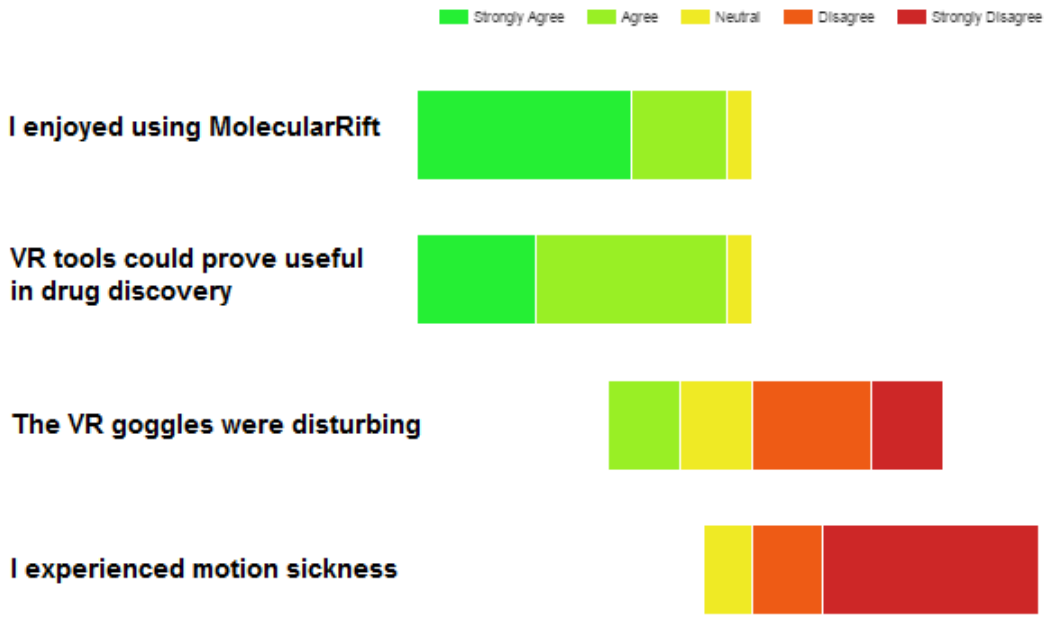


Figure 9.4: The results from the Likert statements in iteration three.

3 out of 14 users found the HMD disturbing. Many users wore glasses which, depending on astigmatism, often are needed to be used under the HMD to provide the stereo effect. Some users stated that having their glasses in combination with the Oculus Rift HMD was uncomfortable.

Most users did not experience any motion sickness during the tests. This hints towards that the improvements performed in iteration one (Section 7.1.1) were successful.

9.2.3 Toggle Menu

An alternative evaluation had to be designed for the in-game menu toggle gesture since we for technological reasons failed to include the gesture in the third user test. Visual Gesture Builder includes building test sets by adding tagged frames and running them through the model. By recording three users when performing different gestures a test set could be created.

The model parses the frames in the test set and evaluates if the gesture is being performed or not for each one. Every frame is given a confidence value (0.0-1.0) representing how certain the model is that the gesture is present. The confidence value can then be compared to the user tagged values to evaluate its accuracy.

When the test clips were run against the model an average root mean square of 0.2 was achieved and additionally no false positives or false negatives were recorded. In addition, three users could toggle the in-game menu without any difficulty or unintended launches. This lead us to believe that the new toggle menu gesture was stable enough to be implemented in MolecularRift.

9.3 Conclusions

The new focus target algorithm allowed switching focus among several potential users and was therefore considered an improvement. The implemented sensitivity cut-off was difficult to evaluate since static gestures triggered auto-movement. This was not foreseen during the construction of the user tests. No decrease in gesture responsiveness was noticed during the tests which suggests that the implementation at least did not reduce usability.

The survey revealed that the testers were positive both about the experience and future potential of MolecularRift. The results are especially promising considering that the test group consisted of potential end users at the department of Medicinal Chemistry at CVMD.

The auto-movement, while potentially useful for experience users, increased the difficulty level of many gestures. The feature was subsequently removed to make MolecularRift more accessible to new users. Future implementations of auto-movement might be created using a continuous gesture, thus allowing users to perform an extra gesture to lock the direction and enable auto-movement.

The developed gesture passed the created test suit and was implemented as the new toggle menu gesture. It was also concluded that the setup, such as sensor position, should be selected to replicate the position used during training to allow gestures created with machine learning to receive expected input data.

Chapter 10
Discussion

MolecularRift achieved the main goal of the project, which was to develop an interactive molecular visualizer that would allow molecules to be rendered in VR. Working gesture interactions were implemented for the main steering functions of MolecularRift. Increased gesture recognition models would be required for more advanced operations, such as molecular editing.

The iterative development process allowed user test feedback to facilitate development. By including test groups and potential end users in the development we made sure that usability and UX could be in focus.

Below application areas for MolecularRift are explored together with a comparison to currently available tools.

10.1 MolecularRift as a Demonstration Tool

During development, MolecularRift was showcased at two science symposiums as well as being tested by many employees at AstraZeneca. The setup, especially the HMD, attracted attention and not just drug designers were interested in testing the tool. Many were eager to try the HMD and to interact with the molecules just for the experience of trying VR. A potential secondary use for the tool could thus get children and students interested in science. By stripping many of the options, such as the in-game menu, new users could quickly learn to navigate MolecularRift.

10.1.1 Roller Coaster

In an attempt to create a demonstration tool without interaction, thus not requiring any instructions, a molecular roller coaster could be developed. By moving the camera along the protein ribbon, using the points of its bézier curve, a user would ride through the molecular complex.

10.2 MolecularRift as an Analytic Tool

The VR environment provides a unique experience compared to available molecular visualizers. The potential for VR tools in drug discovery is promising and adding a new visualization method could prove useful. To be able to compete with industry standard viewers such as PyMol and VIDA additional visualization features needs to be implemented. High customizability of the molecular models makes the tool versatile and adaptable which is key in drug discovery, and science in general. Being able to edit and build in MolecularRift is thus an important success factor.

The 3-D visualisation of MolecularRift is equal to, if not better, than the competition. Adding additional visualization features is just a matter of implementation. Other tools are currently superior in its interaction methods due to the precision and familiarity offered by keyboard and mouse input.

Users are used to performing HCI with keyboard and mouse. The in-game gesture interaction has a learning curve and there was a constant trade-off between the interactions efficiency and learnability. In its current state MolecularRift's strength is the visualization while the gesture interaction is an attempt at supporting an interaction method that does not require external vision. For some users a video game hand controller might be a more natural interaction method. Alternative interaction methods are discussed in the next chapter but new optical interaction methods are emerging rapidly which promises new opportunities for MolecularRift.

Adding features to MolecularRift, through the current or a new interaction method, that allowed users to select individual atoms with precision, combined with additional visualization options, would make it a strong analytic tool. Hopefully leading to added value to real life drug projects through faster and better informed design decisions.

Chapter 11
Future Work

Having recently passed the "through of disillusionment" according to the Gartner Hype cycle [51], the field of virtual reality is now rapidly gaining momentum. Many new releases in the upcoming months might bring new opportunities for the MolecularRift application. As an example, the commercial version of Oculus Rift is due to be released in the spring of 2016 with allegedly improved tracking and reduced motion sickness [52]. In addition, several suggested improvements, which were considered out of scope for this thesis, will be attended to in future versions of MolecularRift. Several examples of such improvements are described in the following sections.

11.1 Improved Parsing

It is believed that improved rendering of the molecular representation can be achieved by implementing a cheminformatics toolkit[1], such as OpenEye's OEChem[2]. This would allow MolecularRift to render correct molecular models on par with the industry standard. The use of cheminformatic toolkits would also facilitate the use of other file formats, such as SDF. The open source chemistry toolbox OpenBabel[3] will be explored in future development. Its source code is available in C# which would allow integration with MolecularRift.

11.2 Molecular Sculpting

The ability of being able to grab and/or edit molecules while taking advantage of the VR environment within MolecularRift is a much desired feature. The introduction of molecu-

[1] http://en.wikipedia.org/wiki/Cheminformatics_toolkits
[2] http://www.eyesopen.com/oechem-tk
[3] http://openbabel.org/wiki

lar sculpting[4] to MolecularRift would be one interesting example to try out. This requires implementing accurate gestures for gripping individual objects. This could potentially be done using Kinect's depth tracking, thus allowing the users to select objects in 3-D space.

The development of molecular sculpting would benefit from having implemented a cheminformatics toolkit, as discussed in Section 11.1, to evaluate the validity of the modified molecules. This may allow MolecularRift to keep correct bond orders and angles when edited. Such an algorithm would reduce the required accuracy of the interaction method, since the toolkit could identify valid atom positions and select the most likely.

Introducing molecular sculpting will be explored in future development.

11.3 Molecular Surfaces

Molecular surfaces are a popular way to visualize protein-ligand complexes and a feature that would make MolecularRift even more useful to drug designers. To represent the molecular surface, a custom mesh that covers the entire protein, as seen in Figure 11.1, must be created. For macromolecules these meshes can consist of over 500,000 polygons [17]. These loads are very high and will impact the performance. The VR experience relies on the HMD updating the scene in real time as the users moves and rotates the view, thus not allowing us to add molecular surfaces without reducing other features at this moment.

One implementation option would be to re-use existing code from open source projects such as GLmol [43]. GLmol has implemented support for four different surface types (molecular, solvent excluded, solvent accessible and van der Waals). The entire implementation consists of approximately 3000 lines of code (LOC), written in Javascript. Since Javascript is supported by Unity it would be possible to integrate it into MolecularRift without major modifications. However the challenge with the computationally complexity and how to best deal with memory issues remain. An acquisition of a strong GPU could solve this issue and the addition of a molecular surface feature is a likely future improvement.

11.4 Alternative Interaction Methods

The Kinect focuses on full body tracking (although this project mainly utilised its hand tracking). Competing sensors such as Leap Motion[5] focuses only on hand tracking and exploring its capabilities could prove useful in future development. Recent advances in eye tracking also offers interesting opportunities as an alternative interaction method.

11.4.1 Leap Motion

Leap Motion develops small (3 inch) 3-D sensors to be used as input device for computers. The sensors come with a SDK packing including skeletal hand tracking. The skeletal tracking model offers a much more detailed hand recognition algorithm than Kinect. In leap motion a finger is treated as 4 different sections, except the thumb which is divided

[4] http://sourceforge.net/p/pymol/mailman/message/5222163/
[5] https://www.leapmotion.com/

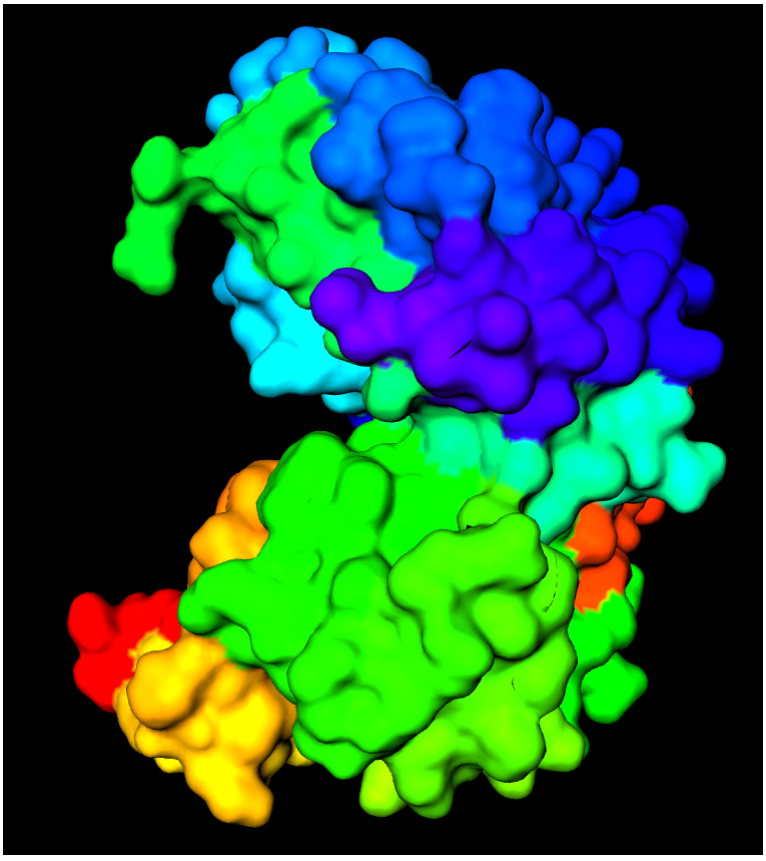


Figure 11.1: A visualization of the molecular surface for horse deoxyhaemoglobin.

into 3, and comes with predefined functions for gripping and even approximating the grip strength [53]. Leap Motion released a beta version of a VR mount (Figure 11.2) in 2014 which allowed developers to attach Leap Motion to VR HMD's such as Oculus Rift.

Figure 11.2: Oculus Rift equipped with Leap Motion VR. (Source: `http://simulatortrends.com/wp-content/uploads/2014/10/Leap-Motion-Oculus-Mount.png`)

The detailed skeletal tracking model and its API could prove useful when implementing molecular sculpting (Section 11.2) where the support for gripping and pinching objects with precision is key.

11.4.2 Eye Tracking

Adding eye tracking algorithms to the HMD would allow users to perform interactions without external sensors. This would allow users to select and move objects in the VR scene through pupil movements, thus freeing up their hands.

FOVE

A VR HMD including eye tracking, developed by FOVE, is estimated to be released in the spring of 2016 [54]. The release includes a developer kit with Unity integration, much like Oculus Rift's DK 2. By combining the view of both pupils not only the direction but the point a user is looking at can be calculated in 3-D space, thus allowing interactions in the VR environment. FOVE claims that the increased speed and accuracy achieved through the eye interaction helps reducing the motion sickness and nausea caused by a regular HMD [54].

While their main market is gaming, applications in Life Science has started to emerge. One example is the very cool Eye Play the Piano project[6], developed by FOVE in collaboration with the University of Tsukuba's Special Needs Education School for the Physically Challenged, which gives people with disabilities the opportunity to play the piano only using eye movements.

11.5 Mixed Reality

One of the issues with the Oculus Rift HMD is that it obscures the wearers view (since they are not transparent). While adding to the VR sensation it could cause users to lose track of which direction they were facing and sometimes even their location. In mixed reality the HMD is transparent and the virtual objects are drawn on top of a users regular view. Utilizing this approach would allow users to interact with models in VR while still being able to navigate the room as usual. This would improve the HCI with sensors where the performance relies on the users positioning, such as Kinect.

11.5.1 HoloLens

Microsoft is due to release a pair of mixed reality glasses called HoloLens, together with the release of Windows 10, in the summer of 2015 [55]. HoloLens uses holograms to draw 3-D objects into the users view (Figure 11.3). Molecular 3-D models could for example be rendered on the users desk. It would also allow for collaborations where multiple users, all wearing HoloLens, interacted with the same model.

[6] http://eyeplaythepiano.com/en/

Figure 11.3: Mixed reality with HoloeLens. (Source: `https://cdn2.vox-cdn.com/thumbor/aotdquRv2ZmMp1l-NTU_HrHcQtA=/cdn0.vox-cdn.com/uploads/chorus_asset/file/3327268/a67d3d33-e1e5-4cf7-bf3d-dbe1befc8d8c.0.jpg`)

Bibliography

[1] R. Zone, *3-D Revolution: The History of Modern Stereoscopic Cinema*. Cheshire, CT, USA: University Press of Kentucky, 2012.

[2] P. Marks, "An extra dimension of realism for games arcades," *New Scientist*, vol. 130, p. 23, 1991.

[3] K. Fowle, "A look back at the doomed virtual reality boom of the 90s," *Kill Screen*, vol. 1, 2015.

[4] Sony Computer Entertainment Europe, "Project morpheus discover the future of virtual reality on playstation 4 - launching in 2016." `https://www.playstation.com/en-gb/explore/ps4/features/project-morpheus/`.

[5] HTC Corporation, "Htc vive." `http://www.htcvr.com/`.

[6] F. Riddell and M. Robinson, "J. H. Van't Hoff and J. A. Le Bel - Their Historical Context," *Pergamon Press*, vol. 30, pp. 2001–2007, 1974.

[7] J. Kendrew, G. Bodo, H. Dintzis, R. Parrish, H. Wyckoff, and D. Phillips, "A three dimensional model of the myoglobin molecule obtained by x-ray analysis," *Nature*, vol. 181, pp. 662–666, 1958.

[8] J. Watson and F. Crick, "Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid," *Nature*, vol. 171, p. 737–738, 1953.

[9] pymol.org, "Pymol." `https://www.pymol.org/pymol`.

[10] OpenEye, "Vida." `http://www.eyesopen.com/vida`.

[11] E. Pettersen, T. Goddard, C. Huang, G. Couch, D. Greenblatt, E. Meng, and T. Ferrin, "Ucsf chimera - a visualization system for exploratory research analysis," *J Comput Chem 2004*, vol. 25, pp. 1605–1612, 2004.

[12] Astex Therapeutics Ltd, "Astexviewer." http://openastexviewer.net/web/.

[13] Jmol, "Jmol: an open-source java viewer for chemical structures in 3d." http://www.jmol.org/.

[14] E. Martz, "Mdl chime." http://www.umass.edu/microbio/chime/getchime.htm.

[15] Wikipedia.org, "List of molecular graphics systems." http://en.wikipedia.org/wiki/List_of_molecular_graphics_systems.

[16] S. Volbracht, G. Domik, K. Shahrbabaki, and G. Fels, "An experimental comparison of 3d display modes," *IEEE Visualization*, vol. 96, pp. 8–11, 1996.

[17] L. Hongjian, L. Kwong-Sak, N. Takanori, and W. Man-Hon, "iview: an interactive webgl visualizer for protein-ligand complex," *BMC Bioinformatics*, vol. 15, 2014.

[18] M. Sakuta, "Oculuscrystal." https://github.com/msakuta/OculusCrystal.

[19] B. Myers, "A brief history of human computer interaction technology," *ACM interactions*, vol. 5, pp. 44–54, 1998.

[20] L. Lamberti and F. Camastra, "Real-time hand gesture recognition using a color glove," *Lecture Notes in Computer Science*, vol. 6978, pp. 365–373, 2011.

[21] C. Preetham, G. Ramakrishnan, S. Kumar, A. Tamse, and N. Krishnapura, "Hand talk-implementation of a gesture recognizing glove," *India Educators' Conference (TIIEC), 2013 Texas Instruments*, pp. 328–331, 2013.

[22] M. Alnowami, A. Khan, A. Morfeq, N. Alothmany, and E. Hafez, "Feasibility study of markerless gait tracking using kinect," *Life Science Journal*, vol. 11, pp. 514–523, 2014.

[23] B. Galna, D. Jackson, G. Schofield, R. McNaney, M. Webster, G. Barry, D. Mhiripiri, M. Balaam, P. Olivier, and L. Rochester, "Retraining function in people with parkinson's disease using the microsoft kinect: game design and pilot testing," *Journal of NeuroEngineering and Rehabilitation*, vol. 11, 2014.

[24] Z. Ren, J. Meng, J. Yuan, and Z. Zhang, "Robust hand gesture recognition with kinect sensor," *IEEE Transactions on Multimedia*, vol. 15, pp. 1110–1120, 2013.

[25] D. Norman, J. Miller, and A. Henderson, "What you see, some of what's in the future, and how we go about doing it," *CHI '95 Proceedings*, 1995.

[26] J. Nielsen and D. Norman, "The definition of user experience." http://www.nngroup.com/articles/definition-user-experience/.

[27] Christian Rohrer, "When to use which user-experience research methods." http://www.nngroup.com/articles/which-ux-research-methods/.

[28] W. Stokoe, *Sign Language Structure: An Outline of the Visual Communication Systems of the American Deaf*. University of Buffalo Press, 1960.

[29] Microsoft Corporation, "Jointtype enumeration." https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx.

[30] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, *Real-Time Human Pose Recognition in Parts from Single Depth Images*, vol. 56. Communications of the ACM, 2013.

[31] Microsoft Corporation, "Jointtrackingstate enumeration." https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtrackingstate.aspx.

[32] Ben Lower, "Custom gestures end to end with kinect and visual gesture builder." http://channel9.msdn.com/Blogs/k4wdev/Custom-Gestures-End-to-End-with-Kinect-and-Visual-Gesture-Builder.

[33] P. DeGrace and L. Hulet Stahl, *Wicked Problems, Righteous Solutions: A Catalogue of Modern Software Engineering Paradigms*. Yourdon Press, 1990.

[34] D. Parnas and P. Clements, "A rational design process: How and why to fake it," *IEEE Transactions on Software Engineering*, vol. 12, 1986.

[35] P. Eeles and P. Cripps, *The Process of Software Architecting*. Addison-Wesley Professional, 2010.

[36] D. Rempel, M. Camilleri, and D. Lee, "The design of hand gestures for human-computer interaction: Lessons from sign language interpreters," *International Journal of Human Computer Studies*, vol. 72, pp. 728–735, 2014.

[37] Jakob Nielsen, "The use and misuse of focus groups." http://www.nngroup.com/articles/focus-groups/.

[38] T. Tullis and B. Albert, *Measuring the user experience*. Burlington, MA, USA: Morgan Kaufmann, 2008.

[39] Oculus VR, "Oculus rift sdk and runtime." https://developer.oculus.com/downloads/.

[40] Microsoft Corporation, "Kinect for windows sdk 2.0." http://www.microsoft.com/en-us/download/details.aspx?id=44561.

[41] unity3d.com, "Unity 5." http://unity3d.com/get-unity/download?ref=personal.

[42] Microsoft Corporation, *Human Interface Guidelines (HIG)*. http://go.microsoft.com/fwlink/?LinkID=403900&clcid=0x409.

[43] biochem_fan, "Glmol." http://webglmol.sourceforge.jp/index-en.html.

[44] A. Dalke, E. Backe, H. Van De Waterbeemd, and S. Boyer, "C-lab: a web tool for physical property model calculations." http://dalkescientific.com/writings/CLab-EuroQSAR2008.pdf.

[45] J. Schwarz, C. Marais, T. Leyvand, S. Hudson, and J. Mankoff, "Combining body pose, gaze and motion to determine intention to interact in vision-based interfaces," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 3443–3452.

[46] N. Bryant, "Kinect for windows unity pro." https://github.com/Dibbin/Kinected-Puzzle/tree/master/KinectForWindows_UnityPro_2.0.1410.

[47] A. Bondi, "van der waals volumes and radii," *Journal of Physical Chemistry*, vol. 68, 1964.

[48] StarManta, "Tuberenderer." http://wiki.unity3d.com/index.php?title=TubeRenderer.

[49] G. Wiggins, "Seven keys to effective feedback," *Educational Leadership*, vol. 70, pp. 10–16, 2012.

[50] N. Robbins and R. Heiberger, "Plotting likert and other rating scales," *JSM*, pp. 1058–1066, 2011.

[51] Rivera Gartner, J. and van der Meulen Gartner, R., "Gartner's 2014 hype cycle for emerging technologies maps the journey to digital business." http://www.gartner.com/newsroom/id/2819918.

[52] Oculus VR, "First look at the rift, shipping q1 2016." https://www.oculus.com/blog/first-look-at-the-rift-shipping-q1-2016/.

[53] Leap Motion Ltd, *Introducing the Skeletal Tracking Model*. https://developer.leapmotion.com/documentation/javascript/devguide/Intro_Skeleton_API.html.

[54] Fove-inc, "Fove - the world's first eye tracking virtual reality headset." http://www.getfove.com/.

[55] Microsoft Corporation, "Microsoft hololens." https://www.microsoft.com/microsoft-hololens/en-us.

Appendices

Appendix A
About this Project

The following environments and tools were used during development:

- Computer: Lenovo ThinkCentre M53p

  Operating system: Windows 8.1 Enterprise

  Processor: Intel(R) Core(TM) i5-4570 CPU @ 3.20 GHz

  RAM: 8.0 GB

  System type: 64-bit Operating System, x64-based processor

- VR-glasses: Oculus Rift DK 2, `https://www.oculus.com/dk2/`

  Oculus Positional Tracker DK2

  Runtime: Oculus Runtime for Windows 0.4.4-beta, `https://developer.oculus.com/downloads/`

  SDK: Oculus SDK for Windows 0.4.4-beta, `https://developer.oculus.com/downloads/`

  Plug-in: Unity 4 Integration 0.5.0.1-beta, `https://developer.oculus.com/downloads/`

- Sensor: Microsoft Kinect v2 for Windows, `http://www.microsoft.com/en-us/kinectforwindows/purchase/`

  Runtime: Kinect for Windows Runtime 2.0, `http://www.microsoft.com/en-us/download/details.aspx?id=44559`

  SDK: Kinect for Windows SDK 2.0, `http://www.microsoft.com/en-us/download/details.aspx?id=44561`

  Depth Recorder: Kinect Studio v2.0

Gesture Builder: Visual Gesture Builder Viewer - Preview

Editor: Microsoft Visual Studio Express 2013 for Windows Desktop, https://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

- Game Engine: Unity 5.0.0, http://unity3-D.com/get-unity/download?ref=personal

  Plug-in: Kinect For Windows Unity Pro, https://GitHub.com/Dibbin/Kinected-Puzzle/tree/master/KinectForWindows_UnityPro_2.0.1410

  Editor: MonoDevelop 4.0.1, http://www.monodevelop.com/

- Molecular Visualisation Systems

  PyMOL 1.5.0.4, https://www.pymol.org/pymol

  GLmol 0.47, http://webGLmol.sourceforge.jp/index-en.html

  C-Lab, (Not available for download)

Appendix B
Code

The code used in all stages of the project:

- HandMap, `https://GitHub.com/Magnusnorrby/HandMap`
- MolyRift, `https://GitHub.com/Magnusnorrby/MolyRift`
- MyRemote, `https://GitHub.com/Magnusnorrby/exjobb`
- Code from Kinect for Windows SDK 2.0, `http://www.microsoft.com/en-us/download/details.aspx?id=44561`

  Audio Basics-WPF

  Body Basics-D2D

  Body Basics-WPF

  Color Basics-WPF

  Depth Basics-WPF

- Code from Oculus SDK for Windows 0.4.4-beta, `https://developer.oculus.com/downloads/`

  OculusTinyRoom

- TubeRenderer.cs, `http://wiki.unity3d.com/index.php?title=TubeRenderer`

Appendix C
Keyboard Commands

Table C.1: Keyboard Commands in MolecularRift.

Action	**Key**
Move Forward	W
Move Left	A
Move Right	D
Move Back	S
Move Up	T
Move Down	G
Zoom	Mouse Wheel
Toggle Menu	N
Toggle Ribbons	R
Center View	V
New Focus Target	P
Display Details	Space

Appendix D
Iteration One

D.1 Gesture Guide

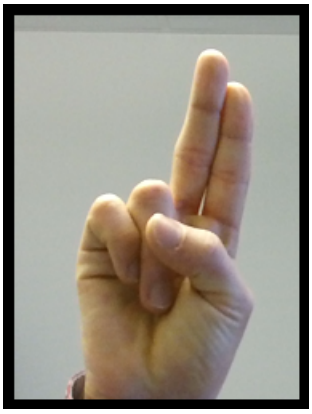
Gestures

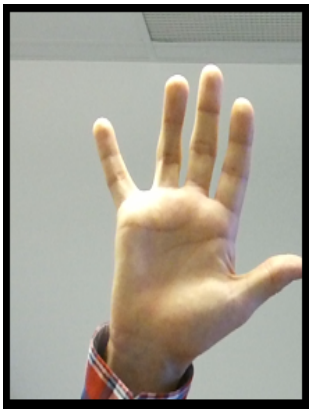
Right Hand Closed

Right Hand Lasso

Right Hand Open

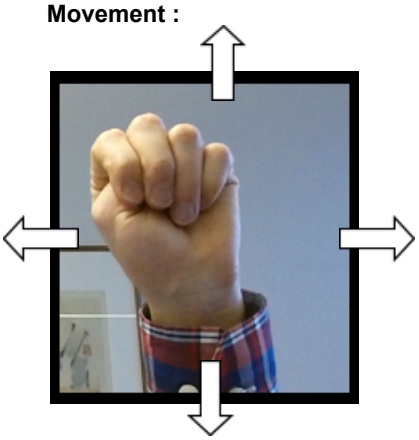
Left Hand Closed


Left Hand Lasso


Left Hand Open

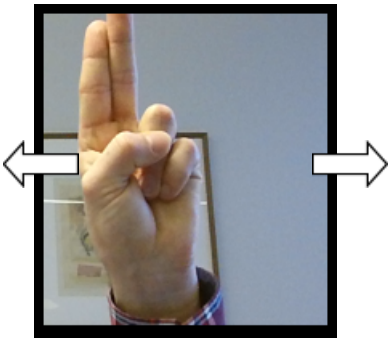
View Mode:

Movement :



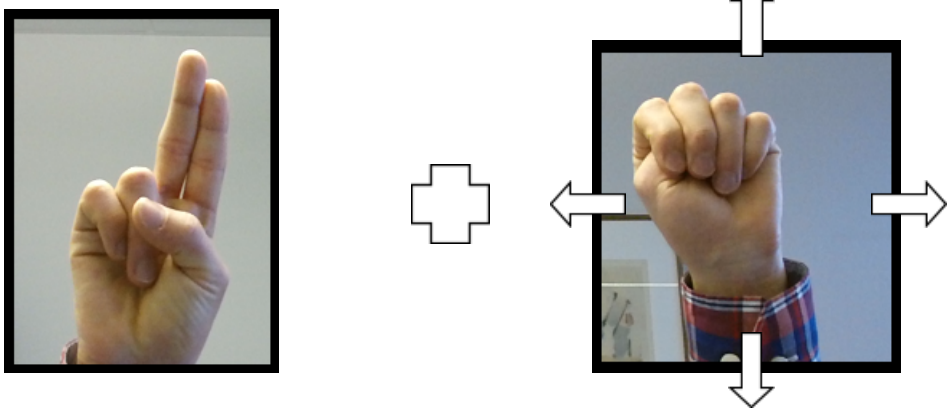
Right Hand

Zoom:



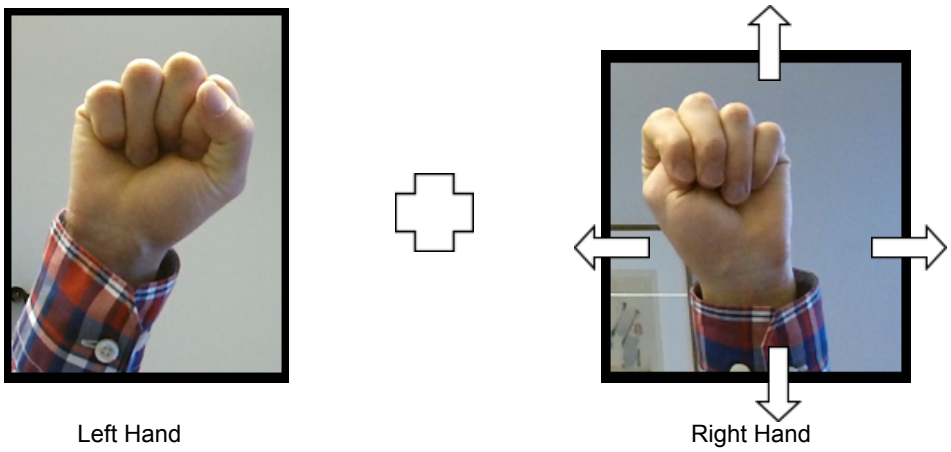
Right Hand

Rotate:



Left Hand Right Hand

Translate:



Left Hand Right Hand

Menu mode:

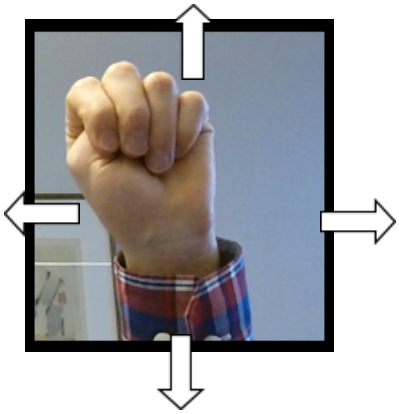
Toggle menu on/off:

Perform in sequence above head

Right Hand

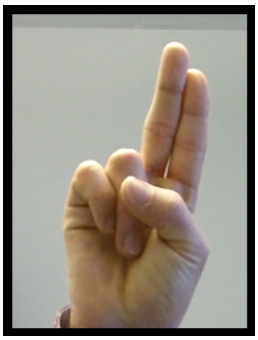
Right Hand

Mouse movement:



Right Hand

Mouse Click:



Left Hand

D.2 Task

Task Iteration One

April 2015

1. Read in protein (with ligand)
2. Move towards the binding site (ligand)
3. Zoom in on the binding site
4. Zoom back out
5. Translate the protein away from you
6. Rotate the protein
7. Toggle the menu on
8. Turn ribbons off using the in-game mouse
9. Turn alpha trace on
10. Deactivate the menu
11. End

D.3 Questionnaire

Questions Iteration One

April 2015

1. Background Questions
   (a) Have you previously tried Microsoft Kinect?

   (b) Have you previously tried Oculus Rift or similar VR environments?

   (c) What is your function at AstraZeneca?

2. Evaluation Questions
   (a) Which gesture was the easiest to use?

   (b) What made the gesture easy to use?

   (c) Which gesture was the hardest to use?

   (d) What made the gesture hard to use?

   (e) Was the movement speed (sensitivity) satisfying?

1

(f) Would you have preferred a different way to activate the menu? (Feel free to add suggestions)

(g) What function would you like to add to the application?

(h) What was your overall experience with MoleRift?

(i) How did you find MoleRift compared to standard tools such as VIDA and PyMol?

3. Rate Gestures according to usability (1 easiest - 10 hardest)

(a) Movement: 1 2 3 4 5 6 7 8 9 10

(b) Zoom: 1 2 3 4 5 6 7 8 9 10

(c) Rotation: 1 2 3 4 5 6 7 8 9 10

(d) Translation: 1 2 3 4 5 6 7 8 9 10

(e) Toggle Menu: 1 2 3 4 5 6 7 8 9 10

(f) Mouse Movement: 1 2 3 4 5 6 7 8 9 10

(g) Mouse Click: 1 2 3 4 5 6 7 8 9 10

Thank you for your participation!

2

D.4 Questionnaire Answers

1. Background Questions

   (a) Have you previously tried Microsoft Kinect?

   User1: No.
   User2: Yes, but very little.
   User3: No.
   User4: No.
   User5: Yes.
   User6: No.
   User7: No.
   User8: No.
   User9: No.

   (b) Have you previously tried Oculus Rift or similar VR environments?

   User1: No.
   User2: No.
   User3: No.
   User4: No.
   User5: No.
   User6: No.
   User7: No.
   User8: No.
   User9: No.

   (c) What is your function at AstraZeneca?

   User1: Computational Chemist.
   User2: MedChem/CompChem CVMD (Modeller).
   User3: TL MedChem.
   User4: PS, Chem reactivity.
   User5: Computational Chemist.
   User6: Computational Chemist.
   User7: Team leader.
   User8: Architect RDI.
   User9: TL CC CVMD.

2. Evaluation Questions

(a) Which gesture was the easiest to use?

User1: Right hand closed, natural, logic moves.
User2: Zoom in/out.
User3: Zoom.
User4: Menu.
User5: Fist.
User6: Forward and backwards.
User7: Right fist.
User8: Rotate.
User9: Movement.

(b) What made the gesture easy to use?

User1: Simple to coordinate.
User2: Easy to remember and different from others.
User3: Little use of arm length.
User4: -
User5: Easy to perform and do.
User6: Good speed.
User7: Intuitive, one hand.
User8: Natural.
User9: Responds quickly.

(c) Which gesture was the hardest to use?

User1: Menu open/close.
User2: Toggle in-game menu.
User3: Closed hand for long time.
User4: Move.
User5: Menu.
User6: Rotation.
User7: See menu.
User8: Zoom.
User9: Zoom.

(d) What made the gesture hard to use?

User1: Difficult to get the right movement in the hand.
User2: Couldn't achieve it.
User3: Tired.
User4: -
User5: Hard to recognize.
User6: Too fast and hard to find the right orientation.
User7: Had to raise my hand and have the thumb inside.
User8: Had to stop in mid-air.
User9: Continue to zoom to where you would like to be.

(e) Was the movement speed (sensitivity) satisfying?

User1: Yes but the movement speed and turning speed didn't match.
User2: Zooming not - to little happened. Rotation not - too much happened.
User3: Would like to have seen more move for less arm movement.
User4: To bumpy.
User5: Yes.
User6: Forward and backwards was good everything else too fast.
User7: Yes except for rotation.
User8: Yes.
User9: Yes, to speedy.

(f) Would you have preferred a different way to activate the menu? (Feel free to add suggestions)

User1: Menu comes up on clap, close on 2 claps.
User2: Yes, another gesture.
User3: No.
User4: -
User5: Yes.
User6: Ok.
User7: Yes.
User8: No.
User9: Would be great to have it open at all time.

(g) What function would you like to add to the application?

User1: Select atoms and amino acids. Center on mass, on selected atoms.

User2: Display surfaces.

User3: Choose axis for rotation.

User4: -

User5: Surface representation of binding site. Shortcuts on keyboard to facilitate navigation.

User6: Choose center atom.

User7: Measure bond distances and add atoms.

User8: Surfaces.

User9: Point to center of rotation.

(h) What was your overall experience with MolecularRift?

User1: Very cool.

User2: It is cool!

User3: Cool stuff.

User4: -

User5: Great. Controlling needs more training.

User6: Fun but hard to steer.

User7: Great!!

User8: OK.

User9: Cool!

(i) How did you find MolecularRift compared to standard tools such as VIDA and PyMol?

User1: Very promising. Need more practice to steer good but much better to have up and down view when you turn your head.

User2: It is cooler.

User3: Likely a superior tool in meetings.

User4: -

User5: 3D much better. Other tools more flexible.

User6: Better stereo less control.

User7: To early to tell.

User8: No clue.

User9: Only thing is resolution that could be improved.

Appendix E
Iteration Two

E.1 Questionnaire

Science Retreat Questionnaire (Iteration two)

Gesture	**Very Easy**	**Easy**	**Medium**	**Hard**	**Very Hard**
Movement	☐	☐	☐	☐	☐
Rotation	☐	☐	☐	☐	☐
Translation	☐	☐	☐	☐	☐
Zoom	☐	☐	☐	☐	☐

Appendix F
Iteration Three

F.1 Questionnaire

Questionnaire - Iteration Three

Gesture	Very Easy	Easy	Medium	Hard	Very Hard
Movement	☐	☐	☐	☐	☐
Rotation	☐	☐	☐	☐	☐
Translation	☐	☐	☐	☐	☐
Zoom	☐	☐	☐	☐	☐
Toggle Menu	☐	☐	☐	☐	☐

VR = Virtual Reality

Statement	Strongly Agree	Agree	Neutral	Disagree	Strongly Disagree
I enjoyed using MolecularRift	☐	☐	☐	☐	☐
VR tools could prove useful in drug discovery	☐	☐	☐	☐	☐
The VR goggles were disturbing	☐	☐	☐	☐	☐
I experienced motion sickness	☐	☐	☐	☐	☐