School of Computing

FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Molecular Dynamics Visualisation in VR

Govind Nangapuram Venkatesh

Submitted in accordance with the requirements for the degree of
MSc High Performance Graphics and Games engineering

2019/2020

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Report (Background research, System design / architecture, ...)	*Report*	*Supervisor, assessor (27/08/20)*
Software packages, Source Code and Performance reports	*GitHub repo in Appendix of report*	*Supervisor, assessor (27/08/20)*

Type of Project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

[illegible]

Summary

In the field of Molecular Dynamics, experiments are carried out at the molecular level to examine the interaction between atoms and molecules. Data is collected from complex interactions between molecules, which are simulated for days, before being visualised to understand and draw meaningful conclusions from it. An issue that arises with visualising such data on standard desktop monitors is the lack of spatial awareness and general clutter created when rendering such complex molecules.

This project aims to create a new type of visualiser capable of rendering these complex molecules in virtual reality, to help solve the aforementioned issues. Such a system will be implemented using the Unreal Engine 4 game engine, chosen for its performance and flexibility when developing for multiple target platforms and virtual reality kits.

Acknowledgements

I would like to thank Dr He Wang for his invaluable guidance and understanding, especially during these difficult times. I would also like to thank my assessor Dr Hamish Carr for his valuable feedback during the progress meeting. A further thanks to Dr Antreas Kalli and Kyle Le Hurray for their enthusiasm, patience and willingness to answer my questions. Finally, thank you to all who participated in my usability tests.

Contents

Summary iii
Acknowledgements iv
Contents v
1 Introduction .... - 1 -
1.1 Aim .... - 1 -
1.1.1 Context .... - 1 -
1.1.2 Problem statement .... - 1 -
1.1.3 Possible solution .... - 3 -
1.1.4 How to demonstrate the quality of the solution .... - 3 -
1.2 Objectives .... - 3 -
1.3 Deliverables .... - 4 -
1.4 Ethical, legal and social issues .... - 5 -
1.5 Project schedule .... - 5 -
1.5.1 Methodology .... - 5 -
1.5.2 Tasks, milestones and timeline .... - 5 -
2 Background Research .... - 7 -
2.1 Chemistry Basics .... - 7 -
2.1.1 What is a protein? .... - 7 -
2.2 Quick Background on Molecular Dynamics (MD) .... - 10 -
2.2.1 What is Molecular Dynamics Visualisation? .... - 11 -
2.2.2 Van der Waals (VDW) radii .... - 11 -
2.2.3 Bonds and Atoms .... - 11 -
2.2.4 Tube .... - 11 -
2.2.5 New Cartoon .... - 11 -
2.3 Game Engines .... - 11 -
2.3.1 Unreal Engine .... - 12 -
2.3.2 Unity .... - 12 -
2.3.3 Choice of game engine .... - 12 -
2.3.4 UE4 Basic Concepts and Terminology .... - 12 -
2.4 Virtual Reality .... - 13 -
2.4.1 Oculus Rift technology .... - 14 -

2.5 Currently used products ....- 15 -

2.5.1 Visual Molecular Dynamics (VMD)................................- 15 -

2.5.2 ChimeraX ....- 15 -

2.6 Summary...- 16 -

3 System Design ....- 17 -

3.1 System Requirements ....- 17 -

3.2 File Formats ....- 20 -

3.3 Secondary Structure Computation using Structural Identification program (STRIDE) ....- 24 -

3.4 UE4 Blueprints and Materials ....- 26 -

3.5 UE4 mesh rendering pipeline ....- 28 -

3.6 Designing for Virtual Reality ....- 31 -

3.6.1 Best practices...- 31 -

3.6.2 Unreal Engine integration...- 33 -

3.7 System Architecture ....- 34 -

3.7.1 Readers ....- 34 -

3.7.2 Model View Controller (MVC)...- 35 -

4 Software Implementation..- 36 -

4.1 User Interface...- 36 -

4.2 Van der Waals (VDW)..- 38 -

4.3 Bonds & Atoms (CPK)...- 40 -

4.4 Tube ....- 40 -

4.5 New Cartoon ....- 41 -

4.6 Meshes...- 42 -

4.6.1 Automatic LOD generation ....- 43 -

4.7 Algorithms ....- 44 -

4.8 Splines ....- 46 -

4.9 Material ....- 47 -

4.10 Mouse and Keyboard controls...- 47 -

4.11 Automation Testing ....- 49 -

4.12 Results ....- 50 -

5 Testing ....- 51 -

5.1 Usability testing ....- 51 -

5.1.1 Methodology..- 51 -

5.1.2 Results ....- 51 -

5.2 Performance testing ....- 53 -
5.2.1Methodology....- 53 -
5.2.2Results ....- 55 -
5.2.3Improvements ....- 57 -
6 Conclusion....- 58 -
6.1 Project Evaluation ....- 58-
6.2 Experience using UE4....- 59 -
6.3 Challenges Faced ....- 59 -
6.4 Future improvements ....- 59 -
6.5 Summary....- 60 -
7 References....- 61 -
Appendix A External Materials....- 66 -
A.1 Deliverables ....- 66 -
A.1.1 Unreal Engine 4 ....- 66 -
A.1.2 Automation Project....- 66 -
A.1.3 Audio Transcripts for Usability Sessions....- 66 -
A.1.4 RenderDoc....- 66 -
Appendix B Ethical Issues Addressed ....- 67 -
B.1 Consent Forms ....- 67 -
Appendix C Additional Materials ....- 70 -
C.1Van Der Waals radii....- 70 -
C.2 User Testing interview script....- 71 -
C.3 Performance report break down ....- 72 -
C.4 RenderDoc BasePass draw calls snippet....- 74 -
C.5 System Architecture Diagram ....- 80 -

1 Introduction

1.1 Aim

The aim of this project is to produce a Molecular Dynamics visualisation tool in Virtual Reality. The main goal is to examine popular tools currently used, to define a set of user and system requirements and to propose & develop a software package to satisfy these requirements.

1.1.1 Context

Molecular Dynamics is a field where computer simulations or physical experiments are carried out to examine the interactions between atoms and molecules over time (1).

Pharmaceutical industries and others involved in drug development use molecular simulations in order to understand how a new drug interacts within the body. An example of this would be to see how drug molecules bind to receptors (proteins) and what effects they have on the receptors' structure. Some other applications involve understanding how proteins fold, thereby understanding how they achieve their 3-dimensional structure; this, in turn, determines their biological function (2) .

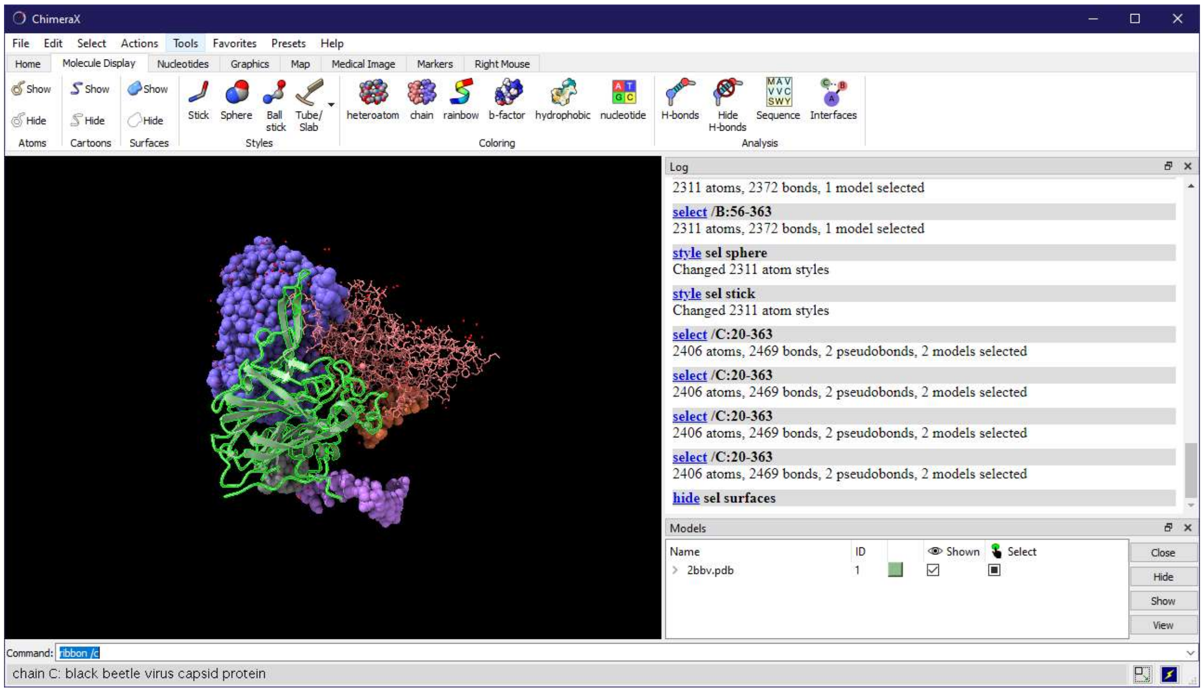
Some popular tools to do these simulations are NAMD and GROMACS. Most of these simulation software packages allow the simulation to be run on a cluster of PCs and on GPU to achieve high parallelisation, thereby speeding it up. Even with this, it is not possible to visualise all simulations in real-time. As a result, data is collected for each timestep and later used in a specialised tool for visualisation (2).

These visualisation tools are then used to render the 3D structure of the molecule and show how it changes as the simulation progressed.

Some popular visualisation tools used are VMD (3), Pymol (4), and ChimeraX, among many others. All of these tools are desktop-based 3D environments, like those seen in most video games and feature keyboard and mouse controls to navigate the geometry. They feature very powerful visualisation options to allow as much meaning as possible to be derived from the simulation.

1.1.2 Problem statement

Shown below are two of the most popular tools for visualising molecular dynamics simulations: ChimeraX and VMD. They have been shown rendering a fairly simple protein.



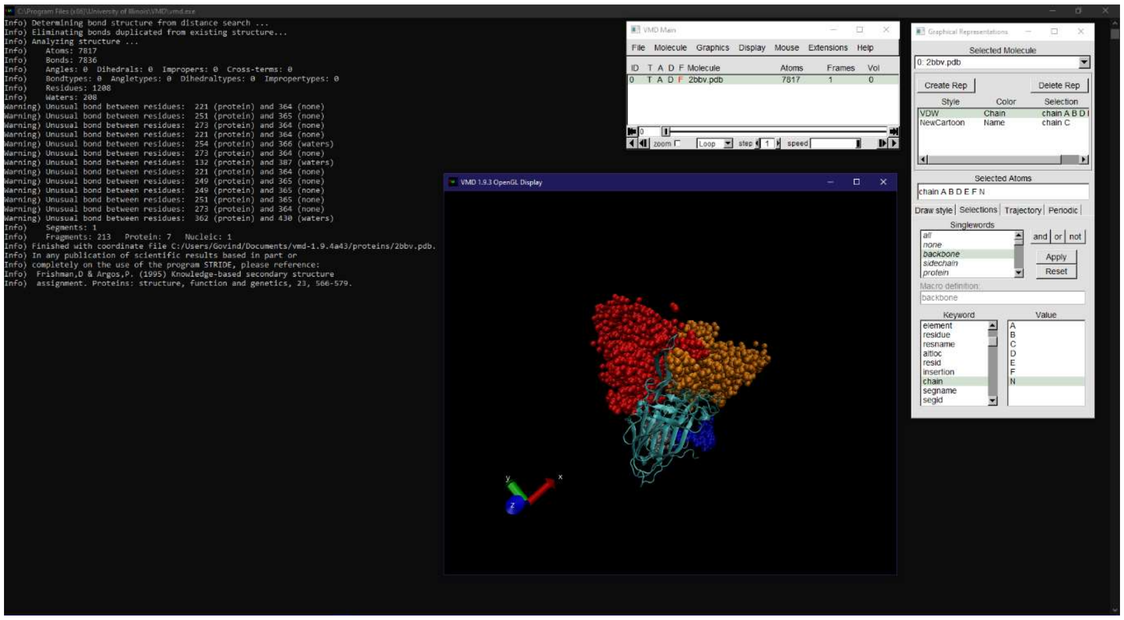
```
2311 atoms, 2372 bonds, 1 model selected
select /B:56-363
2311 atoms, 2372 bonds, 1 model selected
style sel sphere
Changed 2311 atom styles
style sel stick
Changed 2311 atom styles
select /C:20-363
2406 atoms, 2469 bonds, 2 pseudobonds, 2 models selected
select /C:20-363
2406 atoms, 2469 bonds, 2 pseudobonds, 2 models selected
select /C:20-363
2406 atoms, 2469 bonds, 2 pseudobonds, 2 models selected
select /C:20-363
2406 atoms, 2469 bonds, 2 pseudobonds, 2 models selected
hide sel surfaces
```

Command: ribbon /c

chain C: black beetle virus capsid protein

ChimeraX
File Edit Select Actions Tools Favorites Presets Help
Home Molecule Display Nucleotides Graphics Map Medical Image Markers Right Mouse
Show Hide Atoms
Show Hide Cartoons
Show Hide Surfaces
Stick Sphere Ball stick Tube/ Slab Styles
heteroatom chain rainbow b-factor hydrophobic nucleotide Coloring
H-bonds Hide H-bonds Sequence Interfaces Analysis
Log
Models
Name ID Shown Select
2bbv.pdb 1
Close Hide Show View

Figure 1 ChimeraX user interface rendering protein 2bbv



```
Info) Determining bond structure from distance search ...
Info) Eliminating bonds duplicated from existing structure...
Info) Analyzing structure ...
Info)    Atoms: 7817
Info)    Bonds: 7836
Info)    Angles: 0  Dihedrals: 0  Impropers: 0  Cross-terms: 0
Info)    Bondtypes: 0  Angletypes: 0  Dihedraltypes: 0  Impropertypes: 0
Info)    Residues: 1208
Info)    Waters: 208
Warning) Unusual bond between residues:  221 (protein) and 364 (none)
Warning) Unusual bond between residues:  251 (protein) and 365 (none)
Warning) Unusual bond between residues:  273 (protein) and 364 (none)
Warning) Unusual bond between residues:  221 (protein) and 364 (none)
Warning) Unusual bond between residues:  254 (protein) and 366 (waters)
Warning) Unusual bond between residues:  273 (protein) and 364 (none)
Warning) Unusual bond between residues:  132 (protein) and 387 (waters)
Warning) Unusual bond between residues:  221 (protein) and 364 (none)
Warning) Unusual bond between residues:  249 (protein) and 365 (none)
Warning) Unusual bond between residues:  249 (protein) and 365 (none)
Warning) Unusual bond between residues:  251 (protein) and 365 (none)
Warning) Unusual bond between residues:  273 (protein) and 364 (none)
Warning) Unusual bond between residues:  362 (protein) and 430 (waters)
Info)    Segments: 1
Info)    Fragments: 213   Protein: 7   Nucleic: 1
Info) Finished with coordinate file C:/Users/Govind/Documents/vmd-1.9.4a43/proteins/2bbv.pdb.
Info) In any publication of scientific results based in part or
Info) completely on the use of the program STRIDE, please reference:
Info)  Frishman,D & Argos,P. (1995) Knowledge-based secondary structure
Info)  assignment. Proteins: structure, function and genetics, 23, 566-579.
```

VMD Main
File Molecule Graphics Display Mouse Extensions Help
ID T A D F Molecule Atoms Frames Vol
0 T A D F 2bbv.pdb 7817 1 0
VMD 1.9.3 OpenGL Display
Graphical Representations
Selected Molecule
0: 2bbv.pdb
Create Rep Delete Rep
Style Color Selection
VDW Chain chain A B D I
NewCartoon Name chain C
Selected Atoms
chain A B D E F N
Draw style Selections Trajectory Periodic
Singlewords
all none backbone sidechain protein
and or not
Apply Reset
Macro definition:
Keyword Value
element residue resname altloc resid insertion chain segname segid
A B C D E F N

Figure 2 VMD user interface rendering protein 2bbv

According to Gino Cortopassi, a scientist in the field of Molecular Biosciences, visualising many atoms of data on a flat 2D screen becomes messy, and by being able to walk through the molecule and view it from many different angles, it becomes easier to observe important molecular interactions (5).

Figure 1 and Figure 2 show the user interfaces of VMD and ChimeraX. Although the molecule being rendered is not exactly the largest and most complex, it gives a sense of how difficult a complex visualisation may be to comprehend.

When navigating complex visualisation, the user has to keep track of where they are and what they are trying to observe. Therefore, it is easy to get lost in the large amounts of data being rendered, and this will increase the amount of time required to understand the visualisation.

Since these visualisation tools are desktop-based, there is a lot of room to experiment with the interactivity of such tools and improve the user experience when viewing complex visualisations. Using new technologies, allowing the users to explore the complex molecular structures in a highly detailed intuitive way would be an ideal solution to this problem (5).

1.1.3 Possible solution

One solution to this problem would be to create a visualisation tool using Virtual Reality (VR).By positioning the user within the scene, VR, will allow for complex molecular structures and simulations to be visualised with ease and relatively little confusion (6) (7), thus overcoming the previously mentioned issue with current desktop-based visualisers.

1.1.4 How to demonstrate the quality of the solution

The benchmark of quality will be a comparison with existing molecular dynamics visualisation tools. Researchers will not be willing to learn a completely new interface, and so the new VR tool will need to be designed according to existing tools. As such, a set of user and system requirements will be produced to match the features of existing tools (such as VMD / Pymol). By identifying the requirements, it will be possible to create a tool that closely fits what already exists. Furthermore, benchmarking the performance and ensuring that the frame rate is consistently above 90 fps will ensure that when it is ported to VR, it will not cause any detrimental side effects to the user experience such as motion sickness.

1.2 Objectives

Before starting the development of the software, a study has to be carried out encompassing the following:

- Exploration into tools used today and the standards within the industry
- Description of software architecture and design choices
    - Class Diagrams and other standard UML diagrams
- User and System requirements

    - Used to specify functionality accurately and track progress
- Background research

This will be followed by the actual development of the software package

- Replication of the visualisation of protein structures as rendered by an existing tool
- Load proteins from a Protein Data Bank (PDB) file to be used in one of the following representations
- Visualisation of protein structure using Van der Waals representation
    - Spheres scaled by Van Der Waals Radii which are pre-specified per element
- Visualisation of protein structure using New Cartoon representation
    - Visualisation of secondary structures using New Cartoon representation which mainly consists of a Tube and a Beta-Sheet and Alpha-Helix structures.
- Animation of the molecular structure from captured dynamic simulation data, read in from an XTC file format
- Porting software to Virtual Reality
- Software testing and evaluation

1.3 Deliverables

1. The MSc Project Report
    a. Background research
    b. Analysis of current tools
    c. User and System requirements of software package based on analysis of current tools
    d. System architecture and design documentation
    e. Details of software implementation
    f. Analysis of virtual reality for molecular dynamics visualisation
2. Software packages
    a. Desktop visualisation - MOLVIZ (delivered in a Github repository)
        i. Visualisation of protein structures
        ii. Visualisation of Secondary Structures
        iii. Visualisation using Van der Waals and NewCartoon representation
        iv. Animation using pre-collected simulation data
    b. Benchmark
    c. Commented source code

d. Nice To Have features (not implemented due to COVID-19)
    i. Virtual reality visualisation (easy to implement feature using Unreal Engine 4 capabilities)

1.4 Ethical, legal and social issues

There are no risks associated with this project as there are no stakeholders. Test data is readily available on the Worldwide Protein Data Bank website (WWPDB). There is a possibility that test users will feel mild discomfort when using the VR headset; they must be made aware of these and should be monitored when conducting tests.

Due to the situation resulting from the Covid-19 pandemic, it was not possible to access the VR lab at the University of Leeds. As a result, due to the inability to test VR functionalities, these features could not be implemented.

1.5 Project schedule

1.5.1 Methodology

An agile methodology was used wherever possible. When creating the system architecture and design, development happened in tandem. A Kanban board (Trello) was used to monitor progress towards the final goal, as defined by the user and system requirements. Initially, a waterfall approach was used to accumulate background knowledge and to plan the larger chunks of time. User testing was conducted as permitted by the Covid-19 situation at the time. Benchmarking was used to measure the success of the project, with a view of porting it to VR.

1.5.2 Tasks, milestones and timeline

A general guideline was produced at the start. It was mostly adhered to over the course of the project.

- Milestone 1: complete background research and requirements 27th April 2020
- Milestone 2: complete system architecture and design 4th May 2020
- Milestone 3: finish implementation of core features 13th July 2020
- Milestone 4: finish first draft of report 27th July 2020
- Milestone 5: finish second draft of report 4th August 2020
- Milestone 6: finalise report 12th August 2020

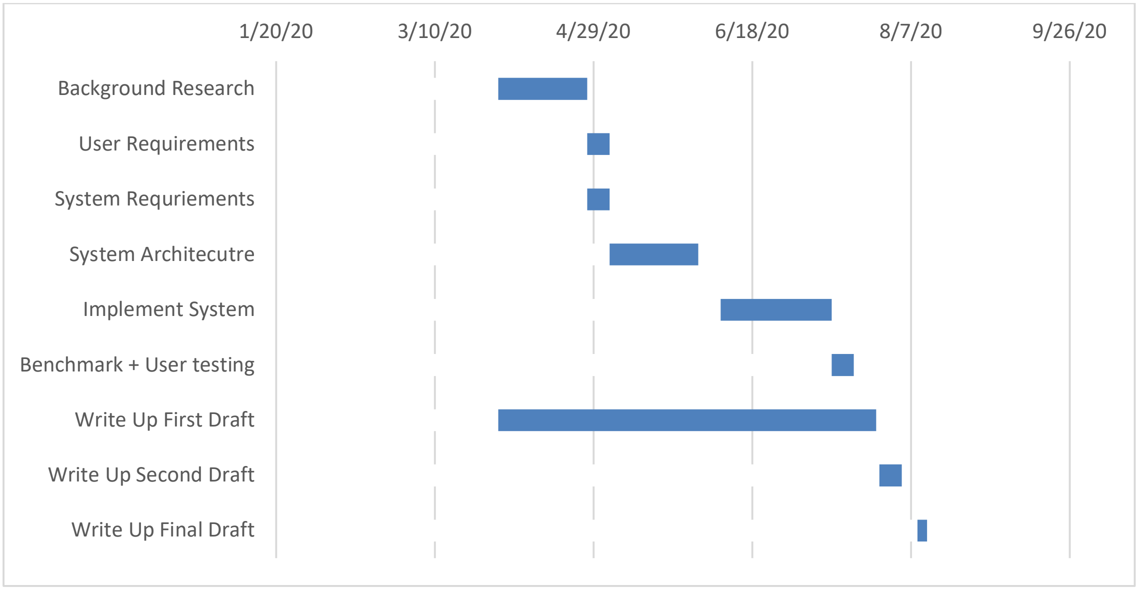


Figure 3 Project plan

2 Background Research

2.1 Chemistry Basics

The main audience for this simulation are biochemists and biophysicists; they need to be able to visualise proteins. In order to visualise a protein, it is necessary to have an understanding of the key terminology used to describe them and their structure.

2.1.1 What is a protein?

A protein is a molecule meaning that it is made of many linked atoms. These atoms are structured into what are called amino acids (also referred to as "residues"), which are molecules by themselves. The protein therefore consists of chains of amino acids that are structured in a specific way by being linked to one another. (8)

There are 20 different kinds of amino acids. Depending on how these 20 amino acids are configured, they make up a protein. These amino acids are bonded by way of a covalent peptide bond. A covalent bond is simply a sharing of electron pairs between two atoms. Therefore, it is common to refer to proteins as polypeptides, meaning many peptides (8).

The amino acids, with their bonds create a specific structure called a protein backbone or, polypeptide backbone. There are also atoms that do not makeup part of the peptide bond and these are referred to as the side chain. Figure 4 shows the structure of a single amino acid (8).

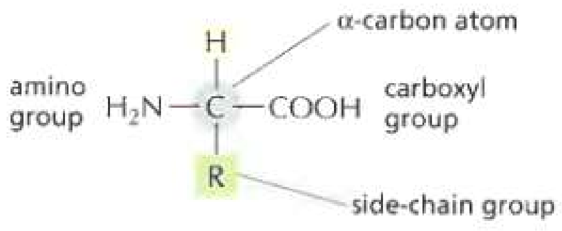


Figure 4 Amino acid structure **(8)**

The structure of an amino acid (Figure 4) contains two groups, an amino group and a carboxyl group. The amino acid will always start and end with these groups in this order. The first amino acid in a chain will have an amino group but no carboxyl group and the last amino acid will have no amino group but end with a carboxyl group. The first and last amino acids are missing one of these groups because each amino acid in between is bonded to the next one. This structure can be seen in Figure 5. (8)

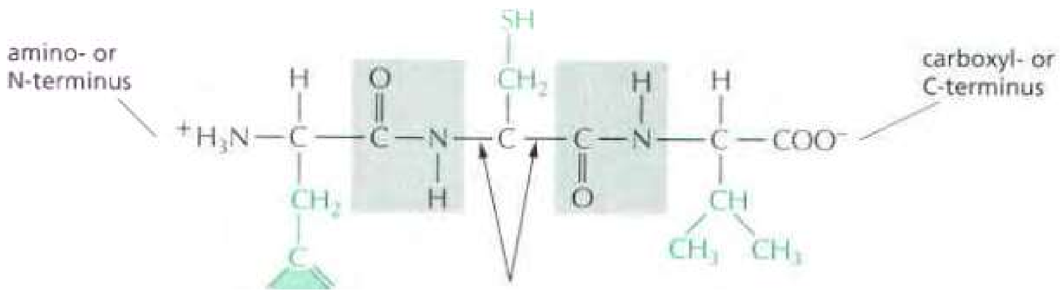


Figure 5 Chain of amino acids **(8)**

The R in Figure 4 represents a side chain, which are atoms that are not part of the backbone. Despite this, they also play a vital role in determining the structure of a protein. The side chains in a protein interact with each other in the following ways: they are charged either negatively or positively, they form weak non-covalent bonds, they may be non-polar (hydrophobic). A non-covalent bond differs from a covalent bond where instead of sharing electrons, it is more related to electromagnetic interactions between the atoms. There are different kinds of non-covalent weak bonds that can occur between the atoms in a protein, these could be hydrogen bonds, electrostatic attractions and Van der Waals attractions. Furthermore, since these are weak bonds, many more are required in order to make a stable shape as opposed to the covalent peptide bonds between the backbone atoms. When these side chains then interact with each other the protein falls into a specific structure, called a conformation (8).

The non-covalent bond called Van der Waals attraction (mentioned above) appears in two contexts; one refers to the radii of atoms that is the minimum space required by that atom, or the closest another atom can approach it. The other context is as above, where the Van der Waals attraction refers to an electromagnetic force felt by atoms close to each other in space that attracts them together. This occurs due to the charge of their electron clouds (8).

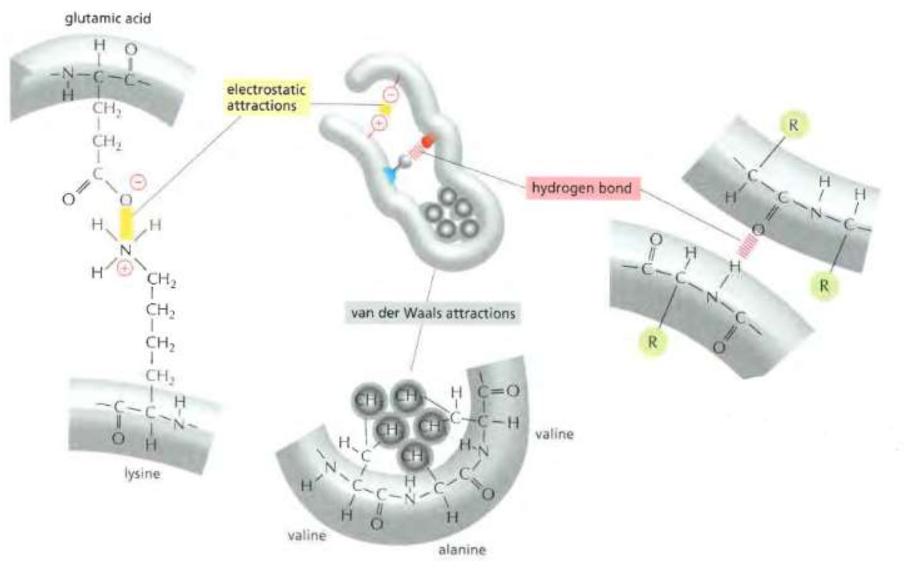


Figure 6 Backbone structure changes due to the properties of side-chain **(8)**
Figure 6 shows the Van der Waals attractions, electrostatic attractions and hydrogen bonds bending the backbone of a protein.

One other important factor of the protein backbone structure is its rigidity/flexibility. There are two important repeating structures found in all proteins, called the secondary structure. These are the Alpha-Helix and Beta-Sheet, which are part of the secondary structure of a protein. The Alpha-Helix is created by a polypeptide chain twisting on itself to form a rigid cylinder and the Beta-Sheet is formed by polypeptide chains that run in the same direction or in opposite directions (in a zig-zagging pattern). Beta-Sheets are found quite abundantly in proteins while Alpha-Helices are less so. A unique property of the Alpha-Helix is that it is extremely rigid. The following image shows how a secondary structure of a protein will look when visualised (8).

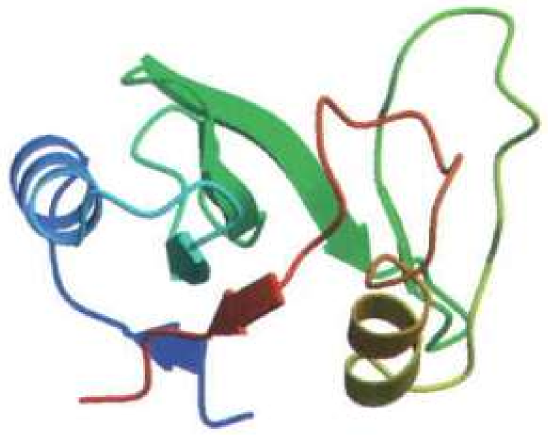


Figure 7 Representing secondary structure information **(8)**
It is common to represent Beta-Sheets by an arrow and Alpha-Helices as flat ribbons.

2.2 Quick Background on Molecular Dynamics (MD)

MD is a way to use computer simulation to conduct experiments based on a theoretical model of interaction or interactions between atoms (1). Many fields make use of these simulations but it is heavily used in the area of Biophysics (2).

Some of the specific applications of such simulation would be observing protein structures and determining their function, such as the structure of Haemoglobin in transporting oxygen. Understanding how a protein maintains its shape and why it is shaped like that helps to understand its biological function. Another application would be to understand the mechanisms that cause proteins to change shape (2).

The experiments described above are conducted in computer simulations over possibly many days. These simulations are extremely complex and involve a very fine time scale. Numerically stable simulation steps are in the range of femtoseconds (fs) (1 femtosecond = 1 x 10^-15 seconds) and recently, due to improvements in computing power, time steps as large as 1 microsecond can be used. Furthermore, changes in the structure of proteins can take milliseconds or longer, and with timesteps being so small and potentially very computationally expensive, the simulation can take a long time to complete (1).

Once this data has been collected, it can be visualised using a special tool specifically made to aid visualisation.

2.2.1 What is Molecular Dynamics Visualisation?

Typically, an MD visualiser is desktop software designed to visualize and animate MD data and will provide a plethora of options to customize the representation of the molecules, to aid the human eye in interpreting the data and understanding the simulation (2).

Some of the most common visualisations available are Van der Waals (VDW) radii, Bonds and Atoms, Tube and New Cartoon.

2.2.2 Van der Waals (VDW) radii

The VDW radii is a simple representation where each element has a specific radius that will be used to render a sphere (9). The radii for the elements can be found in Appendix C.1.

2.2.3 Bonds and Atoms

Bonds and Atoms is also a simple representation. Here the atoms are represented as fixed-size spheres of a certain user-defined radius and bonds between atoms are represented as cylinders (10).

2.2.4 Tube

Tube is a representation that highlights the backbone of the protein. It represents the backbone structure as a tube of constant radius that extends from the first backbone carbon atom to the last backbone carbon atom (11).

2.2.5 New Cartoon

New Cartoon is very much similar to the Tube representation, but as seen before, it must also show the secondary structure of that part of the backbone. As the backbone is traversed, the secondary structure should determine the shape of that part of the backbone. If it were an Alpha-Helix, it should be a flattened ribbon shape and if a Beta-Sheet it should be a rectangular arrow (12).

2.3 Game Engines

Game engines are frameworks that provide tools and features that are required to build games. These can be anything from 2D games to 3D games; they allow for audio and video playback, programming interactivity, user interfaces and gameplay, and most importantly allow for creating content on a variety of target platforms (13) .

The two most important and long-time competitors in this field are Unity 3D and Unreal Engine 4.

2.3.1 Unreal Engine

Unreal Engine 4 is a game engine written and scripted exclusively in C++. It was first developed in 1998 for the Unreal first-person shooter video game (14). It supports a wide variety of platforms and game types. Unreal Engine has been previously used to create architectural programs and simulation software (15).

2.3.2 Unity

Unity 3D is a game engine created with C++ and scripted in C#. By allowing gameplay code to be written entirely in C#, Unity simplifies the process by which code is written (16). Unity provides much of the same features as UE4 does but in a different way through a completely different API. All types of platforms can be targeted and virtual reality is very well supported.

2.3.3 Choice of game engine

Deciding between the two game engines comes down to which platforms are supported and how performant the solution should be. In the case of a molecular dynamics visualiser dealing with hundreds of thousands of atoms, the framework chosen should be the one which promises better performance and control. This would be Unreal Engine 4. Furthermore, it would be wise to choose a framework with good support for target platforms and virtual reality, both of which UE4 has. Therefore, Unreal Engine 4 is the better option to develop this application.

Given more time and a larger workforce, an application made from scratch would benefit even more from a performance boost due to more added control.

2.3.4 UE4 Basic Concepts and Terminology

Unreal Engine uses special terminology to refer to the various elements that comprise a game or simulation. A brief outline of the most important terms is given.

In UE4 objects are the basic building blocks of almost everything. Objects contain a lot of the essential functionality and metadata that is used for type reflection and garbage collection. By inheriting from UObject, this functionality is enabled on the custom class (17).

Classes define the behaviours and properties of a game object. This is one of the basic concepts in object-oriented programming languages; objects, are instances of classes. In UE4, however, classes expose properties/fields / member variables for modification in the Editor. The properties that are commonly editable in the editor are the X, Y and Z positions of the Object in the world, but this can be customized when defining the class through the UPROPERTY macro (17).

Actors are Objects that can be placed within the level. They support translation, rotation and scaling. These Actors are usually spawned at a location in the level and likewise can also be destroyed/removed from the level. There are several different kinds of Actors which provide different functionality (17).

Components are individually isolated bits of functionality that can be added to an Actor. These Components do not exist by themselves but when attached to an Actor, the Actor will be able to access it and use its functionality (17).

Pawns are a subclass of Actor, they normally serve as in-game characters that are controlled by the user. They can additionally be controlled by an AI to make a Non-player Character (NPC). Users can often switch between controlling different pawns at runtime. The act of a user switching control of pawns is called possession, i.e. “a player possesses a pawn” (17).

The Player controller is the class that is used to parse player input and translate it into an interaction with the game. By default, there is always a player controller. When an empty scene is loaded, a default player controller will be set providing basic keyboard and mouse functionality (17).

Levels are the sandboxes in which gameplay occurs. They are created in the editor by dragging and dropping Actors which could be players, terrain and other gameplay related elements that should be available from the start of play (17).

The world in UE4 is a bag which contains all the levels that have been created and it manages loading them and spawning the dynamic actors that will inhabit these levels (17).

GameModes are classes responsible for setting the rules that govern gameplay. This could be winning states and losing states, maintaining a scoreboard, etc. They can also contain more fundamental rules such as what happens when the game is paused or once it has started (17).

UE4 also provides a custom UI framework called Slate. Slate is very similar to Qt, a C++ GUI library, where it provides multiple layout possibilities and a variety of views into lists and other data structures (17).

2.4 Virtual Reality

Virtual Reality is the simulation of an alternate reality from the point of view of the user. The equipment usually needed to do this consists of a powerful computer, head mounted display, base stations and controllers.

A head mounted display (HMD) is a wearable device which contains a high-resolution display and two lenses, one for each eye. The display is rated at a high resolution and refresh rate to mitigate motion sickness and other detrimental side effects. In some cases, the HMD will have cameras installed to track the movement of the controllers.

In the case where there are no cameras installed in the HMD, there are base stations included in the kit. The base stations use LiDAR technology to track the positions and orientations of the HMD and controllers. The disadvantage with camera tracking in the HMD is that there is a large dead zone, encompassing any area the cameras cannot see; the base stations do not suffer any dead zone.

The controllers can come as a single unit or a pair. They usually feature buttons for the index and middle finger (so-called trigger buttons) and joysticks and buttons for the thumbs as commonly seen on many gaming controllers. A recent development in the touch controller field is Valve’s index controller that can track the individual movements of the fingers without bulky wearable gloves; the other controllers at most track index and middle fingers as does the Oculus touch controller.

2.4.1 Oculus Rift technology

The Oculus Rift S is a popular headset which is moderately priced. Due to its competitive pricing and good specs, it is a good option for VR Desktop applications. Furthermore, it is highly portable as opposed to other headsets which require base stations that have to be fixed to walls or be mounted on stands to help track the controllers. The Oculus Rift S makes use of “inside outside” tracking through an array of cameras to track the controllers, reducing the amount of kit needed to set up and get started.

2.4.1.1 Specifications

The Rift S Head Mounted Display (HMD) has a Quad High Definition (QHD, resolution: 2560 x 1440) LCD panel and a refresh rate of 80 Hz. The HMD is connected to a computer via a USB port and a Display port to drive the LCD panel. The Inside-out tracking features 6 Degrees of Freedom (DoF) which means it can keep track of the position and rotation of the controllers (18).

The Rift S is unlike some of the other headsets in the Oculus product line that features integrated compute and wireless connection. The additional compute power needed is provided through a wired USB port and a Display port connector. The minimum requirements for such external compute power is an Intel i3 processor / AMD Ryzen 3 1200 or an older FX4350, 8GB RAM, Windows 10 and NVIDIA GTX 960 4GB / AMD Radeon R9

290 or higher. These minimum requirements date back quite a few years and so purchasing equipment with similar or better specs will be more affordable today (18).

2.5 Currently used products

2.5.1 Visual Molecular Dynamics (VMD)

VMD is a popular software visualiser created by the University of Illinois (3). It is a desktop-based visualiser (available on Windows 10, Linux and Mac OS) and supports all popular formats that are used to describe molecular structures and simulation data. It has capabilities to create high quality renderings of molecules. It is also able to capture video clips of simulation data being animated. The animated frames can also be superimposed one on top of another to create a stylised image of a molecule in motion. VMD also supports a plugin system which can be used to extend the application with analytical features that will help users conduct some experiments on the data. It does not support Virtual Reality unlike ChimeraX, another molecule visualiser.

2.5.2 ChimeraX

ChimeraX is another software product similar to VMD (19). However, in addition to the features common with VMD, ChimeraX also supports a variety of VR kits and platforms. But it has been reported that the VR capabilities are only stable on Windows 10, where VR support is the best. This product does support Linux and Mac OS through the use of Steam VR but has a problem of stability when using VR (20).

Steam VR is a tool that allows a developer to create hardware independent VR experiences. This is possible through the API it uses, called OpenVR, which was developed by Valve, the parent company of Steam. OpenVR is a hardware independent API that can be used to target many different VR kits; it also supports Linux as a development build.

The control scheme for the Oculus Rift VR controllers used by ChimeraX is as follows:

- Translation
    - Right controller thumbstick defaults to zoom, forward pushes models away from the user
- Rotation
    - left controller thumbstick defaults to rotating the model around its centre of mass
- Scaling

  - holding both triggers and physically moving the controllers away from each other scales the model

ChimeraX also reports some limitations with VR found during the development. They acknowledge that support for Linux and Mac is limited. They indicate that while SteamVR works on Linux Ubuntu 18.04, there are bugs with the room play area detection not working properly, which can be dangerous. Steam's VR support for MacOS requires the use of many dongles, especially in the case of a VR headset that requires base stations, and also an external GPU (eGPU) to power the headset, which is not very practical. They have advised against using Linux and MacOS for the time being (until support for them improves), unless absolutely necessary.

ChimeraX in VR is also only able to render small models. Large models, with many thousands of atoms significantly increase the amount of time needed to render. Not being able to meet the timing requirements of 1/90th of a second (i.e. 90 FPS) can cause the VR headset to flicker or blackout (meaning that it will no longer render the frame since the deadline was missed) (20).

Another issue that was reported was the slow rendering of dynamic scenes. Dynamic scenes here refer to rendering a time-series of simulation data. They suggest rendering the simulation data slower than 90 FPS as required by the headset. By rendering fewer frames a second the deadline can be met more easily (20).

ChimeraX also does not support MD visualisation as standard. It supports a plugin system where users can write extensions to the base functionality of ChimeraX. There is an MD viewer which allows loading multi-frame PDB files (instead of trajectories such as XTC or DCD files), where the PDB contains multiple frames of atomic coordinates.

2.6 Summary

The software developed in this project will support Virtual Reality and allow reading in a variety of trajectory files to render dynamic scenes. The software will be built using Unreal Engine to make use of the C++ programming language performance gain, multiple target platforms and easy VR integration. It will also replicate the well-known user interface of VMD such that there is a short learning curve when the software is initially used.

3 System Design

3.1 System Requirements

Two tables are given below providing a list of functional and non-functional requirements. These requirements were established by experimenting with VMD and referencing its official documentation.

Table 1 Functional Requirements

#	Functional requirements	Mandatory (M) / Desirable (D)
F1	A user shall be able to load new Proteins from a PDB file	M
F2	A user shall be able to rotate, scale and translate the viewpoint of the molecule	M
F3	A user shall be able to select different drawing styles	M
F4	A user should be able to select different colouring methods	D
F5	A user shall be able to select different representations of molecules	M
F6	A user shall be able to create multiple representations of the same molecule	M
F7	A user shall be able to hide and show different representations	M
F8	A user should be able to inspect amino acids and their properties	D
F9	A user should be able to save representations and viewpoints, for loading at a later point in time	D
F10	A user should be able to choose from and create different BRDF materials to apply to representations	D
F11	A user should be able to choose from different perspective modes	D
F12	A user shall be able to load trajectory files to play animations	M
F13	A user should be able to smooth choppy simulation data	D
F14	A user shall be able to play animations loaded from trajectory files	M
F15	A user should be able to selectively draw frames from an animation	D
F16	A user should be able to save a scene as an image	D
F17	A user should be able to save a range of animation frames as a	D

	single image	
F18	A user should be able to view the visualisation in VR	D
F19	A user should be able to remap controls	D
F20	A user shall be able to Scrub through animations	M
F21	A user shall be able to run the application on Microsoft Windows 10	M
F22	A user shall be able to run the application on Ubuntu Linux 20.10	M

Table 2 Non-functional requirements

#	Non-functional requirements	Mandatory(M)/Desirable(D)
NF1	The system shall support rotation via left-click and drag	M
FN2	The system shall support rotation around the viewing direction via right-click and drag	M
NF3	The system shall support enabling rotation mode via keyboard shortcut R	M
NF4	The system shall support enabling translation mode via keyboard shortcut T	M
NF5	The system shall support enabling scaling mode via keyboard shortcut S	M
NF6	The system shall default rotation center point to the Center of Mass of a Molecule.	M
NF7	The system shall support Van der Waals representation	M
NF8	The system shall support New Cartoon representation	M
NF9	The system shall be implemented in UE4 and C++	M
NF10	The system should allow for catmull-rom and B-splines as a choice to render geometry	D
NF11	The system shall use the STRIDE program to compute the secondary structure information	M
NF12	The system shall allow for colouring each part of the representation by atom name	M
NF12	The system should allow for colouring each part of the representation by resType	D

NF14	The system should allow for colouring each part of the representation by residue number	D
NF15	The system shall show a menu to display different representations	M
NF16	The system should support a drop-down list or a keyword search for fine tuning representations	D
NF17	The system shall show a menu to create different representations.	M
NF18	The system should allow a representation to be applied to a specific selection of atoms or residues or chains	D
NF19	The system should allow representations to use custom colour schemes	D
NF20	The system should allow representations to use different materials	D
NF21	The system should allow representation to be toggled between viewing all or selecting few or none of them	D
NF22	The system should save a viewpoint, set of selections in a binary file.	D
NF23	The system should load a viewpoint, set of selections from a binary file.	D
NF24	The system should allow the user to select different standard materials to choose from Metallic, Paper, Translucent.	D
NF25	The system shall allow the molecule, when left clicked, to translate in the X, Z plane when translating	M
NF26	The system should support control mappings for Oculus rift OVR controllers	D
NF27	The system shall display secondary structures using New Cartoon representation	M
NF28	The system shall display different parts of the molecule with different representations	M
NF29	The system shall be able to play the simulation, running the simulation from it’s current timestep to finish	M
NF30	The system shall be able to stop, resetting the simulation animation to the start	M

NF31	The system shall be able to pause to stop playing the simulation animation at its current time step	M
NF32	The system shall support rewinding simulation data.	M
NF33	The system shall support fast-forwarding simulation data.	M
NF34	The system shall support looping simulation data.	M
NF35	The system should support taking screen captures of the viewport in its current state.	D
NF36	The system should allow the user to save configured representations to a proprietary binary file format.	D

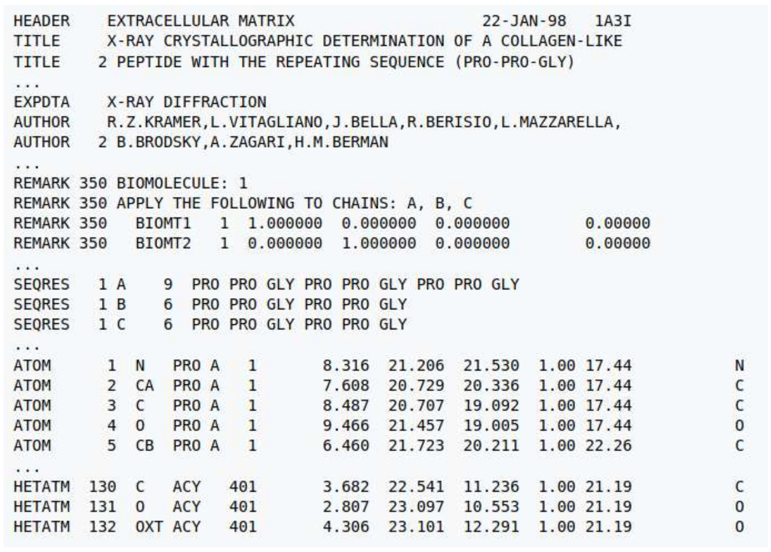
3.2 File Formats

In order to create representations of molecules, the most common file formats for loading them must be understood. Two files are commonly used to visualise MD simulations. One is the Protein Data Bank (PDB, structure file) and the other is the DCD / XTC (trajectory file). There are many other structural / trajectory file combinations that can be used.

3.2.1.1 The PDB File Format

The PDB file format has been around since 1971 and at the time was the best at storing the 3D structural information of macromolecules. It still currently widely used, although more modern formats are available but are not as widely supported (21).

The PDB file format takes the form of records, where each record is on a single row and contains information organized into columns. It is, therefore, possible to parse the file based on specific indices in the row. The file format is human readable and contains a lot of information that is not necessary for visualisation purposes. Figure 8 shows the PDB file format.

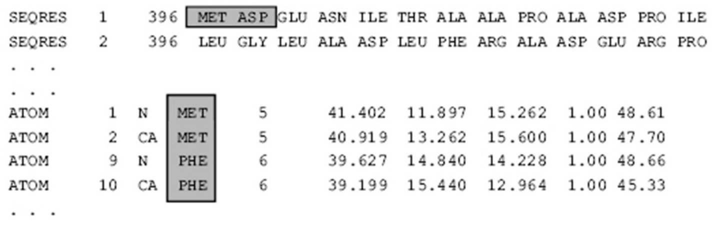


```
HEADER    EXTRACELLULAR MATRIX                    22-JAN-98   1A3I
TITLE     X-RAY CRYSTALLOGRAPHIC DETERMINATION OF A COLLAGEN-LIKE
TITLE    2 PEPTIDE WITH THE REPEATING SEQUENCE (PRO-PRO-GLY)
...
EXPDTA    X-RAY DIFFRACTION
AUTHOR    R.Z.KRAMER,L.VITAGLIANO,J.BELLA,R.BERISIO,L.MAZZARELLA,
AUTHOR   2 B.BRODSKY,A.ZAGARI,H.M.BERMAN
...
REMARK 350 BIOMOLECULE: 1
REMARK 350 APPLY THE FOLLOWING TO CHAINS: A, B, C
REMARK 350   BIOMT1   1  1.000000  0.000000  0.000000        0.00000
REMARK 350   BIOMT2   1  0.000000  1.000000  0.000000        0.00000
...
SEQRES   1 A    9  PRO PRO GLY PRO PRO GLY PRO PRO GLY
SEQRES   1 B    6  PRO PRO GLY PRO PRO GLY
SEQRES   1 C    6  PRO PRO GLY PRO PRO GLY
...
ATOM      1  N   PRO A   1       8.316  21.206  21.530  1.00 17.44           N
ATOM      2  CA  PRO A   1       7.608  20.729  20.336  1.00 17.44           C
ATOM      3  C   PRO A   1       8.487  20.707  19.092  1.00 17.44           C
ATOM      4  O   PRO A   1       9.466  21.457  19.005  1.00 17.44           O
ATOM      5  CB  PRO A   1       6.460  21.723  20.211  1.00 22.26           C
...
HETATM  130  C   ACY   401       3.682  22.541  11.236  1.00 21.19           C
HETATM  131  O   ACY   401       2.807  23.097  10.553  1.00 21.19           O
HETATM  132  OXT ACY   401       4.306  23.101  12.291  1.00 21.19           O
```

Figure 8 Protein Data Bank file format example **(22)**

This format has changed significantly since the original description in 1971 as the users required more detail. However, the format is now facing issues with large macromolecules since the upper limit of the number of atoms in one PDB file is 99,999 atoms. Large molecular structures need to be split up into multiple PDB files, which complicates parsing.

Another issue with the format is the inconsistencies in the records that refer to structural information. An example is given below, where there are issues not only with SEQRES records, which identify the sequence of amino acids in the molecule but also with the Secondary Structure information that may be included.



```
SEQRES   1    396  MET ASP GLU ASN ILE THR ALA ALA PRO ALA ASP PRO ILE
SEQRES   2    396  LEU GLY LEU ALA ASP LEU PHE ARG ALA ASP GLU ARG PRO
. . .
. . .
ATOM     1  N   MET     5       41.402  11.897  15.262  1.00 48.61
ATOM     2  CA  MET     5       40.919  13.262  15.600  1.00 47.70
ATOM     9  N   PHE     6       39.627  14.840  14.228  1.00 48.66
ATOM    10  CA  PHE     6       39.199  15.440  12.964  1.00 45.33
. . .
```

Figure 9 Inconsistencies in the PDB file format **(21)**

In Figure 9, a SEQRES record can be seen conflicting with the order of the ATOM records in the coordinate section.

The PDB file is organised into sections, where the first section is the title section which contains information such as the REMARK records. Then, in the following order, there is the primary structure section, heterogen section, secondary structure section, connectivity annotation section, miscellaneous features section crystallographic and coordinate transform section, the coordinate section, connectivity and finally the bookkeeping section (23).

The Title section contains information to describe the experiment and the molecule represented in the file. Most of this content is for the humans reading the file and is not of much concern to a visualiser. A noteworthy record is the TITLE record which contains string data that is usually the title of the experiment that has been recorded in the PDB file. There can be numerous TITLE records each one being a continuation from the previous one as the title can be longer than what is allowed on a single line (23).

The Primary Structure section contains information about the sequence of residues in the chains of the molecule. This is where the SEQRES records can be found and each of these records contains the sequential links of the residues that make up the polymer chain. As previously mentioned, this section may be inconsistent with the coordinate section records and so can be ignored to simplify parsing. No information is lost as the sequence of residues can also be retrieved from parsing the coordinate section (23).

The Heterogen section contains the description of nonstandard residues present in the file. These are residues that do not contribute to the main polymer, such as water molecules that are present alongside the molecule. This section is also not vital to rendering and can be ignored. Heterogenous atoms will have their coordinates, and residues listed in the coordinate section (23).

The Secondary Structure section contains information about the molecules Alpha-Helix and Beta-Sheets. This section contains records that describe where helices start & end and at which residues it starts & ends. All of these descriptions are useful (23). However, they are seldom provided and in most cases, another program called STRIDE (described in 3.3) can provide this information more reliably.

The Connectivity Annotation section identifies disulphide bonds and other linkages that normally would not be generated by an algorithm. This section is not as relevant, again, as the coordinate section since that would also list these bonds (23).

The Miscellaneous Features section provides descriptions of the properties in the molecule that are non-standard. This is irrelevant to the visualisation of molecular structures and can be ignored (23).

The Crystallographic and Coordinate Transformation section provides a description of the molecules geometry and coordinate system during the conducted crystallographic experiment. Crystallography is a way in which one can observe a molecule during an experiment; it is a physical simulation. As such, it requires a coordinate system to be described so orientation can be determined. This is not essential as the user will be allowed to orient themselves and the protein in whichever way they require (23).

Finally, the Coordinate section contains the vital information required to render a molecule. This is the location and type of each atom that is part of the molecule; these are the ATOM records. It also contains information of the heterogenous atoms as HETATM records which contain the same information as ATOM records. These records are followed by the name, alternate location indicator, residue name, chain identifier, residue sequence number, code for insertion of residues, the x,y,z coordinates in Angstrom units, occupancy, temperature indicator, element symbol (right-justified) and the charge on the atom. The reason why they are present is due to the fact that proteins are suspended in, for example, a water solution and so HETATMS would describe the water molecules (23). There are also TER records which describe when a chain ends and a new one starts.

The Connectivity section contains the CONNECT records which will specify bonds between atoms. These are special bonds that cannot be easily detected algorithmically (23).

3.2.1.2 The DCD file format

DCD files do not have a well-documented format, but rather some loose guidelines can be found. Therefore, it would be recommended to use previously written and accepted code, which can be found within the VMD source code or even the NAMD source code.

The DCD file appears to contain a header block, title block, natoms block (which contains one number indicating the number of atoms that are tracked by each frame), fixedIdx block (optional block that contains an index of free atoms), any number of frame blocks, each containing 3 blocks for x coordinates, y coordinates and z coordinates of atoms in that frame (24).

3.2.1.3 The XTC File Format

Another popular file format is the XTC trajectory file used by GROMACS, a molecular dynamics simulation tool (not a visualiser). It contains the positions of all atoms in each frame of the simulation. This format is more popular among MD researchers than the DCD format (25).

The only complication with XTC file format is that it is written using a platform independent format. So in order to read it, it is necessary to use the same routines that created it. The writing is done through XDR routines, which stands for External Data Representation and is only available on Unix like operating systems. This is because the XTC file format was made for GROMACS which is mainly supported on Linux. The GROMACS program distributes a XTC file reader for C programs. Using this program it will be possible to read in the XTC file format accurately; however, it will have to be done on a Linux operating system (25).

The XTC file format is composed of a header and a body. The header contains metadata about the frame rate, number of frames and number of atoms. All of this can then be used to parse the locations of all atoms in the body of the file. Among the positional data, there are also forces, velocities and energies, which are commonly used to color code the molecules during the simulation (25).

3.3 Secondary Structure Computation using Structural Identification program (STRIDE)

The Stride program supplied with VMD is used in a separate thread to create the data required to render a secondary structure representation of the protein. Stride is a command-line application, which takes a protein structure file, such as a PDB, and outputs the secondary structure information to the console output. This can be redirected to a file using simple command line (CLI) tools and then parsed to generate the secondary structure geometry in the application.

```
REM  -------------------- Secondary structure summary -------------------  ~~~~
REM                                                   ~~~~
CHN  proteins/alanin.pdb -                                    ~~~~
REM                                                   ~~~~
REM           .                                       ~~~~
SEQ  1   XAAAAAAAAAAX                          12       ~~~~
STR      HHHHHHHHHH                                   ~~~~
REM                                                   ~~~~
REM                                                   ~~~~
REM                                                   ~~~~
LOC  AlphaHelix   ALA    2 -    ALA    11 -                 ~~~~
REM                                                   ~~~~
REM  --------------- Detailed secondary structure assignment------------  ~~~~
REM                                                   ~~~~
REM  |---Residue---|   |--Structure--|  |-Phi-|  |-Psi-|  |-Area-|     ~~~~
ASG  ACE -   1   1   C       Coil   360.00   360.00     0.0    ~~~~
ASG  ALA -   2   2   H   AlphaHelix   -57.01   -47.00     0.0    ~~~~
ASG  ALA -   3   3   H   AlphaHelix   -56.99   -47.04     0.0    ~~~~
ASG  ALA -   4   4   H   AlphaHelix   -57.00   -46.96     0.0    ~~~~
ASG  ALA -   5   5   H   AlphaHelix   -57.03   -46.93     0.0    ~~~~
ASG  ALA -   6   6   H   AlphaHelix   -57.04   -47.02     0.0    ~~~~
ASG  ALA -   7   7   H   AlphaHelix   -57.01   -46.94     0.0    ~~~~
ASG  ALA -   8   8   H   AlphaHelix   -57.09   -46.98     0.0    ~~~~
ASG  ALA -   9   9   H   AlphaHelix   -56.98   -47.04     0.0    ~~~~
ASG  ALA -  10  10   H   AlphaHelix   -56.96   -47.04     0.0    ~~~~
ASG  ALA -  11  11   H   AlphaHelix   -56.96   -47.02     0.0    ~~~~
ASG  CBX -  12  12   C       Coil   360.00   360.00     0.0    ~~~~
```

Figure 10 STRIDE program output

Figure 10 shows the output of the secondary structure information from the STRIDE program. It can be seen that there are Alpha-Helices within the secondary structure, which is one of two shapes possible (the other being Beta-Sheets). This information can be used to create the helix primitive and sheet primitive.

Take for example the first “ASG” tagged line. In this line the data is delimited by spaces and sometimes a dash. The structure of one ASG line is as follows: we have first the residue name, followed by a dash, chain id, residue id and secondary structure type. The secondary structures are Coil (C) on residues1 and 12 and Alpha-Helix (H) on residues 2 – 11 (26).

The values are as follows, after the ASG line tag: Residue type, residue number, SS line number, One letter secondary structure code, full secondary structure name, phi angle and psi angle and finally the residue solvent accessible area (26).

```
6-8  Residue name
10-10 Protein chain identifier
12-15 PDB  residue    number
17-20 Ordinal residue number
25-25 One  letter secondary structure code   **)
27-39 Full secondary structure name
43-49 Phi  angle
53-59 Psi  angle
65-69 Residue solvent accessible area
```

Figure 11 STRIDE format description **(26)**

In the above description, we can see clearly which column numbers contain what data.

We are primarily concerned with the residue number and the assigned secondary structure type. This will allow us to generate the appropriate geometry for that residue as we trace along the backbone of the protein structure.

In order to identify the secondary structure shape, we must first collect all atoms which make up the backbone. In a protein there are exactly 3 atoms in each residue that form part of the backbone. These can easily be identified by finding a common atom between the two atoms that are neighbours to atoms outside of this residue. An even simpler method would be to use the atom names which are within the PDB file to collect them immediately. The protein structure, as described previously is such that the backbone is comprised of Carbon, Alpha Carbon and Nitrogen (8).

In order to find the backbone of a protein, first the bonded atoms must be found. Given two atoms, the inter atom distance should be between 0.4 and 1.2 angstroms and if one of the two is a Hydrogen atom it must be between 0.4 and 1.7 angstroms. Once they are found, it is just a matter of creating a link between the two atoms of that length, which satisfy the aforementioned conditions (27).

After the network of atoms has been established, the PDB information can then be used, where the atom name is used to find the N, CA, C and O atoms. Once this is done we can store the backbone atoms in the correct order.

3.4 UE4 Blueprints and Materials

Blueprints in UE4 refers to their visual scripting language. It is very powerful and lets one program gameplay logic in an object-oriented manner without writing a single line of code. It is also possible to extend the functionality of Blueprints by writing a C++ class marked with the relevant macros, such as UCLASS(BlueprintType) and UFUNCTION(…). This will allow users to create Blueprints that inherit from the C++ class, thereby providing all UFUNCTION marked member functions and UPROPERTY marked member fields accessible from the Blueprint editor (28).

The usual workflow in UE4 would be to write custom functions in C++ and inherit them in a Blueprint to code the gameplay logic. However, this does not suit the open source/performance mindset since it becomes a lot more difficult to track what is happening and when it is happening. There is also less control over the performance aspects since the expression is restricted to what is available in Blueprints (28).

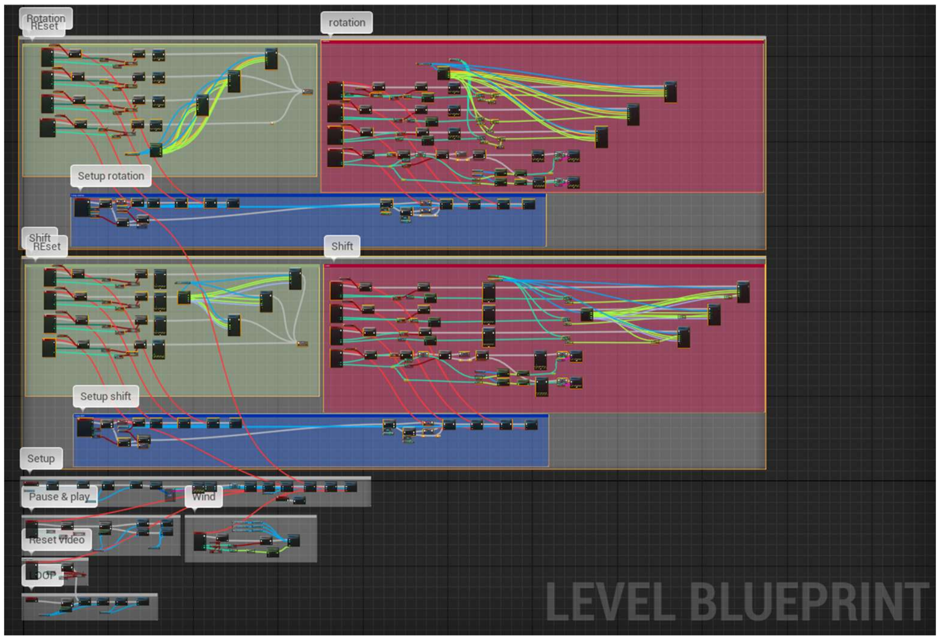


Figure 12 Blueprint example **(29)**

Figure 12 shows why one would not want to use blueprints. Some of the organisational aspects of a blueprint can be seen here by the comment boxes that enclose parts of the blueprint graph. The other organisation aspects are blueprint functions, which are isolated chunks of blueprint graphs with input and output specifiers.

Despite having these organisational facilities, blueprints can still be hard to understand. The comment boxes do not lend themselves well to long comments which may be needed if the functionality being describing is complicated. Furthermore, clicking through to many different blueprint functions needs the user to maintain information on what they are tracking, making it easy to get lost.

For this project, C++ will be used to the maximum. This is to simplify the overhead required for new users to understand the codebase. It is also good practice to move complex operations into C++ code rather than keeping them in Blueprints. Figure 12 is a poor example of a Level Blueprint, meaning that it is a global script since a Level contains every actor for play. This script is a setup script which initialises rotations, and implements some global logic like pause and play, reset video, etc which could be easily moved to C++ code.

Materials are assets that can be applied to meshes to control their visual appearance. Materials, technically, define the way in which light interacts with the surface of the mesh, so in a sense, this can be thought of as an instance of a shader program (30).

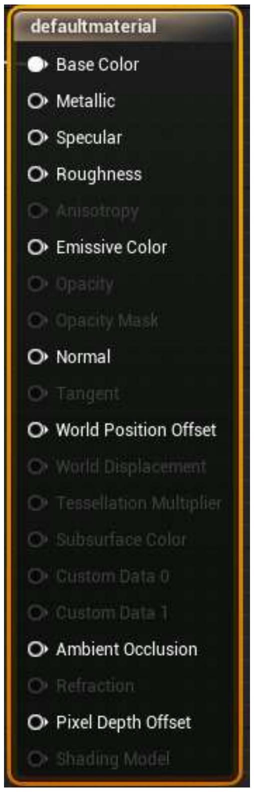


Figure 14 Material inputs

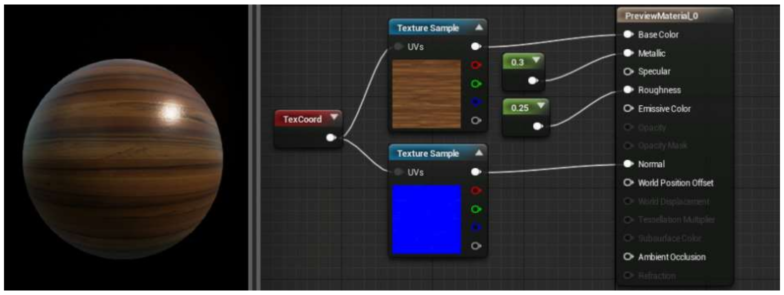


Figure 13 Material editor **(31)**

Materials are created and edited without any coding and are very much like blueprints. However, although previously noted that blueprints should be avoided, it is simpler to do material programming in blueprints than in code.
Figure 14 shows the available input values that the material provides.

- Base colour is the colour of the material,
- Metallic determines how metallic in appearance the material is (32),
- Specular defines how reflective a non-metallic surface is,
- Roughness controls the matte effect (32),
- Anisotropy is how the materials roughness changes with the direction of light (32),

- Emissive colour is the colour that the material emits (32),
- Opacity is how the transparent or opaque it is and the opacity mask is a texture which applies opacity in specific areas of the material (it only contains the values 0 / 1 for opaque and transparent) (32),
- World position offset allows for vertices in the mesh to be manipulated by the material which is useful for ambient animations, like waving trees (32),
- Normal is the input for the mesh normal map which is used in lighting calculations that compute shadows and highlights or roughness (32),
- World displacement is similar to world offset except that it creates new geometry, essentially extending the bounds of the mesh using tessellation; tessellation multiplier controls how much geometry is produced on the surface (32),
- Subsurface colour is the colour that is beneath the surface of the mesh; this is used when modelling human flesh such that the colour changes depending on how the lights pass through the surface (32),
- Ambient occlusion simulates self-shadowing within crevices of meshes, the input to this is a texture map describing how much light can enter a part of the mesh (32),
- Pixel depth offset allows a user to create custom logic to blend objects that overlap,
- Tessellation is the process by which new primitives are created on GPU (32).

For the simple material that will be used in the system, most of these will not be used. VMD only allows modification of Ambient, Diffuse, Specular, Shininess, Mirror, Opacity, Outline and OutlineWidth. In our case, this would translate to controlling BaseColor, Metallic, Specular, Roughness and Opacity. Outline and OutlineWidth would not be included due to performance concerns as reported by ChimeraX in the previous section.

Materials also support strong custom data per primitive, where a primitive is a single mesh instance. So each instance of a mesh can have its own set of variable values to modify how the mesh looks; like its base colour of the material can vary by the element type of the atom that is being rendered. Instance variables are the equivalent of a uniform buffer array (33).

3.5 UE4 mesh rendering pipeline

In order to display custom meshes and understand performance limitations an analysis of the mesh drawing pipeline is provided.

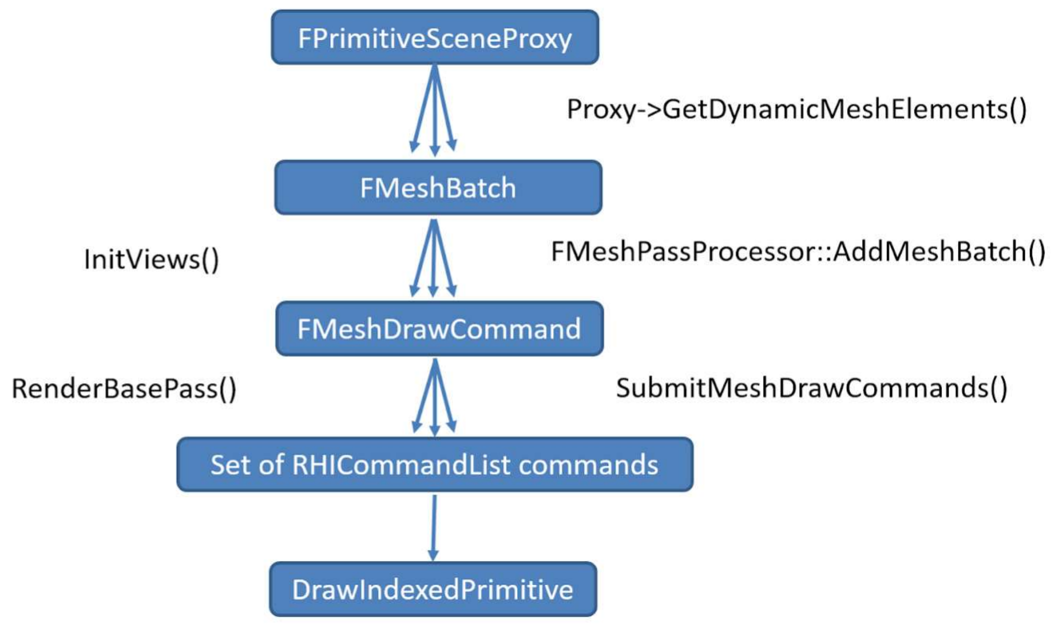


Figure 15 Mesh rendering pipeline **(34)**

Figure 15 shows the render thread architecture of UE4. The UStaticMeshComponent, which represents 3D geometry, provides an override for a function call CreateSceneProxy which returns an instance of the FPrimitiveSceneProxy. This function is called by UE4 to create the representation of the mesh in the rendering thread; this is the FPrimitiveSceneProxy which can be seen at the top of Figure 15. A call to draw the mesh is then processed by invoking either the GetDynamicMeshElement or DrawStaticElements methods. This produces a set of FMeshBatch structs that contains all the information necessary for the Render Hardware Interface (RHI) to draw the geometry. However, the RHI does not process these FMeshBatch structs; first they must be converted to FMeshDrawCommands. FMeshDrawCommands are how draw calls are represented in UE4 and at this stage it is possible to merge them to optimize performance, but this sort of automatic batching is only supported when using the Direct X 3D 11 and 12 APIs (34).

Draw call merging occurs when two draw calls have the same shader binding. This means that they use the same shader program to render the geometry and so can be merged into an instanced draw call (34).

Once any optimization to the FMeshDrawCommands is done, they are passed to the RHI to render via the SubmitMeshDrawCommands method. This method converts the FMeshDrawCommands into an RHICommandList which is finally used by the RHI to draw the mesh (34).

The USimpleDynamicMesh is based on a struct called a DynamicMesh3 which is essentially a directed edge data structure that can be used to efficiently store meshes and process them (35). The generated mesh can then be cached and regenerated when quality settings are changed. VMD does it similarly with the built in Quadric data types supplied with the GLUT library.

Once the geometry has been produced it seldom changes unless quality is decreased or increased at the user's request. In this case the static mesh would be invalidated, along with all of its instances and would thereafter be recreated with the newly specified quality. This might cause poor performance and so a better approach would be to use an instanced mesh rather than a dynamic mesh.

Better memory usage can be achieved by making use of mesh instancing, which unfortunately is not provided with the USimpleDynamicMeshComponent. It is possible to first create the geometry in Blender or any other 3D modelling software and then assign it to a UInstancedMeshComponent, which can create instances of the mesh efficiently (36). This further gives us the opportunity to use instanced material variables, which will be useful in colouring the atoms according to some preselected criteria, such as element name or atomic charge. Instanced variables are stored in an array in the GPU such that they can be accessed efficiently. Although these could be just stored on the component with USimpleDynamicMeshComponent, this does not lead to good memory usage in the GPU as data will have to be loaded in for each primitive, unlike with the UInstancedMeshComponent (36).

The main advantage of creating meshes on the fly was to allow users to dynamically set the quality of the meshes as needed to improve performance. With UInstancedMeshComponent this becomes difficult to code in C++. The alternative is to make use of UE4's automatic computation of Level of Detail (LOD) for meshes. LOD is a way in which a mesh's number of vertices can be reduced depending on the size of the mesh/screen. The logic behind doing this is to save GPU time by processing fewer vertices without perceivable loss in quality. UE4 is able to do this by using quadratic mesh simplification which is a technique that computes the amount of visual difference there is when joining two connected vertices; then choosing to join vertices with the least perceivable difference. The LOD groups are then applied depending on how visible the mesh is. For example, if the mesh were take up most of the screen space it should be rendered in high quality, otherwise in low quality (37).

3.6 Designing for Virtual Reality

3.6.1 Best practices

When developing for Virtual Reality, there are certain areas such as Vision, Locomotion and User Input that should be given extra attention. If not, side effects such as motion sickness, fatigue and vection (illusion of self-motion) could occur ruining the user experience.

For good user experience, as a general rule, it should be possible for a user to suspend their session to take a break and resume right where they left off. This is very important as usually users stand up when interacting with a VR scene. Allowing them to take the HMD off and put the controllers down to have a seat or tend to other matters, then pick up right where they left off will help in making them more comfortable (38).

Having shorter load times will also greatly help in making the user more comfortable. Usually in a desktop application a user is free to look at their phone or do something else while waiting for the application to load. This is not easy to do when an HMD is strapped on. Optimizing load times will greatly help improve the user experience as less time is spent waiting (38).

The human brain makes use of a number of different properties to determine the depth of objects in the world. Depth is an extremely important aspect of the visible experience provided by an HMD. If this were to be even slightly off, it would be uncomfortable for the user (39) .

These properties are

- depth cues such as motion parallax, which is how fast objects at differing distances move relative to each other,
- occlusion, which is close objects obstructing the vision of distant objects,
- curvilinear perspective which is the property that straight lines eventually converge as they reach the horizon,
- relative scale, where distant objects are smaller than closer ones,
- aerial perspective where the further away objects are, the fainter they appear, due to refraction of light in the atmosphere,
- texture gradients which are how tightly packed certain patterns become when they are in the distance, and finally
- lighting which casts shadows in the world and on objects.

Another important aspect of creating a VR app is related to the positioning of objects that are meant to be focused on for extended periods of time. Unlike in the real world where the eye can focus using two features called accommodative demand and vergence demand, in VR, only vergence demand is used. Accommodative demand is where the eye will adjust the shape of the lens to bring focus to an image plane, and vergence demand is how much the eyes rotate inwards to focus on a single point. In VR, there is only one image plane to focus on and the only way they eyes can adapt is through vergence. In the real world these are highly correlated but when wearing an HMD, this is not the case (39).

This decoupling of Vergence with Accommodation can create eye strain, where the eyes rotate drastically to bring objects into focus. Different users experience different levels of eye strain and so as a general guideline it is ideal to place objects that require the user's attention for long periods of time at a distance of 0.5 to 1 meter away (39).

It is also found that adding blur or depth field effects around objects on which users will focus can be more immersive as distracting background objects are out of focus (39).

Other areas that affect the user experience are Heads Up Displays (HUD) and Menu screens. These usually occlude the screen in normal desktop applications; however, doing this in VR can be extremely disorienting as it affects the user's depth perception by creating a contradiction. It is better to create Menus that reside on a 2D plane in the 3D game world which can be occluded by objects in the world (39).

Locomotion is the most difficult thing to get right when developing a VR application. Poorly implemented locomotion is one of the primary causes of motion sickness. While the user can move around the room that they are in to provoke a movement in the game, this usually is not enough to move across the entire map. Locomotion refers to movement such as rotation and acceleration not initiated by real-world movement of the user (40).

Common VR locomotion techniques that are industry tested are teleportation, where a user points using their controller and then teleports to that location while forcing a blink (lerp to black and back) during the teleportation. The reason for the blink is that it leverages the brain's ability to fill in gaps in the perceptual stream of information.

Other approaches for locomotion include artificial movement where the user moves at a constant velocity and along straight lines, like walking between two fixed points. HMD acceleration is where a user tilts in the direction they wish to move. This moves the environment instead of the user, in a grab and pull fashion (40).

The touch controllers provided with the Rift S give 6 DoF and finger tracking. This allows developers to create somewhat realistic hand controls with virtual hands in place of real ones. In order to have a comfortable feeling, it is best to use non-realistic hand models, i.e., ones that do not have fleshy appearances and colors, since if they do not look like the user's real hands when they try to mimic, it will be uncomfortable for the user. The model should only be hands; no forearms and elbows should be included as these would need to be aligned with the user's real forearms and elbows, which is more difficult to do (41).

When interacting such as lifting, pulling and pushing weighted objects, the controllers will be unable to provide resistance. Mimicking this resistance on the model should be avoided as this can be disorienting. It is best to stick to lightweight interactions (41).

Other factors to take into account are animations, which should be avoided. Picking up objects should be intuitive and should snap correctly to the hand, i.e., picking up a gun should only be done in one way, by the grip (41).

The Oculus Rift S has a requirement that the frame rate must be at 90 FPS. Frame rate consistency is also very important as dips in the framerate are going to make the user uncomfortable. UE4 also does not allow pipelining multiple frames due to input tracking latency, meaning the game thread is only one frame ahead of the render thread and the GPU and render thread is in sync. This leaves 11 ms of GPU time because projection takes longer as it needs to be done from two perspectives, one from each eye (42). The three factors that need to be balanced are the game thread for gameplay code, render thread for draw calls and the GPU thread for transformations and shading (42).

3.6.2 Unreal Engine integration

Unreal engine already provides integrations for the Oculus Rift SDK. This will allow relatively easy porting to the Oculus Rift S HMD and touch controllers. To get intuitive controls and good performance is a design and architectural problem which might be more challenging than porting the standard controls.

3.7 System Architecture

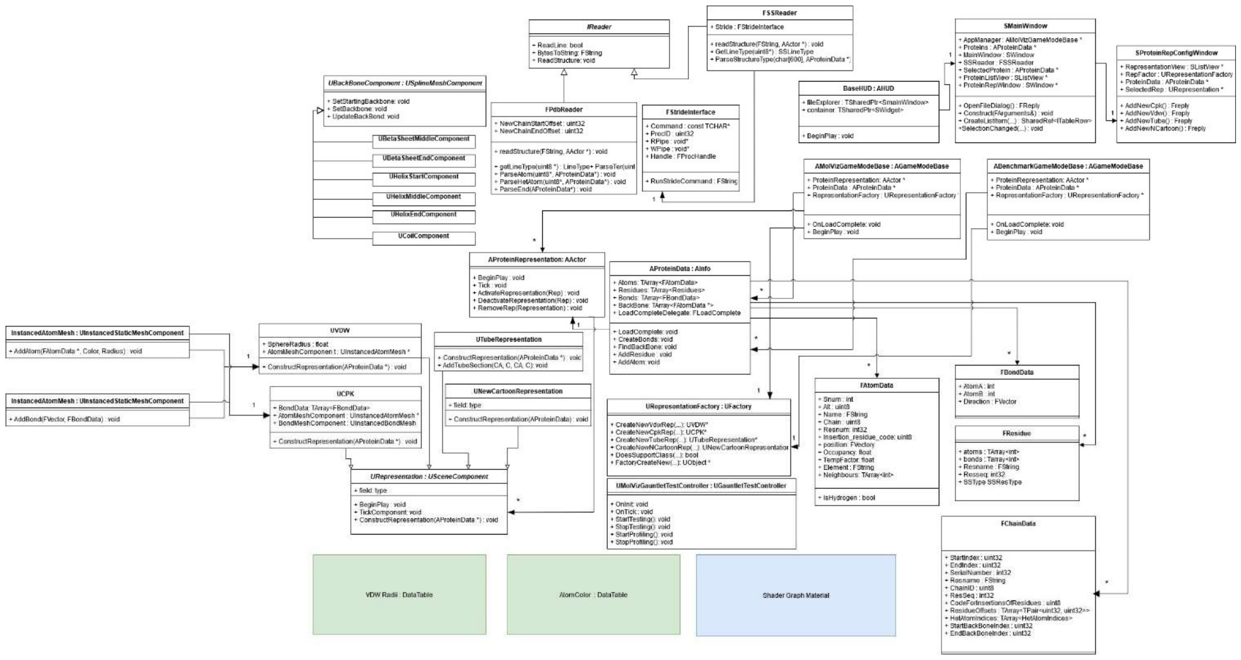

IReader
FSSReader
SMainWindow
SProteinRepConfigWindow
UBackBoneComponent : USplineMeshComponent
FPdbReader
FStrideInterface
BaseHUD : AHUD
UBetaSheetMiddleComponent
UBetaSheetEndComponent
UHelixStartComponent
UHelixMiddleComponent
UHelixEndComponent
UCoilComponent
AMolVizGameModeBase : AGameModeBase
ABenchmarkGameModeBase : AGameModeBase
AProteinRepresentation: AActor
AProteinData : AInfo
InstancedAtomMesh : UInstancedStaticMeshComponent
UVDW
UTubeRepresentation
UCPK
UNewCartoonRepresentation
URepresentationFactory : UFactory
FAtomData
FBondData
FResidue
URepresentation : USceneComponent
UMolVizGauntletTestController : UGauntletTestController
FChainData
VDW Radii : DataTable
AtomColor : DataTable
Shader Graph Material

Figure 16 System architecture (see Appendix C.5 for a larger version of the diagram)

In Figure 16, the architecture of the software is shown. The major parts of the system are, the representation of molecules, the data behind each molecule, user interface and readers.

3.7.1 Readers

The two file formats that require parsing are the PDB and SS structure program output.

The PDB file, being a human readable format contains a lot of information that is not necessary to parse. Luckily each line contains a tag and so any information that is not required can be ignored after reading the first few bytes in each line. This is handled by matching against the relevant line type and reading subsequent bytes in the line in a certain format.

The lines that are of key interest are the ATOM, HETATM, TER and END records. The ATOM record contains information key to determining the element and position of an atom in the molecule. The HETATM record contains information much like the ATOM record but of heterogenous atoms, which do not belong to the molecule and so should be stored separately. These can be H_2O atoms or free floating O atoms, which are present due to the environments in which the physical simulation was done. TER and END records show when a Chain in the molecule ends and when the end of the file is reached respectively (23). Chain is a collection of residues, they are not connected to each other. Sectioning off atoms and HETATMs into the chains that they belong to makes it easier to later extend representations by applying them to specific chains and also is an easy way to make sure bonds are not drawn where they should not be.

3.7.2 Model View Controller (MVC)

The system was architected to follow the Model View Controller design pattern.

The View is the ProteinRepresentation class, which handles the geometry needed to render the various representations. In order to help with this, the factory design pattern is used to help setup the various representations. Having all of the rendering related code separated allows for easier extendibility and flexibility in the future when new representations need to be added. In this case no changes are required to be made to the way the underlying data is stored.

Interaction with the Model is via a User Interface implemented in UE4 UI framework called Slate. This is the same UI framework used to build the UE4's Editor interface. It is very much similar to Qt in that it works by polling for changes in data, provides call-backs and delegates as an event based system to cause updates to the UI (43).

It also features a declarative syntax to build up UIs. This means that there is heavy use of Macros and overloaded operators as it is all written in C++ in order to benefit from compile-time checks. As a part of its language the UI framework features various layouts and views that can be nested into a tree-like structure (43).

The controller allows for the View to interact with the Model. The view is the User Interface which is implemented as a set of windows with buttons and lists. The View needs to be able to reflect the changes made to the model as well as initiate them. In order to do so while maintaining a clean interface on the model, a Controller is introduced. The Controller is in the form of a AGameModeBase which is normally used to govern the rules and scoring of a game session. This can be repurposed to setup the UI, and allow it to interact with the Actors in the scene.

Finally, the Model keeps up to date the various data needed to render a Molecule. The AProteinData class maintains lists of Atoms, HETATMs, Residues and Chains. All of these are stored as separate arrays so as to make use of data locality

4 Software Implementation

4.1 User Interface

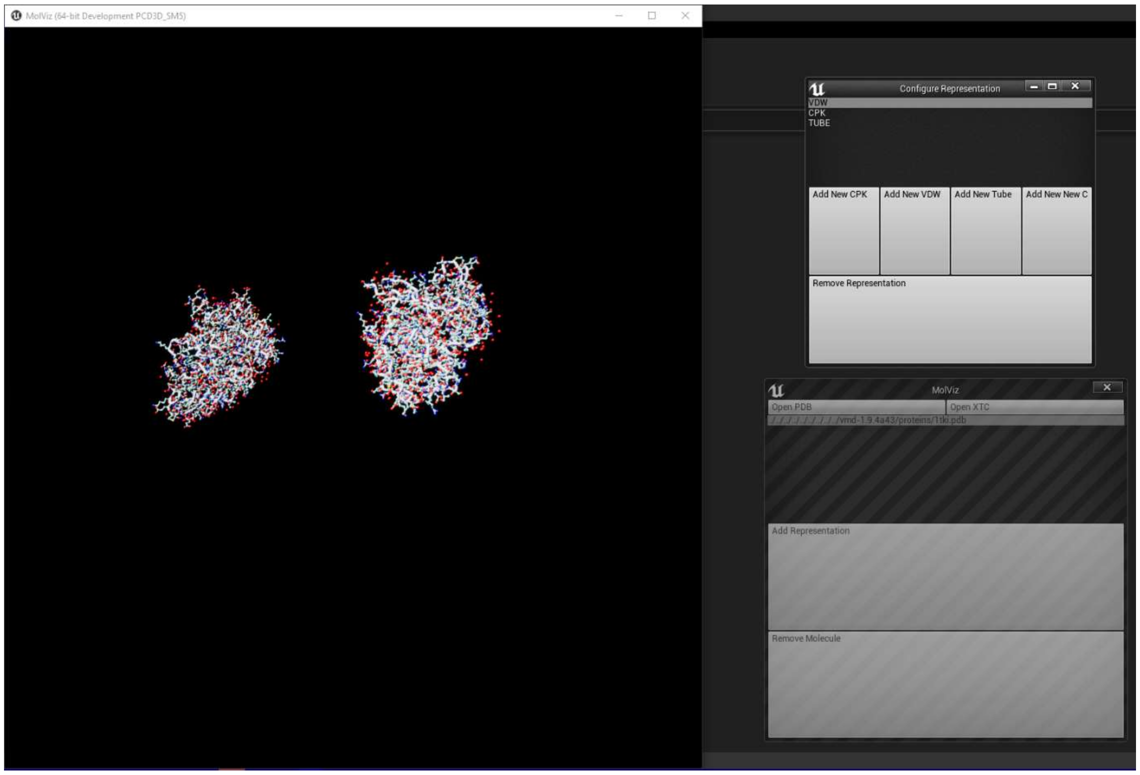
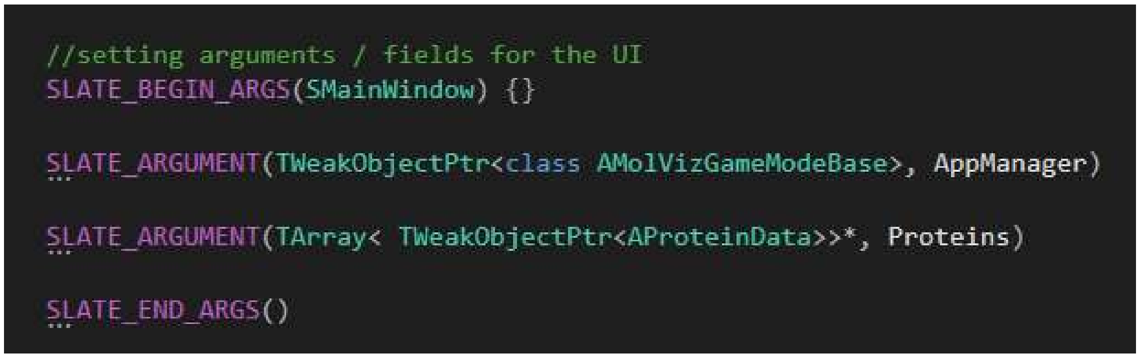


Figure 17 User Interface to control application

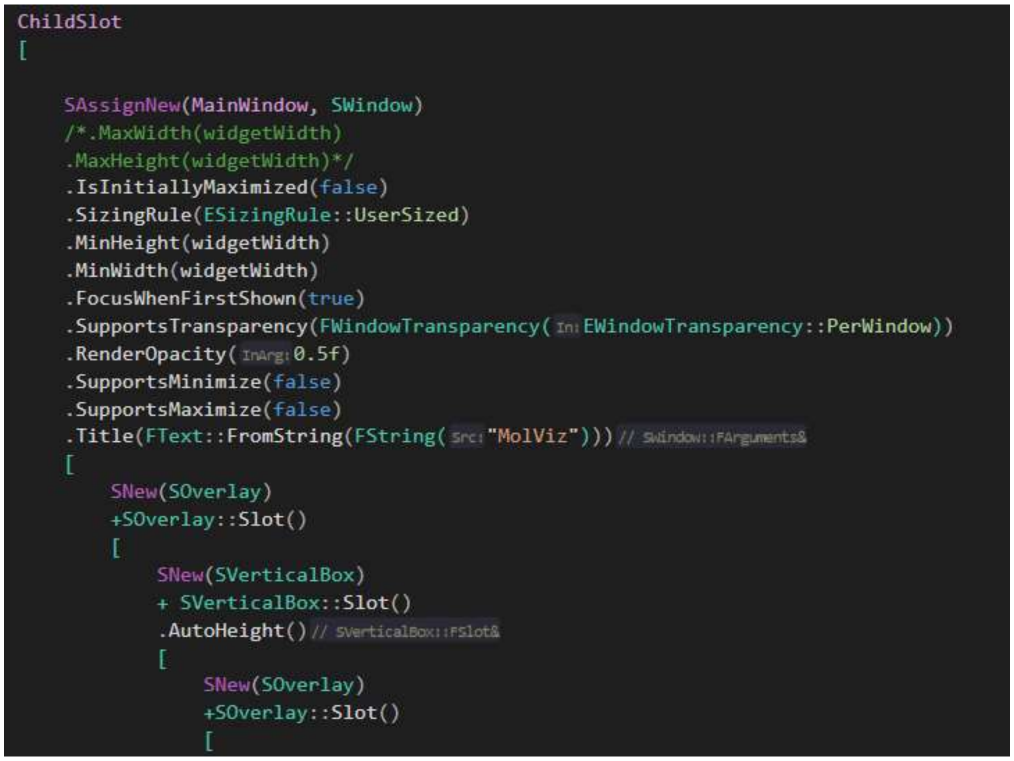
In Figure 17 the user interface used to add and manipulate a molecule is shown. The main window, which is on the bottom right shows all loaded PDBs in a list view and allows a user to load and remove them as needed. Clicking on the buttons to add representations open a secondary window similar to the main window. It can be seen on the top right and it features a list view showing all the created representations. There are also buttons to add new representations and to remove representations that have been added. Furthermore, double-clicking an entry in the list will hide the representation and doing so again will show it.



```
//setting arguments / fields for the UI
SLATE_BEGIN_ARGS(SMainWindow) {}

SLATE_ARGUMENT(TWeakObjectPtr<class AMolVizGameModeBase>, AppManager)

SLATE_ARGUMENT(TArray< TWeakObjectPtr<AProteinData>>*, Proteins)

SLATE_END_ARGS()
```

Figure 18 Slate UI macros for setting up the required code

Figure 18 shows the C++ macros that are used to setup the required boilerplate code for a Slate UI class. This specific snippet shows the setup of the arguments that can be passed to an instance of this UI class, by way of the SLATE_ARGUMENT macro. This macro takes a type and a variable name, creating the declaration of the variable and Accessor / Mutator functions. It also creates additional fields of the specified types in a special struct passed to the constructor of the UI class for initializing parameters.



```
ChildSlot
[

    SAssignNew(MainWindow, SWindow)
    /*.MaxWidth(widgetWidth)
    .MaxHeight(widgetWidth)*/
    .IsInitiallyMaximized(false)
    .SizingRule(ESizingRule::UserSized)
    .MinHeight(widgetWidth)
    .MinWidth(widgetWidth)
    .FocusWhenFirstShown(true)
    .SupportsTransparency(FWindowTransparency( In: EWindowTransparency::PerWindow))
    .RenderOpacity( InArg: 0.5f)
    .SupportsMinimize(false)
    .SupportsMaximize(false)
    .Title(FText::FromString(FString( Src: "MolViz"))) // SWindow::FArguments&
    [
        SNew(SOverlay)
        +SOverlay::Slot()
        [
            SNew(SVerticalBox)
            + SVerticalBox::Slot()
            .AutoHeight() // SVerticalBox::FSlot&
            [
                SNew(SOverlay)
                +SOverlay::Slot()
                [
```

Figure 19 Slate UI declarative syntax

Figure 19 shows the declarative syntax used to create the various UI elements. The square bracket is an overloaded function that takes a TSharedReference to a UI class, adding it to the parent UI elements list of child elements. The macro used to create new UI elements is SAssignNew or SNew. SAssignNew creates an instance of the specified class, for example SWindow, and assigns it to a specified variable, MainWindow for example. SNew does the same as SAssignNew but without assigning to a variable. Once this has been done, the newly created UI element can be configured, such as setting its SizingRule or MinHeight / MinWidth.

In order to create more complex UI layouts the "+" operator is overloaded on certain UI classes such as the SVerticalBox class. This function takes a Slot, which is used to assign child UI elements. Every new Slot that is added to the SVerticalBox class creates a new vertically stacked box in the UI where a new child element can be assigned. Layouts like this can be nested one in the other to create complex and highly organized UIs.

4.2 Van der Waals (VDW)

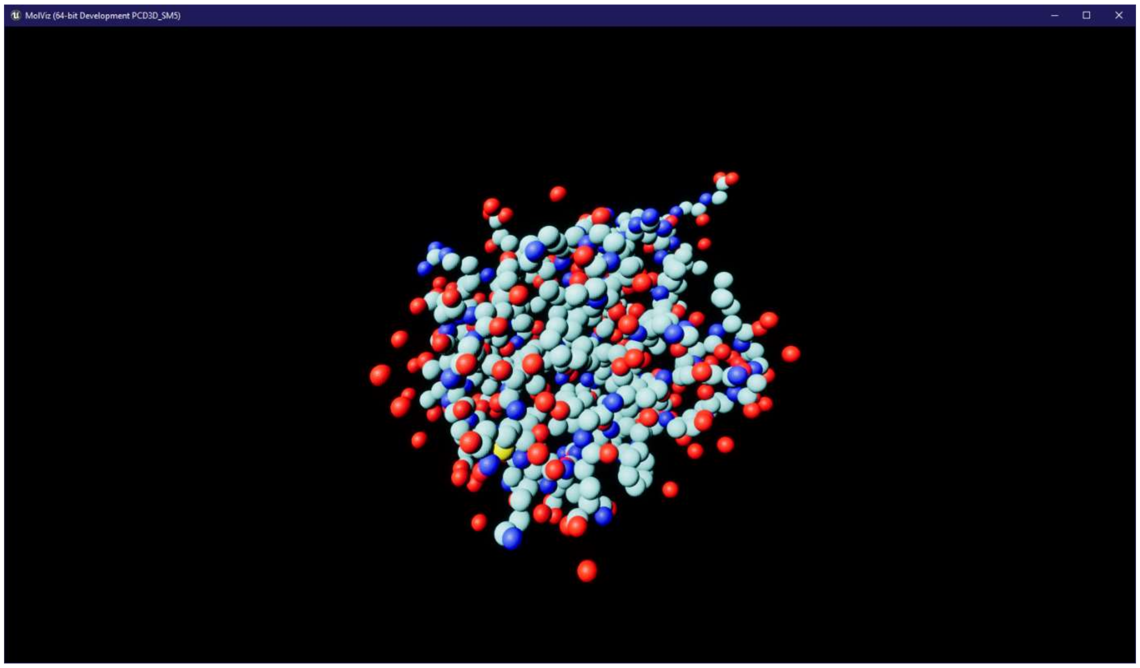
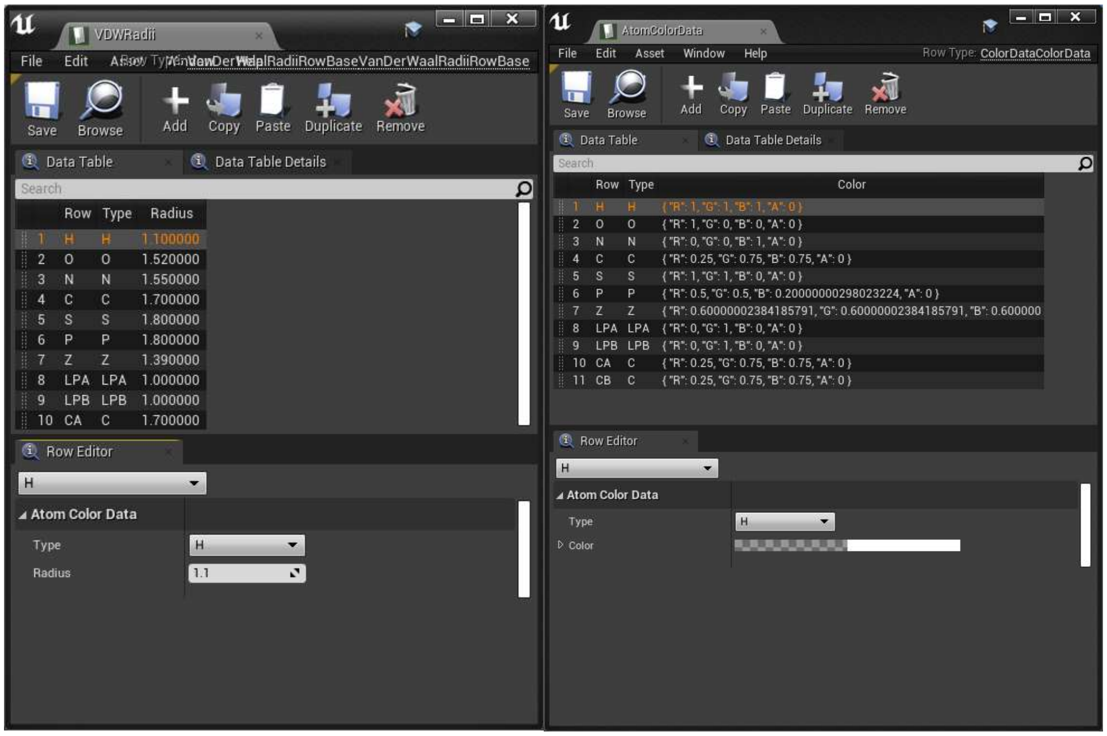


Figure 20 VDW representation (as generated by the software)

The VDW representation is the least complex one to do. It involves creating a sphere at the position of each atom in the molecule. However, it is not so simple as the radii of the atoms differ depending on the element. The VDW radii represent the space in which no other atom cannot enter; it is the closest another atom can be. Therefore, each atom takes up some amount of space and such a representation is referred to as space-filling. Once the molecule has been rendered the atoms will fill gaps when they are close enough.



	Row	Type	Radius
1	H	H	1.100000
2	O	O	1.520000
3	N	N	1.550000
4	C	C	1.700000
5	S	S	1.800000
6	P	P	1.800000
7	Z	Z	1.390000
8	LPA	LPA	1.000000
9	LPB	LPB	1.000000
10	CA	C	1.700000

H

Atom Color Data

Type: H

Radius: 1.1

	Row	Type	Color
1	H	H	{ "R": 1, "G": 1, "B": 1, "A": 0 }
2	O	O	{ "R": 1, "G": 0, "B": 0, "A": 0 }
3	N	N	{ "R": 0, "G": 0, "B": 1, "A": 0 }
4	C	C	{ "R": 0.25, "G": 0.75, "B": 0.75, "A": 0 }
5	S	S	{ "R": 1, "G": 1, "B": 0, "A": 0 }
6	P	P	{ "R": 0.5, "G": 0.5, "B": 0.20000000298023224, "A": 0 }
7	Z	Z	{ "R": 0.60000002384185791, "G": 0.60000002384185791, "B": 0.600000
8	LPA	LPA	{ "R": 0, "G": 1, "B": 0, "A": 0 }
9	LPB	LPB	{ "R": 0, "G": 1, "B": 0, "A": 0 }
10	CA	C	{ "R": 0.25, "G": 0.75, "B": 0.75, "A": 0 }
11	CB	C	{ "R": 0.25, "G": 0.75, "B": 0.75, "A": 0 }

H

Atom Color Data

Type: H

Color

Figure 21 Tables in UE4

Figure 21 Tables in UE4 shows what tabular data looks like in the UE4 editor. It populates a Map-like data structure which is loaded from disk at runtime. This can then be queried by the Row name (key value). This returns the row, which is represented as a struct; in this case the struct has two member variables, type and radius. The struct's members can then be accessed; here the Radius field is used to set the sphere's radius.

Another table is used to represent the colour data. This can be seen on the right-hand side of Figure 21. In this case, we do not have a simple float value but a 4D Vector to represent the Color. The struct in this case contains a Type field and a Color field which is accessed the same way as before. This is then used to set the color of the sphere depending on the element type.

4.3 Bonds & Atoms (CPK)

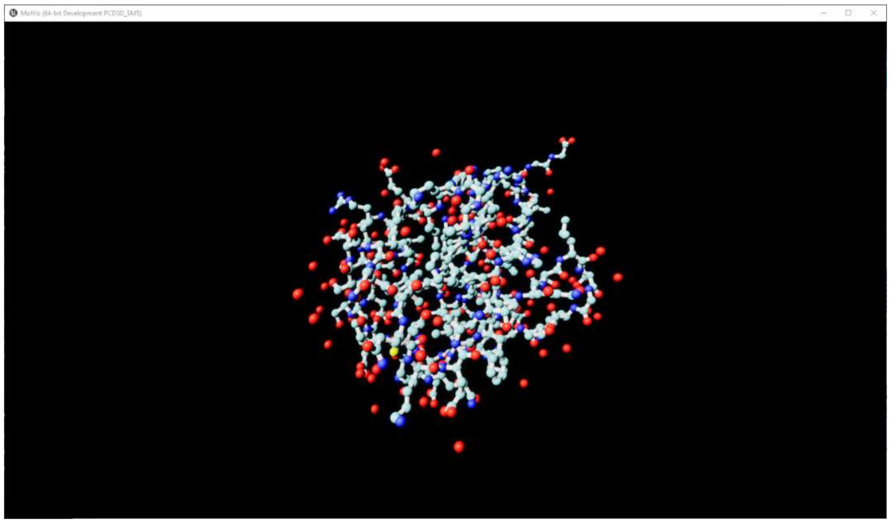


Figure 22 CPK representation (as generated by the software)

CPK refers to the scientists Robert Corey, Linus Pauling and Walter Koltun who invented the space filling model. In this context CPK actually refers specifically to the cylinders and spheres model. This is where a bond is represented as a cylinder extending between two bonded atoms and an atom is represented as a sphere, in this case with non-varying radii.

4.4 Tube

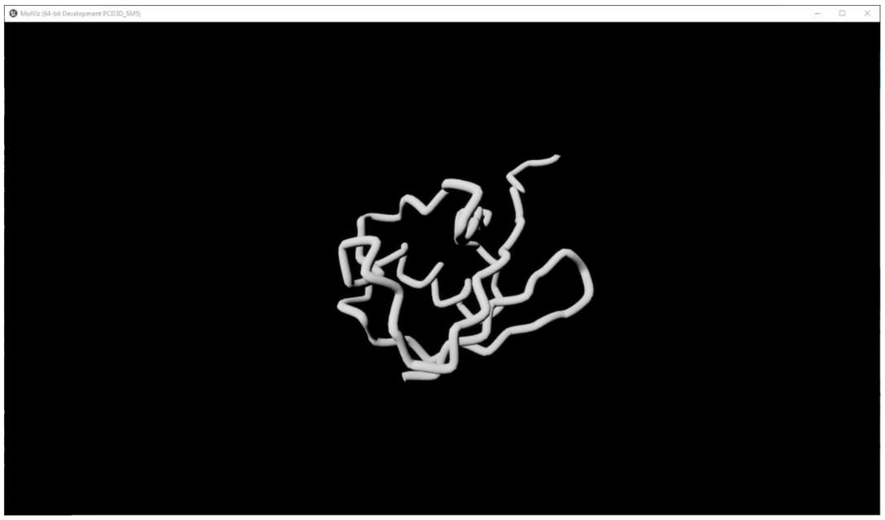


Figure 23 Tube representation (as generated by the software)

The Tube model draws a cylindrical structure similar to a tube that flows between the backbone atoms of the molecular structure. In the specific context of a protein, the molecule will have a chain of carbon atoms that are linked together. This is the backbone, and drawing a tube through these atoms helps visualise the shape of the protein that would have otherwise been hidden.

4.5 New Cartoon

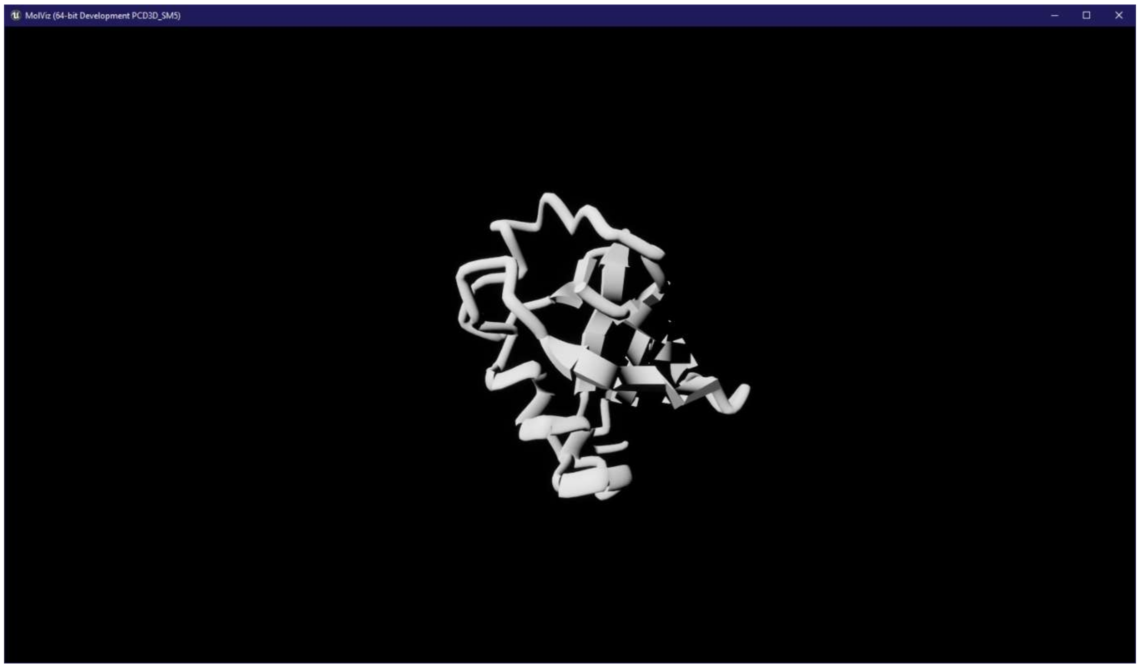


Figure 24 New Cartoon representation (as generated by the software)

The New Cartoon method is similar to the Tube representation. However, instead of just using a cylinder along the backbone, depending on the Secondary Structure of the residue, it applies a flat sheet or arrow in its place. The secondary structure of a protein refers to the shape of the backbone caused by how the atoms attract each other. The Alpha-Helix is represented as a flattened out cylinder and the Beta-Sheet as a rectangle finished with an arrowhead.

An issue that shows up in the above image is the jagged appearance near the edges of individual components in the mesh. This is due to the rotations not being properly set. Although this does not matter so much since the information is being correctly displayed, eliminating it would improve the user experience.

4.6 Meshes

The meshes for the Atom and Bond as well as the New Cartoon were made in Blender, an open-source 3D tool suite.

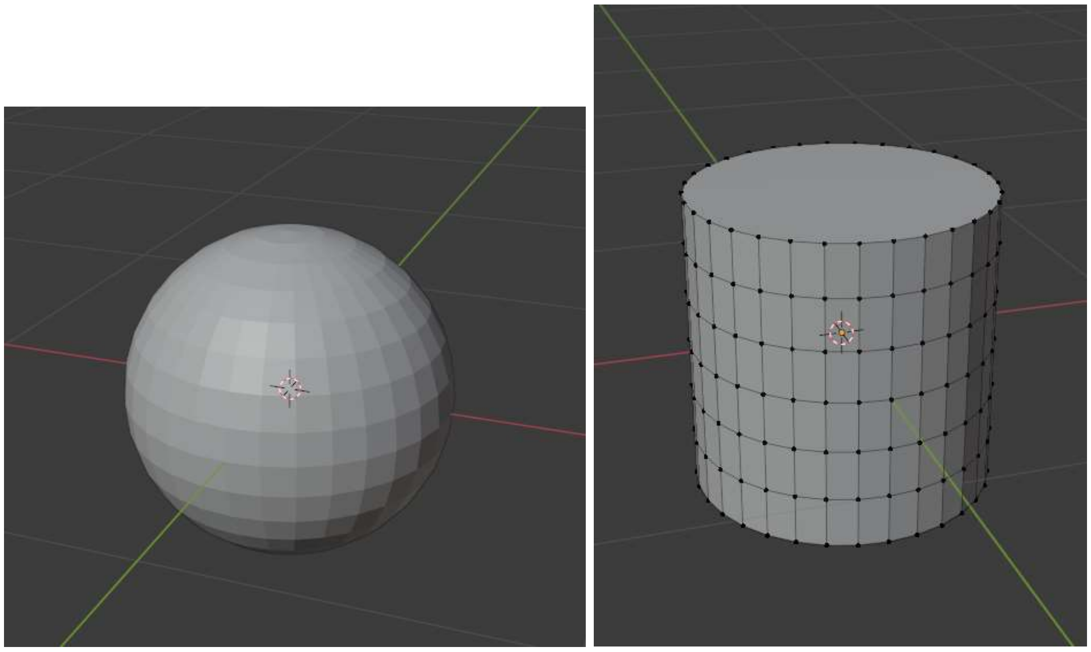


Figure 25 Sphere / Atom and Cylinder **/** Bond meshes

The atoms and bonds were not difficult to make; they were all spheres and cylinders of unit length and width. For the Tube it was the same case, the mesh for the bond was reused.

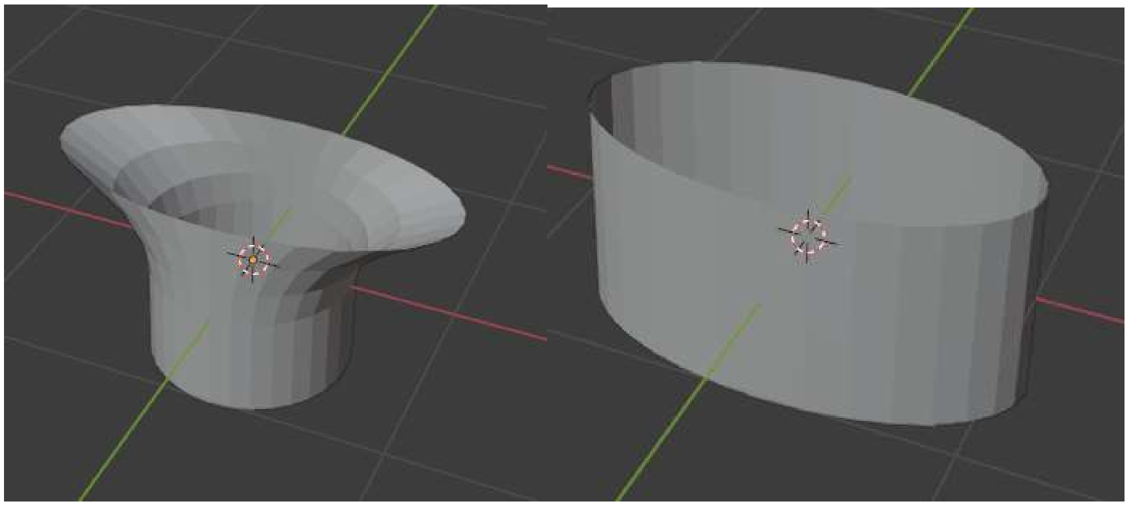


Figure 26 Helix Start / End and Helix Middle

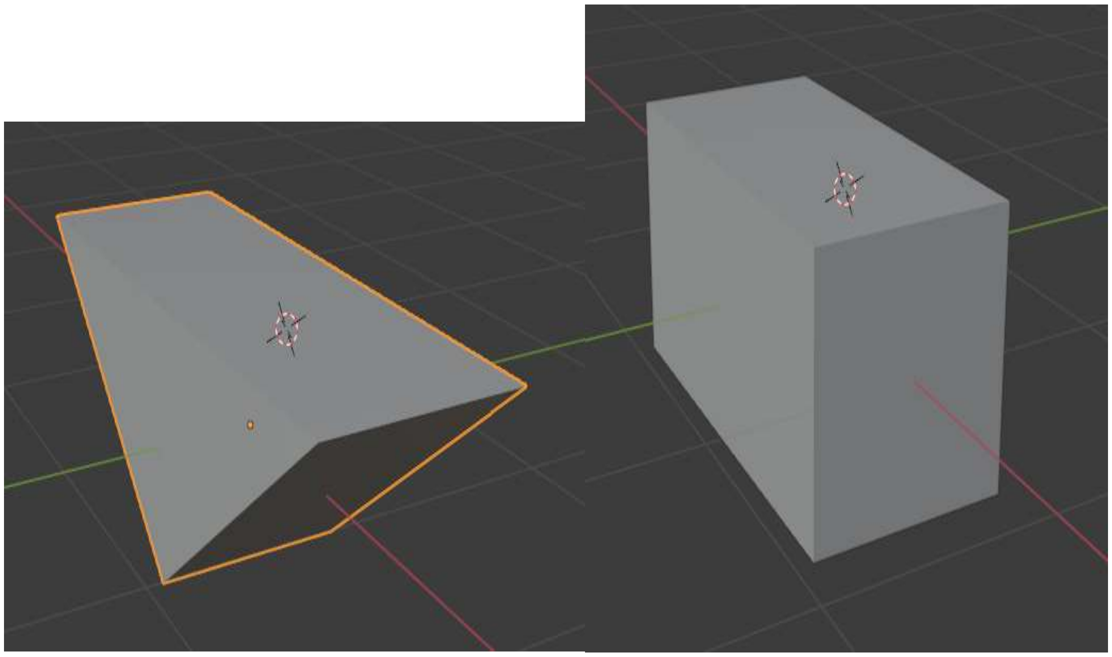


Figure 27 Sheet End and Middle / Start

The New Cartoon however, is more complicated. The Helix was made using 2 parts, the start/end and middle meshes. By doing this, a tapered effect was achieved where the Helix starts from a cylinder and ends by going back to a cylinder. The Beta-Sheet is more abrupt; so it does not need as many meshes as the Alpha-Helix. This is an approach highlighted by Matus et al. (44).

4.6.1 Automatic LOD generation

In order to help improve performance, UE4’s automatic LOD generation is used. Since the tool is automated and doesn’t understand what an appropriate simplification of the mesh could be, the LODs, after generation were removed depending on the quality.

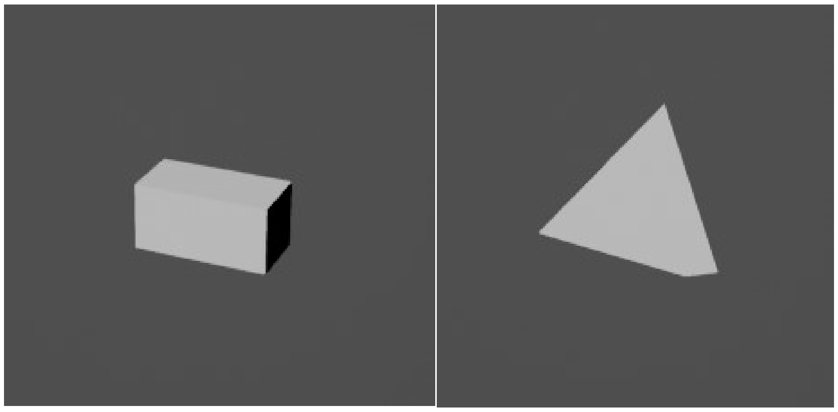


Figure 28 LOD levels for the Beta-Sheet start component

On the left of Figure 28 the highest quality of the Beta-Sheet component can be seen and on the right the lowest quality version can be seen. The left has 24 vertices and the right 6 vertices. This shows that the automatic LOD generation cannot simplify this meaningfully past a certain number of levels. Therefore, these levels were removed.

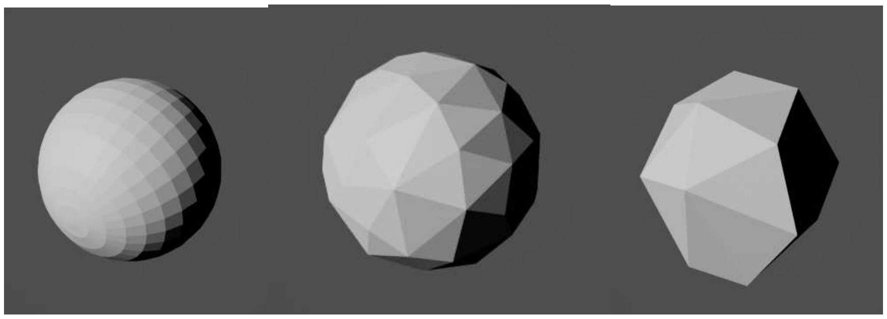


Figure 29 Sphere LOD (High, Medium, Low from left to right)

In Figure 29 LOD level 0, 3, and 5 can be seen for the Sphere mesh. The LOD level of the Sphere will be increased as the amount of screen space that the sphere takes up is reduced, such as when the user moves further away from a sphere.

4.7 Algorithms

In order to create the CPK, Tube and New Cartoon representations, special algorithms are required to first detect the bonded atoms and then to detect the backbone atoms.

```
Algorithm 1 CreateBonds
1. for all atoms A in molecule
2.     for all atoms B in molecule
3.         if A == B skip
4.             if A or B is Hydrogen
5.                 Compute inter atom squared distance SqaredDistance
6.                 if 0.16 <= SquaredDistance and SquaredDistance <= 1.44
7.                     add Bond between A and B
8.         else
9.             Compute inter atom squared distance SqaredDistance
10.            if 0.16 <= SquaredDistance and SquaredDistance <= 3.61
11.                add Bond between A and B
```

Figure 30 Algorithm to detect bonds

The algorithm above shows how to create the bond data needed to render the Bonds and Atoms (CPK) representation. The algorithm performs a double “for" loop to iterate over every atom and compare it to every other atom. If the inter atom distance is within a threshold they are bonded together. This causes a slight issue; however, when the protein is in a warped position, such that two normally unconnected parts are within the distance required to make a bond. This would be incorrect and may arise when simulation data is dynamic.

```
Algorithm 2 FindBackBone
1. for all chains C
2.     for all residues R in chain C
3.         for all atoms A in R
4.             if A.type = CA
5.                 R.CA = A
6.             if A.type = C
7.                 R.C = A
8.             if A.type = N
9.                 R.N = A
10.            if A.type = O
11.                R.O = A
```

Figure 31 Algorithm to detect backbone

The second important algorithm shown above is to detect the backbone atoms. This algorithm is much simpler as the structure of a protein and the available information in the PDB is exploited. It is known that there is a chain of carbon atoms that are linked together, with a nitrogen and oxygen atom as well. The names of these elements are contained in each ATOM record and so can be simply matched up. There is always 1 of each in a residue that makes up that part of the backbone.

In order to get the information for the secondary structure rendering, another application called STRIDE was used. This application was provided in binary form with VMD.

By copying this binary and packaging it with the MolViz application it was possible to run it as a child process. Providing STRIDE with the path to the opened PDB results in a command line output that can be read in via stdout. The output then needs parsing as described before. Following parsing, the residues are ascribed a secondary structure type. The tube algorithm is used and modified slightly to use the correct mesh when iterating through the backbone atoms, to check what the secondary structure type is.

4.8 Splines

Hermite splines are the basis for all spline shapes in UE4. They were used to create the Tube representation and the New Cartoon representation.

The USplineMeshComponent allows to attach a mesh and have it deform and repeat along an underlying Hermite spline. The main focus is to create the spline such that the mesh takes on the desired shape.

Hermite splines feature a start and endpoint as well as a start and end velocity or tangent vector. This creates a smoothly interpolating curve that goes between the points curving in the direction of the supplied tangent vectors. In addition, it also supports defining a start and end roll such that meshes that are not symmetrical can be aligned correctly.

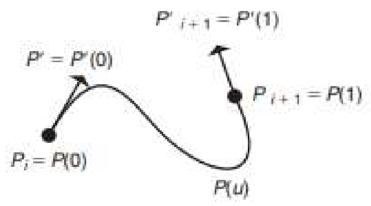


Figure 32 Hermite Spline **(45)**

The Hermite spline creates a cubic polynomial curve and can be expressed as shown below, in matrix multiplication form:

$$U^T = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}$$

$$M = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+1} \\ P'_i \\ P'_{i+1} \end{bmatrix}$$

Figure 33 Hermite spline in matrix notation **(45)**

In the image above, the terms U, M and B can be formulated as P(u) = UT * M * B. This will return the location of a point u along the Hermite curve. Furthermore, these Hermite splines can be connected together in a smooth manner, by making the first tangent vector of subsequent spline the end tangent vector of the current spline segment (45).

4.9 Material

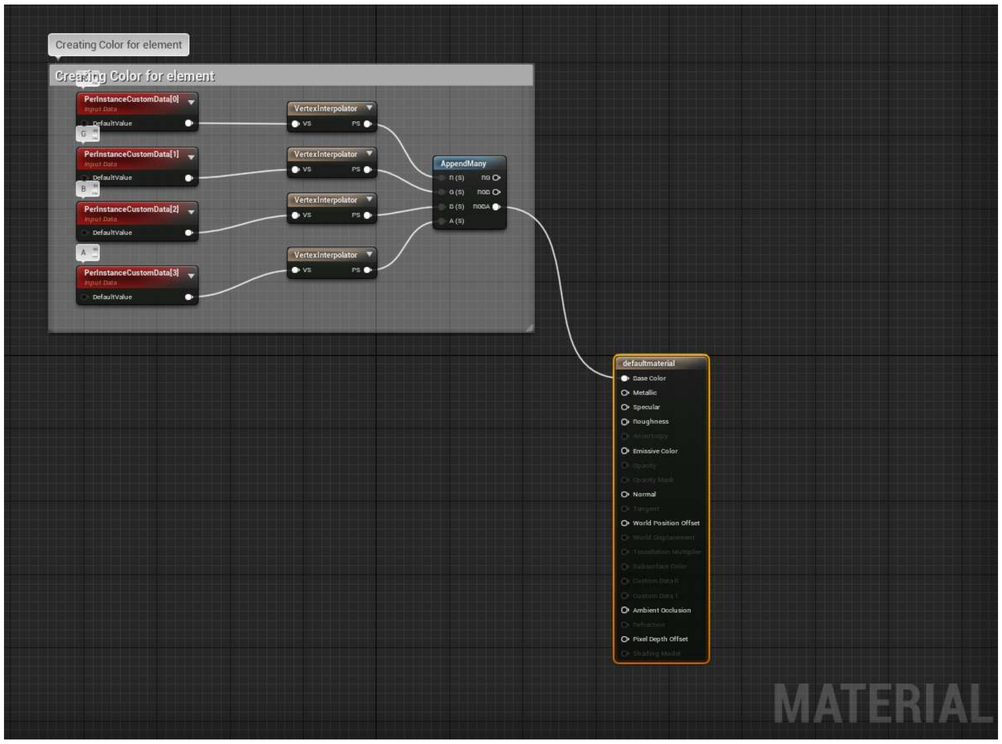


Figure 34 Material asset for colouring atoms

This Material Blueprint (Figure 34) is fairly simple. Each instance of the mesh stores 4 float values, for R, G, B and A. These are read in by the Material in the PerInstanceCustomData array, which ranges from 0 – 3. Each value is put into a VertexInterpolator node; this is necessary since the values may not contain interpolated data, which is needed for areas between vertices. Once this is done, they are fed into an AppendMany node, which can be used to make a colour; in this case RGBA is the output which is then fed into the input, BaseColor of the material.

4.10 Mouse and Keyboard controls

Keyboard controls are minimalist and simple. The R, T, and S keys control switching between Rotation, Translation and Scaling transformation modes. This is internally represented as an Enumerated class.

The enumerated class is then used in a switch in the main mouse movement routine to control the type of transformation specified. This is handled in the AMousePlayerController class which links action mappings defined in the Engine to the low-level C++ methods used to control the Protein’s transformation.

Action mappings tie a unique name to a hardware input device. This name can then be accessed in code, allowing a developer to link a method to it, essentially a call back function. This will get triggered whenever the specified input mapping is invoked. UE4 handles this internally, so the developer only needs to create the routine called in response to the triggering of the input mapping.

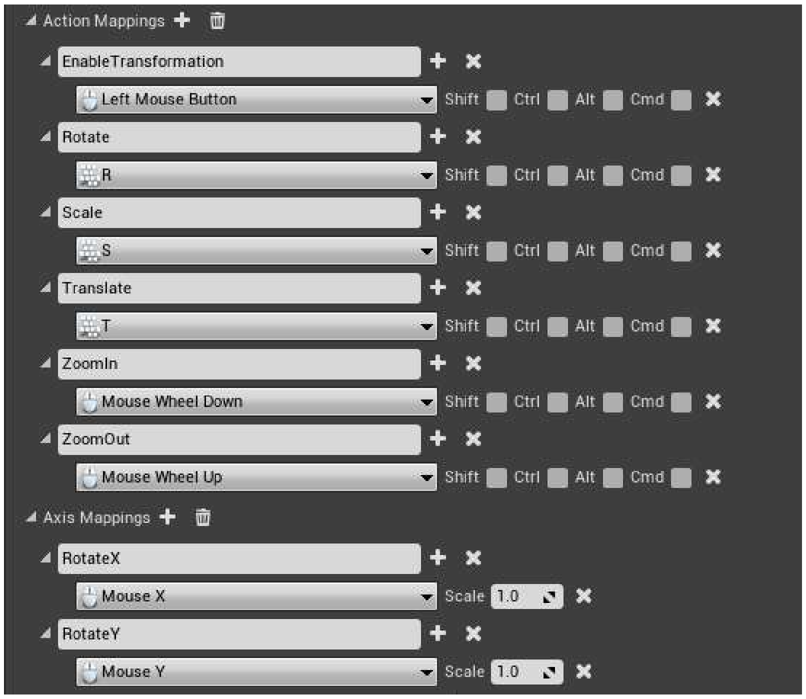

```
Action Mappings
  EnableTransformation
    Left Mouse Button    Shift  Ctrl  Alt  Cmd
  Rotate
    R                    Shift  Ctrl  Alt  Cmd
  Scale
    S                    Shift  Ctrl  Alt  Cmd
  Translate
    T                    Shift  Ctrl  Alt  Cmd
  ZoomIn
    Mouse Wheel Down     Shift  Ctrl  Alt  Cmd
  ZoomOut
    Mouse Wheel Up       Shift  Ctrl  Alt  Cmd
Axis Mappings
  RotateX
    Mouse X              Scale 1.0
  RotateY
    Mouse Y              Scale 1.0
```

Figure 35 Action Mappings

In Figure 35, the action mappings created can be seen. These are delegates that functions can be bound to and so C++ routines were linked to them at startup. The most complex one is the routine to rotate the molecule about its centre of mass.

The rotation makes use of an arc ball controller. This essentially is similar to rotating a sphere about its centre and applying the amount and direction which it has rotated to the object being controlled.

First the mouse position needs to be transformed from a range of 0 – width and 0 – height to -1 to 1. The next step is to constrain the scaled mouse position to the surface of the sphere. Once this is implemented, the last and current mouse positions need to be kept track of. By computing the sphere positions of the last and current mouse positions, two vectors are formed from the centre of the sphere to those points. A third vector can be computed from the cross product of the two previously computed. This is the same as an axis and angle and can be converted into a quaternion that represents this rotation. This rotation needs to be tracked as well, otherwise the rotation will snap to identity before rotating again (46).

4.11 Automation Testing

Finally, the last system that needed implementing was the benchmarking puppeteer and automation program.

Unreal Engine provides an Automation tool called Gauntlet that will spin up a build of the application and allow the user to run a profiler, collecting artefacts along the way and uploading them when the test completes (47).

In order to make use of this system, one must code an automation program in C# as this is what the tool is written in. This program will be tasked with creating an instance of the puppeteer, and checking to see if the test has been completed or not and processing all artefacts that have been produced throughout the lifetime of the test (47).

The puppeteer is tasked with making sure that the test’s prerequisite conditions are met and that it starts executing correctly. It also reports when the test has been completed. The puppeteer is called a Gauntlet Controller and resides in the main application project; it is written in C++ and has what is normally accessible to instances in the application (47).

The controller in this case makes use of a tool called the CSV profiler. This tool will collect all internal stats in every frame and when ordered to stop recording, will write it out to a CSV file. The reason it writes only when ordered to stop recording is to avoid incurring the cost of IO during the running of the application, as this will introduce noise to the stats. The CSV profiler was created such that it would be as low cost as possible during runtime (48).

Finally, when this CSV has been recorded and written out to memory, it is considered as one the artefacts produced by the test. It is then processed by the Gauntlet program by running a tool to convert the CSV into an HTML and SVG interactive report. This tool is also provided but needs to be compiled from source.

All this brought together by the Unreal Automation Tool, which is a batch script that will run an executable with the appropriate Gauntlet test as specified when invoked.

The Gauntlet program, and the Gauntlet Controller were adapted from the “HORU game” blogpost on performance testing using Gauntlet (49)

4.12 Results

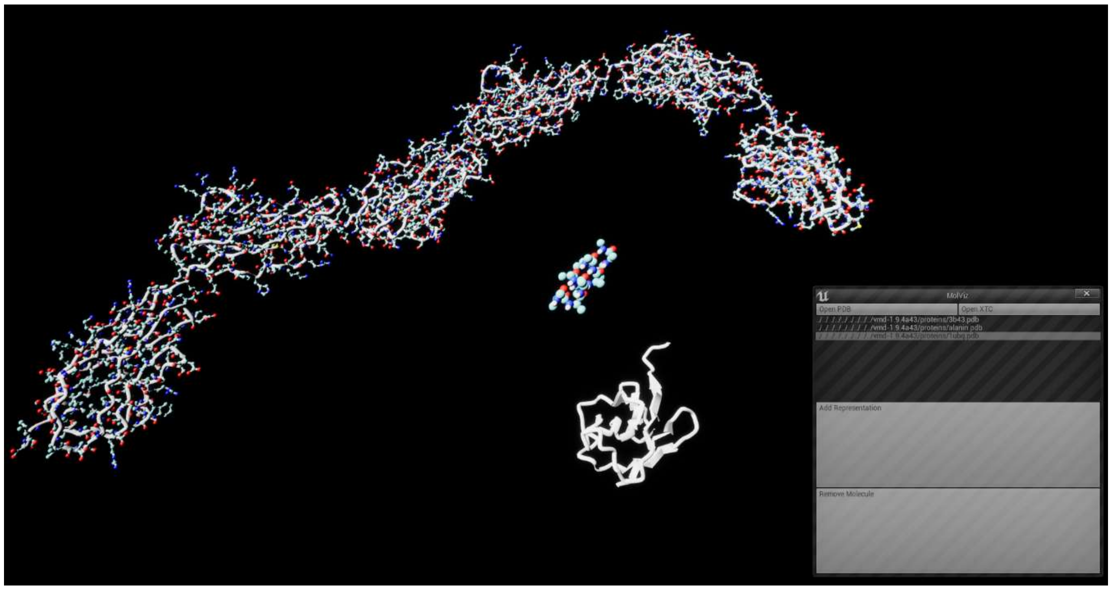

MolViz
Open PDB
Open XTC
/ / / / / / / / / /vmd-1.9.4a43/proteins/3b43.pdb
/ / / / / / / / / /vmd-1.9.4a43/proteins/alanin.pdb
/ / / / / / / / / /vmd-1.9.4a43/proteins/1ubq.pdb
Add Representation
Remove Molecule

Figure 36 Implemented software

In Figure 36 the final implemented system can be seen. The system is showing three proteins rendered in different representations. On the left, CPK and tube are being used to render a long chain of Titin, next to it is a small Alanine protein rendered using VDW and next to that is the secondary structure of Ubiquitin rendered in New Cartoon. The user interface handles switching between controlling multiple molecules independently and configuring their representations.

5 Testing

Two forms of testing were used to assess the quality of the produced software: Usability testing and Performance testing.

5.1 Usability testing

5.1.1 Methodology

A usability session is where the user would use the application and fill out a short questionnaire designed to determine the accuracy and quality of the software. The target users are experts in the field of Molecular Dynamics at the University of Leeds.

Due to Covid-19, in-person usability sessions were impossible to organise, and getting a build for Linux and Mac for university administrated computers proved to be difficult since the prerequisite software had not been installed. So a session was hosted on Microsoft Teams with a screen share showing how the application worked. The session was recorded and transcribed and can be found in Appendix A.1.3.

The script provided in Appendix C.2 was followed. It was designed such that all the features that have been implemented would be showcased. The interviewee would be allowed to interject at any point and direct the script in a particular direction once step 5 in the script was reached.

5.1.2 Results

The various representations that have been implemented were reported as being accurate. Some points were raised regarding the New Cartoon representation and the smoothness of the mesh. These were not major issues but should be implemented to aid in analysing the information.

5.1.2.1 Feedback Summary

One user recommended that Python scripting, GRO files and Coarse-Grained representations should be added in the future.

One of the shortcomings of VMD is the use of an outdated and deprecated language to write plugins. This should have been moved to Python as this is now a dominant language in the scientific field.

GRO files are a popular format that is well supported. It is created by a program available only on Linux called “GROMACS”. This is a simulation software and outputs animation data as well as structural data like a PDB file. This should be supported since many users in this Biophysics/Structural biology field are Linux users.

Furthermore, support for a Linux build should be incorporated, as well as for a Mac build. This would allow for reading XTC files that require the use of specialised routines available only on a Linux operating system.

Finally, coarse-grained representations should be implemented. This is with reference to the representation styles; currently the application only supports atomistic representations, meaning that each atom is rendered. A coarse-grained representation only shows atoms that are necessary, abstracting other atoms into larger groups. This could also help with performance as it would cut down on the amount of geometry needed to be rendered.

The other user mentioned that because the software focuses on creating visualisations in VR, the New Cartoon should be smoothed out so it becomes more aesthetically pleasing. It was also mentioned that the ability to create specific representations should be allowed, i.e. representing certain residues of a protein as VDW or a specific chain as CPK.

In order to smoothen the New Cartoon representation, the rotations at the start and end of each spline segment must be set correctly. Since each spline travels from one alpha carbon to the next, these matrices are created by identifying the previous alpha carbon and the next alpha carbon. A vector can be formed between these two alpha carbons making a vector T. A second vector B can be formed between the previous alpha carbon and the current alpha carbon. It is now possible to create a vector N perpendicular to T by taking the cross product of B and T. At this point, the coordinate frame is almost complete, and the final step is to simply take a cross product of T and N to get the last axis of the frame. This method is highlighted in detail in the paper by Weber et al. (50).

It was also stressed that there should be flexibility in the user interface, such as allowing for a desktop interface that can then be switched to a VR based interface. An example scheme would be to have all UI windows related to creating representations on the Desktop and then having playback related User interface in both. So when the user is viewing the Protein in VR they at least are able to scrub through the animation frames.

5.2 Performance testing

5.2.1 Methodology

The objective of the performance test was to push the limits of the application, to ensure it is performant when ported to VR.

The performance test was implemented as a separate application, with automated loading of a PDB file. The application will load the PDB file until 100,000 atoms were attained.

This was chosen due to the maximum limit of atoms in a PDB file being 99,999 atoms. The PDB file chosen for this benchmark contains 6 chains of the Titin protein. Titin is the largest protein and 6 chains of it are at 4412 atoms. This was chosen so that less time was spent loading the model during the benchmark. Furthermore, the first Titin chain loaded in the level will rotate 360 degrees continuously throughout the benchmark to add a dynamic element to rendering the scene. This will recreate the effect of a user rotating the molecule in a normal session.

Statistics were collected for every aspect of the Engine during this benchmark testing. The important aspects that required attention were the game thread and the render thread performance. When analysing this data, it is vital to make sure that the performance scales as atoms are added, since when porting to VR the timing requirements are much stricter.

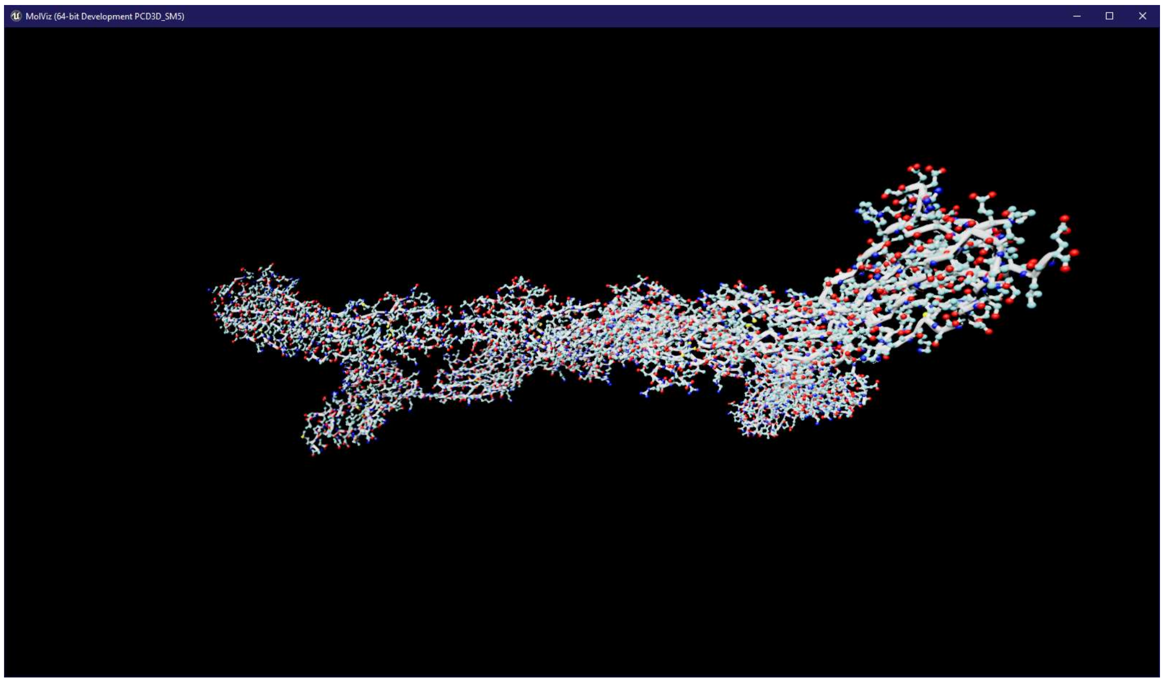
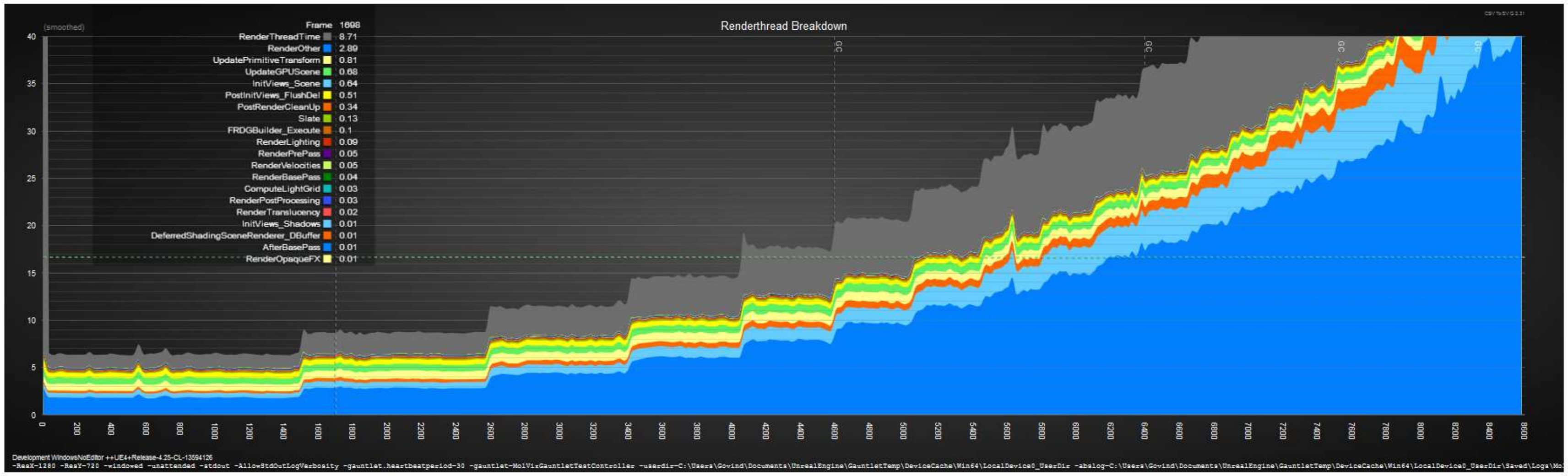
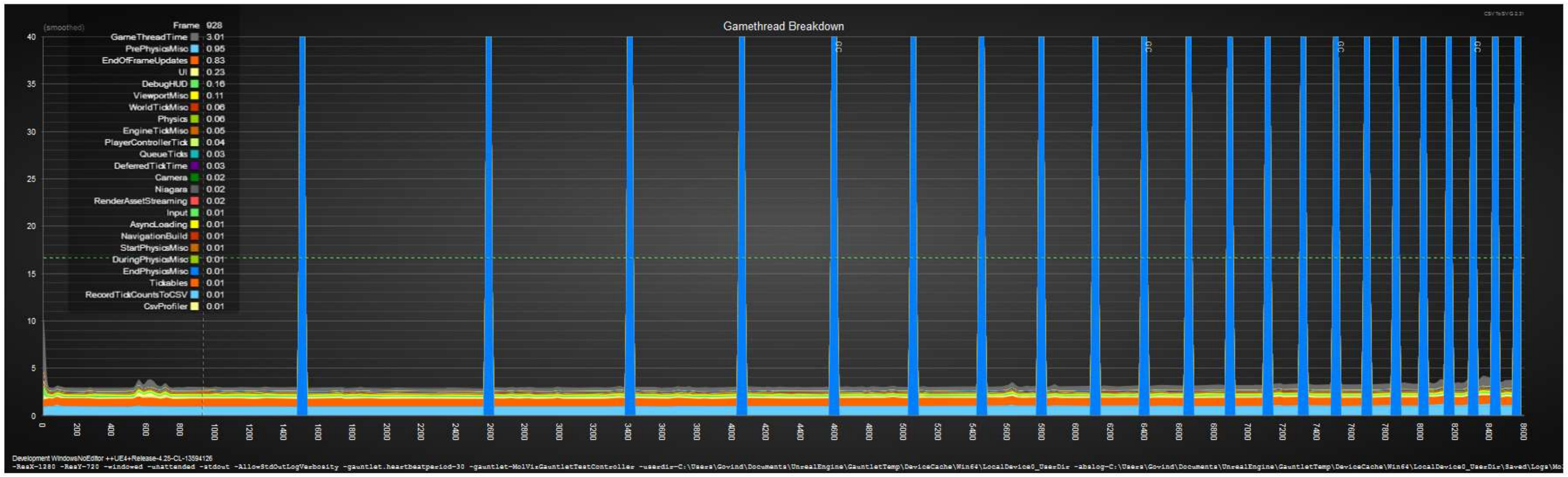


Figure 37 Benchmark program with two files of Titin chains loaded



```
Development WindowsNoEditor ++UE4+Release-4.25-CL-13594126
-ResX-1280 -ResY-720 -windowed -unattended -stdout -AllowStdOutLogVerbosity -gauntlet.heartbeatperiod=30 -gauntlet=MolVizGauntletTestController -userdir=C:\Users\Govind\Documents\UnrealEngine\GauntletTemp\DeviceCache\Win64\LocalDevice0_UserDir -abslog=C:\Users\Govind\Documents\UnrealEngine\GauntletTemp\DeviceCache\Win64\LocalDevice0_UserDir\Saved\Logs\Mo
```

Figure 38 Render Thread break down (x-axis: ms, y-axis: frame number, occlusion culling enabled) See Appendix A.1 for full report.



```
Development WindowsNoEditor ++UE4+Release-4.25-CL-13594126
-ResX-1280 -ResY-720 -windowed -unattended -stdout -AllowStdOutLogVerbosity -gauntlet.heartbeatperiod=30 -gauntlet=MolVizGauntletTestController -userdir=C:\Users\Govind\Documents\UnrealEngine\GauntletTemp\DeviceCache\Win64\LocalDevice0_UserDir -abslog=C:\Users\Govind\Documents\UnrealEngine\GauntletTemp\DeviceCache\Win64\LocalDevice0_UserDir\Saved\Logs\Mo
```

Figure 39 Game Thread break down (x-axis: ms, y-axis: frame number, occlusion culling enabled) See Appendix A.1 for full report

5.2.2 Results

5.2.2.1 Understanding the results

The following breakdown explains the significance of the game thread and render thread. For a full breakdown of the performance report see Appendix C.3.

5.2.2.1.1 Gamethread Breakdown

The game thread is where UE4 computes physics, updates actor positions and runs the tick function of all the actors that can tick in the level, carries out async IO, processes input from HID controllers and does other non-rendering related operations. The game thread performance should be such that it is ahead of the render thread by 1 or 2 frames at all times; by analysing this graph it is possible to optimize the performance of the game thread so it remains ahead of the render thread (51).

Although not labelled on the graph, the unit of measurement for the y-axis is in milliseconds and the unit of measurement on the x-axis is the frame number.

5.2.2.1.2 Renderthread Breakdown

The render thread is where all the rendering occurs; this means draw calls are batched/merged and prepared for the appropriate graphics API to process. Important optimizations occur here such as occlusion culling and frustum culling. The render thread also sets up the appropriate draw calls for all post-processing effects such as bloom, ambient occlusion, depth of field and motion blur (51).

Although not labelled on the graph, the unit of measurement for the y-axis is in milliseconds and the unit of measurement on the x-axis is the frame number.

5.2.2.2 Analysis of the Results

The game thread remains at around 3 – 4ms processing time, no matter how much the amount of data increases. The benchmark started at around 4400 atoms and ended at about 100,000 atoms, not including bonds. This shows that the scalability of the game thread is viable once larger molecules are loaded. This is expected since there is very little logic that needs to be executed in the game thread. The only tasks involve parsing input controls.

There is however one issue with the game thread. Whenever a user loads a new molecule from a PDB, there is a spike in latency in the game thread. This is due to the single threaded nature of the reading code used to parse the PDB. It should be moved to another thread such that there is no spike. This will not only improve user experience on the desktop version of the application but also substantially improve the usability of the VR version of the app. One of the main criteria to making the user more comfortable in VR, as discussed in section 3.6.1, is having short load times , which can be achieved by multithreading the reading.

The render thread shows a steady increase as the Titin chains are being added. This goes to show that in fact VR may have some issues rendering some of the larger molecules and maybe even when rendering the animation data.

In section 3.5, the InitViews method is mentioned where the FPrimitiveProxy creates FBatchDescrption structs. In Figure 38 the InitViews_Scene method performance can be seen, and as the number of Titin chains increases in the level so does the time needed to create the FBatchDescriptions. This is expected as there is more data to represent in the render thread.

The number of draw calls fluctuates but generally is on an upward trend. This should not be the case since the draw calls should be batched with the use of the UInstancedMeshComponent. Going further into this issue, by using RenderDoc, a third-party software to diagnose bugs in rendering pipelines, the number of draw calls increase as the USplineMeshComponents are added to the scene. There are multiple draw calls for rendering each cylinder of the USplineMeshComponent as opposed to the two single draw calls for the Atoms (Spheres) and Bonds (Cylinders) of the CPK representation. The exported draw calls can be found in Appendix C.4. This shows that the UInstancedMeshComponent correctly optimized the draw calls as opposed to the USplineMeshComponent. This contributes to the poor performance of the render thread. There is however a steady increase in RenderOther. This steady increase could be due to increasing processing time required to cull occluded primitives.

Occlusion culling is the process by which geometry is eliminated from rendering based on its visibility from the position of the camera. This is different from frustum culling where geometry outside the view of the camera is culled. This method culls geometry hidden by other geometry in the view of the camera. This is done by occlusion queries where the GPU will return the number of visible pixels. An occlusion query would be done per Actor and then depending on the result, the Actor is rendered or not, if its returned visible pixel count is greater than 0. The Occlusion query will make use of a Bounding Box to do the test. A Bounding Box is a simple collision volume that may exceed the space of the geometry or be within it (52).

After a certain point this technique will underperform due to a large amount of small occluding shapes. This test is usually done to reduce the overhead of rendering many primitives; however, as a result, it syncs the GPU and CPU to a certain extent, slowing down the time it takes to render.

By running the benchmark with occlusion culling disabled, there was perceivable difference in the render thread graphs produced by UE4. This could be due to the already large impact on rendering caused by the USplineMeshComponent. The performance report with occlusion culling disabled is provided in Appendix A.1.

After adding around 17,648 atoms, the render thread drops to 13 ms. This is at the edge of what is acceptable as per the Oculus Rift and Unreal Engine best practices for developing VR applications.

5.2.3 Improvements

Removing Occlusion culling will render lots of detail unnecessarily. This is also not efficient and can be optimized. In the usability session, a coarse-grained representation was suggested. This will reduce the overhead by reducing the number of meshes that will be needed to draw and also simplify the process of occlusion culling.

To improve the rendering of the USplineMeshComponents, they should be merged into a single USplineMeshComponent (single mesh). UE4 provides some utilities to do so but may cause issues with the variables per primitive that have been used in the Material to render the colour of the mesh. This would improve the performance by minimising the number of draw calls that need to be prepared in the draw thread and finally improve render thread by optimized batching of spline meshes.

6 Conclusion

6.1 Project Evaluation

Table 3 lists the project objectives and their status.

Table 3 Project objectives

Objective	Status
Replication of the visualisation of protein structures as rendered by an existing tool	Completed
Load proteins from a Protein Data Bank (PDB) file to later represent using the VDW and New Cartoon representations	Completed
Visualisation of protein structure files using Van der Waals (VDW)	Completed
Visualisation of protein structure using New Cartoon representation	Completed
Animation of the molecular structure from captured dynamic simulation data, read in from an XTC file format	Incomplete
Porting software to Virtual Reality	Incomplete

The software produced - MolViz - satisfied almost all of the basic functionalities and was able to create accurate visualisations of complex protein structures. The user interface provided flexibility and was easy for users to understand. The software showed good desktop scalability as it was able to render 100,000 atoms with a relatively good frame rate. It was shown that it would be possible to extend quite easily to VR in the future and would scale relatively well, for molecules up to approximately 17,648 atoms.

The software was not able to create specialised representations for subsets of residues in the protein. It did not feature different colouring schemes based on the properties of atoms. It also did not consider all the possible edge cases for PDB file formats and so would fail silently or outright crash. The software does not process other simulation data formats such as XTC or DCD files. In addition, it did not feature Virtual Reality capabilities and also showed that for extremely large molecules, it would suffer a performance hit.

For the large number of users who are on Linux, UE4, although supporting Linux, does not allow for VR to be correctly ported to Linux.

6.2 Experience using UE4

Unreal Engine greatly simplified developing a visualization application by providing the basic features needed to do so. The editor was very powerful, allowing to quickly iterate meshes and new features. By using C++, code performance was achieved but as a consequence, certain bugs required completely wiping cached artefacts and recompiling from scratch, which wasted significant amounts of time. Another major issue was the lack of support and poor documentation for Mac and Linux, which caused issues with producing builds for those platforms.

6.3 Challenges Faced

Throughout the development of the software, many challenges were faced due to strict timing requirements and the lockdown caused by Covid-19. Having no prior experience in Molecular Dynamics and any related field, and a rusty knowledge of basic chemistry, a lot of time was invested in learning this. However, having prior experience in UE4, time was saved by not having to relearn the editor interface and other basic features. Furthermore, with limited experience in writing unit tests and automated benchmark tests, time was spent learning the necessary tooling by reading unofficial posts and reading the source code due to poor official documentation.

Despite developing software code faster in UE4 the other factors slowed down progress to such an extent that there was not much flexibility in planning and in the end resulted in missed objectives. No VR capabilities were implemented since the VR lab was not accessible for testing purposes. Once such restrictions are removed work on these important features may be resumed, along with much needed improvements and new features.

6.4 Future improvements

The following improvements are suggested when developing a fully-fledged software solution. Using Unreal Engine would be a non-starter as it removes stable support for important target platforms such as Linux. Furthermore, using the official Oculus Rift SDK which UE4 would prefer to do, would cause the software to be no longer Linux targetable. The alternative is to use Steam VR / OpenVR libraries which UE4 supports and work around the known issues. Therefore it is highly recommended that future implementations should be written from scratch in C++ or another high-performance language with a compiler that supports targeting Linux and Mac as well as Windows and also using LibXdr as an open-source cross kit VR library.

Important features as suggested previously should be prioritised, such as being able to create highly specific representations. VR should be supported first. Instead of creating a desktop visualiser that can be then be enabled in VR, it should be the other way around.

Other features that should be worked in as suggested are the Python scripting environment. Coarse grained representations should be implemented, for added clarity and performance in VR and support for GRO files and Linux operating system.

6.5 Summary

The software produced gives an idea of how such a solution could be implemented. It shows that UE4 is an interesting option to create such software but, in the end, it falls short due to the severe limitation in the number of stable target platforms. Looking towards the future of the field, VR molecular visualisations will play a big part in driving discoveries, as its intuitiveness is unparalleled. However, the field of molecular dynamics poses interesting challenges from a rendering standpoint that need to be tackled in order to create a smooth experience when viewing highly complex and interactive molecular structures. This paper provides some suggestions.

7 References

1. Emiliano , Ippoliti. *What is Molecular Dynamics?* s.l. : German Research School for Simulation Sciences, 2011.

2. *Molecular dynamics simulation.* Dror, Ron . Stanford : s.n., 2019. CS/CME/BioE/Biophys/BMI 279.

3. Theoretical and Computation Biophysics Group. VMD. *Theoretical and Computation Biophysics Group.* [Online] University of Illinois At Urbana-Chamapaign. [Cited: 10 August 2020.] https://www.ks.uiuc.edu/Research/vmd/.

4. Schrödinger. PyMOL by Schrödinger. *PyMol.* [Online] Schrödinger. [Cited: 10 August 2020.] https://pymol.org/2/.

5. Matthews, David. Virtual-reality applications give science a new dimension. *nature.* [Online] nature, 30 April 2018. [Cited: 10 August 2020.] https://www.nature.com/articles/d41586-018-04997-2.

6. Eichacker, Lutz . *PaleBlue Molecular VR.* Pale Blue, 2017.

7. O'Connor, Michael B, et al. *Interactive molecular dynamics in virtual reality from quantum chemistry to drug binding: An open-source multi-person framework.* s.l. : The Journal of Chemical Physics, 2019.

8. Alberts, Bruce, et al. Proteins. *Molecular Biology of the Cell.* s.l. : Taylor & Francis Group, 2008.

9. Theoretical and Computational Biohysics. VDW. *VMD Visual Molecular Dynamics.* [Online] University of Illinois. [Cited: 10 August 2020.] https://www.ks.uiuc.edu/Research/vmd/vmd-1.7.1/ug/node58.html.

10. Theoretical and Computational Biophysics Group. CPK. *VMD Visual Moleuclar Dynamics.* [Online] University of Illinois. [Cited: 10 August 2020.] https://www.ks.uiuc.edu/Research/vmd/vmd-1.7.1/ug/node56.html.

11. —. Tube. *VMD Visual Molecular Dynamics.* [Online] University of Illinois. [Cited: 10 August 2020.] https://www.ks.uiuc.edu/Research/vmd/vmd-1.7.1/ug/node61.html.

12. —. NewCartoon. *VMD Visual Molecular Dynamics.* [Online] University of Illinois. [Cited: 10 August 2020.] https://www.ks.uiuc.edu/Research/vmd/current/ug/node70.html.

13. Unity`. Game engines - how do they work? *Unity.* [Online] Unity. [Cited: 18 August 2020.] https://unity3d.com/what-is-a-game-engine.

14. Lilly, Paul. Doom to Dunia: A Visual History of 3D Game Engines. *maximumpc.* [Online] maximumpc, 7 June 2009. [Cited: 24 August 2020.] https://web.archive.org/web/20090724065520/http://www.maximumpc.com/article/features/3d_game_engines?page=0%2C3.

15. Epic Games. Unreal Engine. *Unreal Engine.* [Online] Epic Games. [Cited: 22 August 2020.] https://www.unrealengine.com/en-US/.

16. Axon, Samuel. Unity at 10: For better—or worse—game development has never been easier. *ars technica.* [Online] ars technica, 9 September 2016. [Cited: 24 August 2020.] https://arstechnica.com/gaming/2016/09/unity-at-10-for-better-or-worse-game-development-has-never-been-easier/.

17. Epic Games. Unreal Engine 4 Terminology. *Unreal Engine.* [Online] Epic Games. [Cited: 10 August 2020.] https://docs.unrealengine.com/en-US/GettingStarted/Terminology/index.html.

18. Oculus. Oculus Rift and RIft S Minimum Requirements and System Specifications. *Oculus.* [Online] Oculus. [Cited: 11 August 2020.] https://support.oculus.com/248749509016567/.

19. UCSF ChimeraX Home Page. *RBVI Resource for Biocomputing, Visualisation and Informatics.* [Online] RBVI. [Cited: 10 August 2020.] https://www.rbvi.ucsf.edu/chimerax/.

20. RBVI Resource for Biocomputing, Visualisation, and Informatics. ChimeraX Virtual Reality. *RBVI Resource for Biocomputing, Visualisation, and Informatics.* [Online] RBVI. [Cited: 10 August 2020.] https://www.cgl.ucsf.edu/chimerax/docs/user/vr.html.

21. Gu, Jenny and Bourne, Phillip E. The PDB format, mmCIF Formats, and other data formats. *Structural Bioinformatics, 2nd Edition.* s.l. : Wiley-Blackwell, 2009.

22. Scouloudi, H. *rcsb.org.* [Online] 22 March 1979. [Cited: 10 March 2020.] https://files.rcsb.org/view/1mbs.pdb.

23. Worldwide Protein Data Bank. Atomic Coordinate Entry Format Version 3.3. *wwpdb.org.* [Online] [Cited: 10 August 2020.] https://www.wwpdb.org/documentation/file-format-content/format33/v3.3.html.

24. Priimak, Dmitri. NAMD 32bit DCD File Format. *Github.* [Online] Github, 9 June 2017. [Cited: 10 August 2020.] https://github.com/priimak/scala-data/blob/master/NAMD-DCD-File-Format.md.

25. GROMACS. xtc file format. *gromacs.org.* [Online] GROMACS. [Cited: 10 August 2020.] http://manual.gromacs.org/archive/5.0.3/online/xtc.html.

26. Dmitrij, Frishman and Argos, Patrick. *STRIDE: Protein secondary structure assignment from atomic coordinates.* Heidelberg : European Molecular Biology Laboratory, 2002.

27. ...How do RasMol and Chime determine which atoms in a PDB file are covalently bonded?... *umass.edu.* [Online] umass.edu. [Cited: 10 August 2020.] https://www.umass.edu/microbio/rasmol/rasbonds.htm.

28. Epic Games. Introduction to Blueprints. *Unreal Engine.* [Online] Epic Games. [Cited: 10 August 2020.] https://docs.unrealengine.com/en-US/Engine/Blueprints/GettingStarted/index.html.

29. UE4 Blueprints From hell. *UE4 Blueprints From hell.* [Online] 15 January 2019. [Cited: 10 August 2020.] https://blueprintsfromhell.tumblr.com/image/182038604686.

30. Epic Games. Materials. *Unreal Engine.* [Online] [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/index.html.

31. —. Essential Material Concepts. *Unreal Engine.* [Online] Epic Games. [Cited: 18 August 2020.] https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/IntroductionToMaterials/index.html.

32. —. Material Inputs. *Unreal Engine.* [Online] [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/MaterialInputs/index.html.

33. —. Storing Custom Data in a Material Per Primitive. *Unreal Engine.* [Online] [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Engine/Rendering/Materials/CustomPrimitiveData/index.html.

34. —. Mesh Drawing Pipeline. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Programming/Rendering/MeshDrawingPipeline/index.html.

35. —. USimpleDynamicMeshComponent. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/API/Plugins/ModelingComponents/USimpleDynamicMeshComponent/index.html.

36. —. UInstancedStaticMeshComponent. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/API/Runtime/Engine/Components/UInstancedStaticMeshComponent/index.html.

37. —. Setting Up Automatic LOD Generation. *Unreal Engine.* [Online] Epic Games. [Cited: 22 August 2020.] https://docs.unrealengine.com/en-US/Engine/Content/Types/StaticMeshes/HowTo/AutomaticLODGeneration/index.html.

38. Oculus. General User Experience. *Oculus.* [Online] Oculus. [Cited: 11 August 2020.] https://developer.oculus.com/learn/bp-generalux/.

39. —. Vision. *Oculus.* [Online] Oculus. [Cited: 11 August 2020.] https://developer.oculus.com/learn/bp-vision/.

40. —. Locomotion. *Oculus.* [Online] Oculus. [Cited: 11 August 2020.] https://developer.oculus.com/learn/bp-locomotion/.

41. —. User Input. *Oculus.* [Online] Oculus. [Cited: 11 August 2020.] https://developer.oculus.com/learn/bp-locomotion/.

42. Epic Games. VR Performance Testing. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Platforms/VR/DevelopVR/Profiling/Overview/index.html.

43. —. Slate Architecture. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Programming/Slate/Architecture/index.html.

44. Zamborsky, Matus, Szabo, Tibor and Kozlikova, Barbora. *Dynamic Visualisation of protein secondary structures.* Wien : Central European Seminar on Computer Graphics, 2009.

45. Parent, Rick. Appendix B.5.5. *Computer Animation.* Waltham : Elsevier, 2012.

46. Neon Helium Productions. ArcBall Rotation. *Neon Helium Productions.* [Online] Neon Helium Productions. [Cited: 20 August 2020.] http://nehe.gamedev.net/tutorial/arcball_rotation/19003/.

47. Epic Games. Gauntlet Automation Framework Overview. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Programming/Automation/Gauntlet/Overview/index.html.

48. —. CSV Profiler. *Unreal Engine.* [Online] Epic Games. [Cited: 11 August 2020.] https://docs.unrealengine.com/en-US/Engine/Performance/CSVProfiler/index.html.

49. Nadaski, David . Gauntlet Automated Testing and Performance Metrics in UE4. *HŌRU.* [Online] HŌRU, 16 May 2020. [Cited: 11 August 2020.] https://horugame.com/gauntlet-automated-testing-and-performance-metrics-in-ue4/.

50. Webber, Joseph R. *ProteinShader: illistrative rendering of macromolocules.* s.l. : BMC Structural Biology, 2009.

51. Epic Games. Threaded Rendering. *Unreal Engine.* [Online] Epic Games. [Cited: 18 August 2020.] https://docs.unrealengine.com/en-US/Programming/Rendering/ThreadedRendering/index.html.

52. nvidia. Hardware Occlusion Queries Made Useful. [book auth.] Matt Pharr. *GPU Gems 2.* s.l. : Pearson, 2005.

53. Van Der Waals Radius of the elements. *periodictable.* [Online] periodictable.com. [Cited: 10 August 2020.] https://periodictable.com/Properties/A/VanDerWaalsRadius.an.html.

Appendix A
External Materials

The following section contains links to source code and additional material.

A.1 Deliverables

The following link is to a GitHub repository containing all source code, executables, and performance reports. A ReadMe is included with instructions on how to set up the project.

https://github.com/bonorumetmalorum/MolViz/releases/tag/v1.0.0

A.1.1 Unreal Engine 4

To download the Epic Installer use the following link:

https://www.unrealengine.com/en-US/auth?state=https://www.unrealengine.com/en-US/download/ue_non_games

Install UE4 version 4.25, clone the repository above and open it using UE4.25.

A.1.2 Automation Project

To use the automation project with the Unreal Automation Tool (UAT), clone the following repo:

https://github.com/bonorumetmalorum/MolVizBenchmark.Automation

A.1.3 Audio Transcripts for Usability Sessions

The following link contains the audio transcripts for the usability sessions:

https://drive.google.com/file/d/1t1SvSsS1xIOKtCkQH5WYBwq8r0YKhuiZ/view?usp=sharing

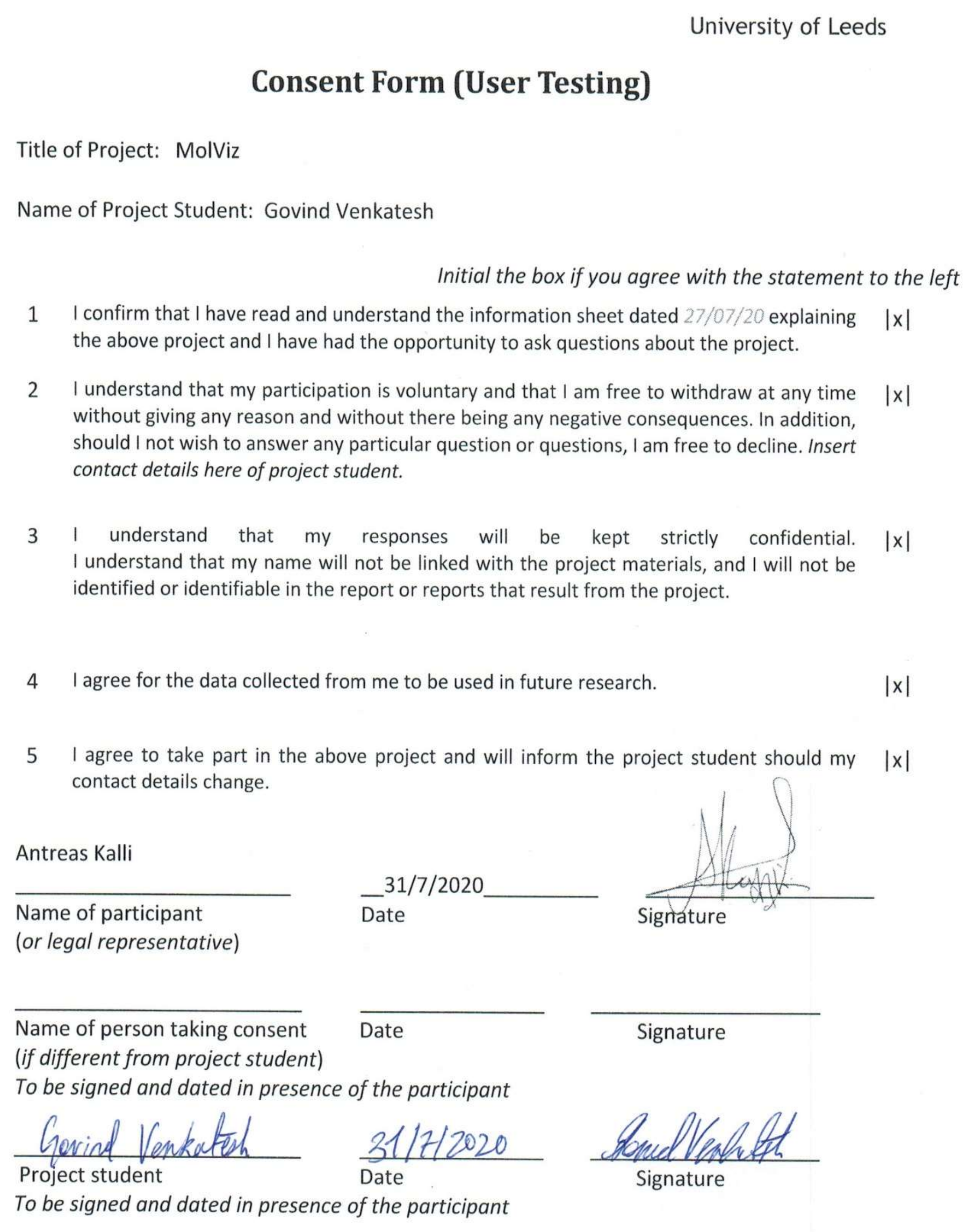
A.1.4 RenderDoc

The following link is to the official RenderDoc website.

https://renderdoc.org/

Appendix B
Ethical Issues Addressed

B.1 Consent Forms



University of Leeds

Consent Form (User Testing)

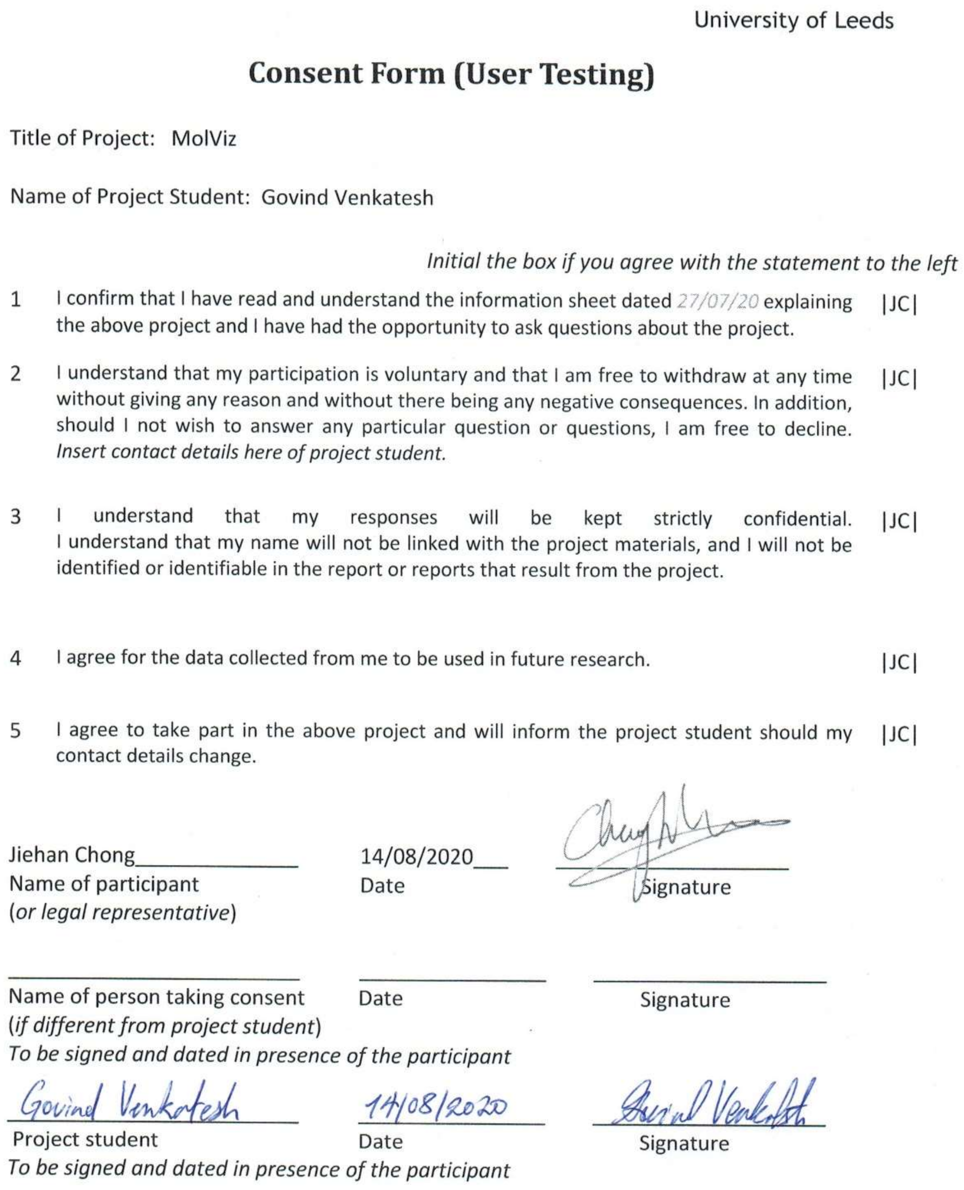
Title of Project: MolViz

Name of Project Student: Govind Venkatesh

Initial the box if you agree with the statement to the left

1	I confirm that I have read and understand the information sheet dated 27/07/20 explaining the above project and I have had the opportunity to ask questions about the project.	|x|
2	I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason and without there being any negative consequences. In addition, should I not wish to answer any particular question or questions, I am free to decline. *Insert contact details here of project student.*	|x|
3	I understand that my responses will be kept strictly confidential. I understand that my name will not be linked with the project materials, and I will not be identified or identifiable in the report or reports that result from the project.	|x|
4	I agree for the data collected from me to be used in future research.	|x|
5	I agree to take part in the above project and will inform the project student should my contact details change.	|x|

Antreas Kalli	31/7/2020	[illegible]
Name of participant (*or legal representative*)	Date	Signature
Name of person taking consent (*if different from project student*) *To be signed and dated in presence of the participant*	Date	Signature
Govind Venkatesh	31/7/2020	[illegible]
Project student *To be signed and dated in presence of the participant*	Date	Signature



University of Leeds

Consent Form (User Testing)

Title of Project: MolViz

Name of Project Student: Govind Venkatesh

Initial the box if you agree with the statement to the left

1	I confirm that I have read and understand the information sheet dated 27/07/20 explaining the above project and I have had the opportunity to ask questions about the project.	|JC|
2	I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason and without there being any negative consequences. In addition, should I not wish to answer any particular question or questions, I am free to decline. *Insert contact details here of project student.*	|JC|
3	I understand that my responses will be kept strictly confidential. I understand that my name will not be linked with the project materials, and I will not be identified or identifiable in the report or reports that result from the project.	|JC|
4	I agree for the data collected from me to be used in future research.	|JC|
5	I agree to take part in the above project and will inform the project student should my contact details change.	|JC|

Jiehan Chong	14/08/2020	[illegible]
Name of participant (*or legal representative*)	Date	Signature
Name of person taking consent (*if different from project student*) *To be signed and dated in presence of the participant*	Date	Signature
Govind Venkatesh	14/08/2020	[illegible]
Project student *To be signed and dated in presence of the participant*	Date	Signature

University of Leeds

Consent Form (User Testing)

Title of Project: MolViz

Name of Project Student: Govind Venkatesh

Initial the box if you agree with the statement to the left

1	I confirm that I have read and understand the information sheet dated 27/07/20 explaining the above project and I have had the opportunity to ask questions about the project.	|KLH|
2	I understand that my participation is voluntary and that I am free to withdraw at any time without giving any reason and without there being any negative consequences. In addition, should I not wish to answer any particular question or questions, I am free to decline. *Insert contact details here of project student.*	|KLH|
3	I understand that my responses will be kept strictly confidential. I understand that my name will not be linked with the project materials, and I will not be identified or identifiable in the report or reports that result from the project.	|KLH|
4	I agree for the data collected from me to be used in future research.	|KLH|
5	I agree to take part in the above project and will inform the project student should my contact details change.	|KLH|

Kyle Le Huray 05/08/20 *klehuray.*

Kyle Le Huray 05/08/20		*klehuray.*
Name of participant (*or legal representative*)	Date	Signature
Name of person taking consent (*if different from project student*) *To be signed and dated in presence of the participant*	Date	Signature
Govind Venkatesh	05/08/20	[illegible]
Project student *To be signed and dated in presence of the participant*	Date	Signature

Appendix C
Additional Materials

C.1Van Der Waals radii

Table 4 Van Der Waals radii for all elements **(53)**

Element	Radius (picometer)	Element	Radius (picometer)
Hydrogen	120	Helium	140
Lithium	182	Carbon	170
Nitrogen	155	Oxygen	152
Fluorine	147	Neon	154
Sodium	227	Magnesium	173
Silicon	210	Phosphorus	180
Sulfur	180	Chlorine	175
Chlorine	175	Argon	188
Potassium	275	Nickel	163
Copper	140	Zinc	139
Gallium	187	Arsenic	185
Selenium	190	Bromine	185
Krypton	202	Palladium	163
Silver	172	Cadmium	158
Indium	193	Tin	217
Tellurium	206	Iodine	198
Xenon	216	Platinum	175
Gold	166	Mercury	155
Thallium	196	Lead	202
Uranium	186	-	-

C.2 User Testing interview script

Table 5 User Testing interview script

Step No.	Step description
1	LOAD MOLECULE 1
2	DEMO ROTATION
3	DEMO SCALING
4	DEMO TRANSLATION
5	REMOVE VDW
6	DEMO CPK
7	DEMO TUBE
8	HIDE CPK
9	HIDE TUBE
10	SHOW NCARTOON
11	SHOW CPK
12	LOAD MOLECULE 2

C.3 Performance report break down

The performance report is created as an HTML page with interactive SVG graphs. It can be found in the GitHub repository mentioned in Appendix A.1.

The graphs available for viewing are as follows:

- Stat Unit
    - Shows a smoothed graph by computing the running average
        - Frame Time
            - The total amount of time taken in milliseconds to complete a frame's worth of operations
        - GPU Time
            - The total amount of time the GPU was used this frame
        - Render Thread Time
            - The total amount of time the render thread took to finish executing in milliseconds
        - Game Thread Time
            - The total amount of time the game thread took to finish executing in milliseconds
- Stat Unit Raw
    - This is the unsmoothed version of the Stat Unit graph
- Gamethread Breakdown
    - Shows a smoothed graph by computing the running average
    - Breaks down all the various routines that are executed in the game thread in milliseconds
- Gamethread Breakdown Raw
    - This is the unsmoothed version of the Gamethread Breakdown
- Gamethread Waits
    - Shows in milliseconds the Frame Time and the Gamethread Time
    - This graph is used to detect issues with the Gamethread blocking, either due to long synchronous operations in the Gamethread or due to slow performance on the Renderthread
        - The game thread must be 1 or 2 frames ahead of the Renderthread, if it falls behind too far the Gamethread must wait for it to catch up.
- Gamethread Physics
    - Shows the time taken to process physics and the EndPhysics tick group to finish in milliseconds.

- Ticks Breakdown
    - Shows the number of ticks queued, total number of actors that are ticking in a frame
    - The number of particle system manager ticks
    - Line batch component, which seems to be inaccurate and should be ignored
    - ProteinRepresentation ticks queued, which is the actor class used to render a protein.
- Actor Counts
    - Measure the total number of actors loaded in the level for each frame
- Physical Memory
    - Measure the amount of free memory in MegaBytes (MB) and the amount of used memory in MB each frame
- Niagara Breakdown
    - Measure the amount of time used by the Niagara visual effects system each frame
- RHI Drawcalls
    - Measure the number of draw calls created each frame
- Renderthread Breakdown
    - Shows the detailed view of all render thread processes and time taken by each in milliseconds
    - RenderOther encompasses every unlisted post processing step, and optimization algorithms such as Occlusion Culling.
- Renderthread Breakdown Raw
    - The unsmoothed version of the Renderthread breakdown
- Log Counts
    - Shows the number of calls the logging functions each frame
- CSV Profiler
    - Shows statistics related to the performance of the CSV profiler tool
    - Measure the number of time stamps and custom statistics processed each frame as well as the total execution time each frame in milliseconds.

C.4 RenderDoc BasePass draw calls snippet

Find below a snippet of the draw calls in the BasePass where the molecule geometry is rendered (see section 5.2.2.2 for details). Highlighted in yellow, are the instanced draw calls.

```
     |     - BasePass                                        | 188
     |      \- defaultmaterial Cylinder                        | 188
992  |         \- DrawIndexed(666)                             | 188
     |       - defaultmaterial Cylinder                        | 189
998  |         \- DrawIndexed(666)                             | 189
     |       - defaultmaterial Cylinder                        | 190
1004 |         \- DrawIndexed(666)                             | 190
     |       - defaultmaterial Cylinder                        | 191
1010 |         \- DrawIndexed(666)                             | 191
     |       - defaultmaterial Cylinder                        | 192
1016 |         \- DrawIndexed(666)                             | 192
     |       - defaultmaterial Cylinder                        | 193
1022 |         \- DrawIndexed(666)                             | 193
     |       - defaultmaterial Cylinder                        | 194
1028 |         \- DrawIndexed(666)                             | 194
     |       - defaultmaterial Cylinder                        | 195
1034 |         \- DrawIndexed(666)                             | 195
     |       - defaultmaterial Cylinder                        | 196
1040 |         \- DrawIndexed(666)                             | 196
     |       - defaultmaterial Cylinder                        | 197
1046 |         \- DrawIndexed(666)                             | 197
     |       - defaultmaterial Cylinder                        | 198
1052 |         \- DrawIndexed(666)                             | 198
     |       - defaultmaterial Cylinder                        | 199
1058 |         \- DrawIndexed(666)                             | 199
```

```
     |      - defaultmaterial Cylinder                    | 200
1064 |        \- DrawIndexed(666)                          | 200
     |      - defaultmaterial Cylinder                    | 201
1070 |        \- DrawIndexed(666)                          | 201
     |      - defaultmaterial Cylinder                    | 202
1076 |        \- DrawIndexed(666)                          | 202
     |      - defaultmaterial Cylinder                    | 203
1082 |        \- DrawIndexed(666)                          | 203
     |      - defaultmaterial Cylinder                    | 204
1088 |        \- DrawIndexed(666)                          | 204
     |      - defaultmaterial Cylinder                    | 205
1094 |        \- DrawIndexed(666)                          | 205
     |      - defaultmaterial Cylinder                    | 206
1100 |        \- DrawIndexed(666)                          | 206
     |      - defaultmaterial Cylinder                    | 207
1106 |        \- DrawIndexed(666)                          | 207
     |      - defaultmaterial Cylinder                    | 208
1112 |        \- DrawIndexed(666)                          | 208
     |      - defaultmaterial Cylinder                    | 209
1118 |        \- DrawIndexed(666)                          | 209
     |      - defaultmaterial Cylinder                    | 210
1124 |        \- DrawIndexed(666)                          | 210
     |      - defaultmaterial Cylinder                    | 211
1130 |        \- DrawIndexed(666)                          | 211
     |      - defaultmaterial Cylinder                    | 212
1136 |        \- DrawIndexed(666)                          | 212
     |      - defaultmaterial Cylinder                    | 213
1142 |        \- DrawIndexed(666)                          | 213
```

```
     |        - defaultmaterial Cylinder                          | 214
1148 |          \- DrawIndexed(666)                                | 214
     |        - defaultmaterial Cylinder                          | 215
1154 |          \- DrawIndexed(666)                                | 215
     |        - defaultmaterial Cylinder                          | 216
1160 |          \- DrawIndexed(666)                                | 216
     |        - defaultmaterial Cylinder                          | 217
1166 |          \- DrawIndexed(666)                                | 217
     |        - defaultmaterial Cylinder                          | 218
1172 |          \- DrawIndexed(666)                                | 218
     |        - defaultmaterial Cylinder                          | 219
1178 |          \- DrawIndexed(666)                                | 219
     |        - defaultmaterial Cylinder                          | 220
1184 |          \- DrawIndexed(666)                                | 220
     |        - defaultmaterial Cylinder                          | 221
1190 |          \- DrawIndexed(666)                                | 221
     |        - defaultmaterial Cylinder                          | 222
1196 |          \- DrawIndexed(666)                                | 222
     |        - defaultmaterial Cylinder                          | 223
1202 |          \- DrawIndexed(666)                                | 223
     |        - defaultmaterial Cylinder                          | 224
1208 |          \- DrawIndexed(666)                                | 224
     |        - defaultmaterial Cylinder                          | 225
1214 |          \- DrawIndexed(666)                                | 225
     |        - defaultmaterial Cylinder                          | 226
1220 |          \- DrawIndexed(666)                                | 226
     |        - defaultmaterial Cylinder                          | 227
1226 |          \- DrawIndexed(666)                                | 227
```

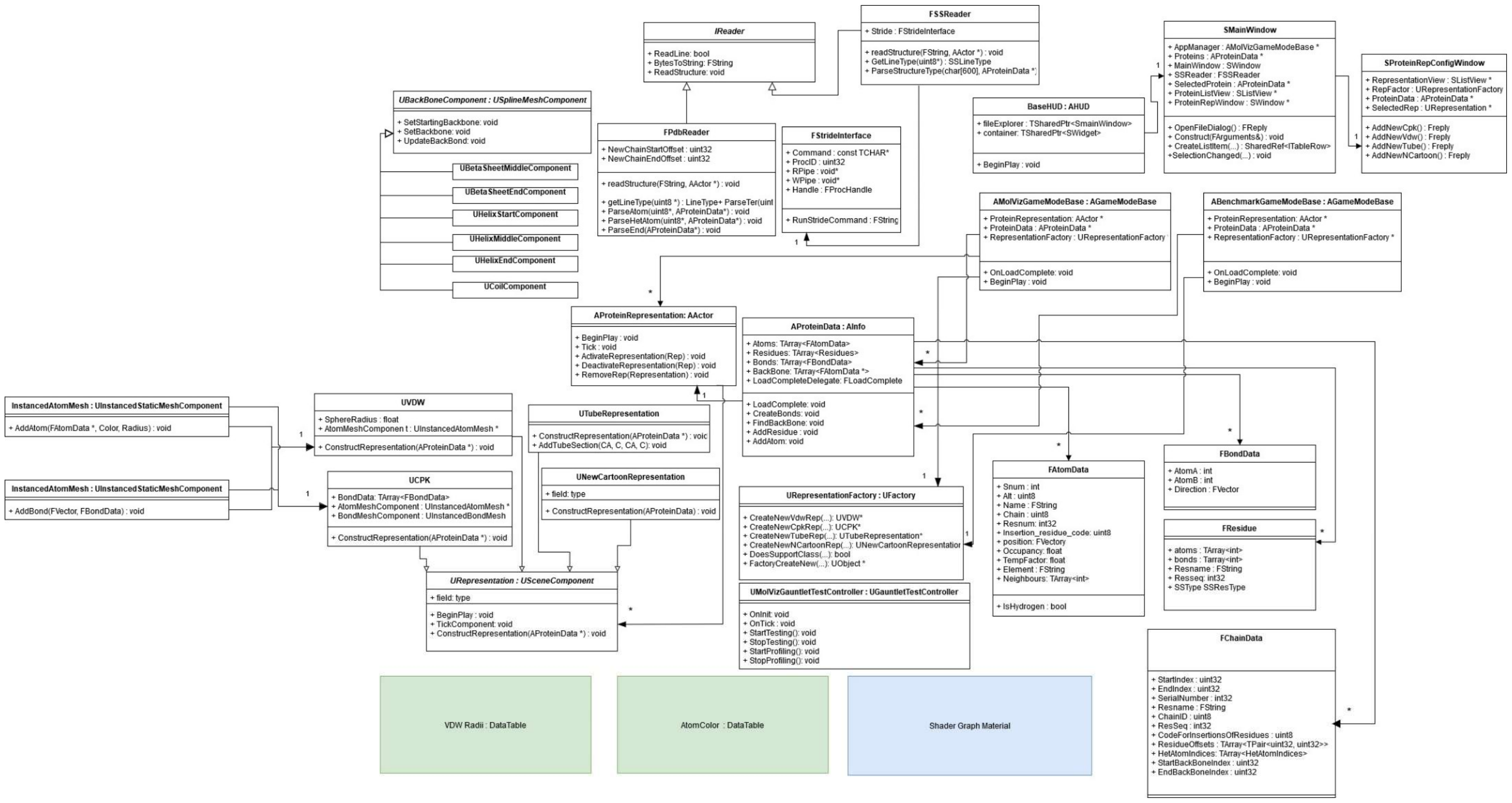
```
     |     - defaultmaterial Cylinder              | 228
1232 |        \- DrawIndexed(666)                   | 228
     |     - defaultmaterial Cylinder              | 229
1238 |        \- DrawIndexed(666)                   | 229
     |     - defaultmaterial Cylinder              | 230
1244 |        \- DrawIndexed(666)                   | 230
     |     - defaultmaterial Cylinder              | 231
1250 |        \- DrawIndexed(666)                   | 231
     |     - defaultmaterial Cylinder              | 232
1256 |        \- DrawIndexed(666)                   | 232
     |     - defaultmaterial Cylinder              | 233
1262 |        \- DrawIndexed(666)                   | 233
     |     - defaultmaterial Cylinder              | 234
1268 |        \- DrawIndexed(666)                   | 234
     |     - defaultmaterial Cylinder              | 235
1274 |        \- DrawIndexed(666)                   | 235
     |     - defaultmaterial Cylinder              | 236
1280 |        \- DrawIndexed(666)                   | 236
     |     - defaultmaterial Cylinder              | 237
1286 |        \- DrawIndexed(666)                   | 237
     |     - defaultmaterial Cylinder              | 238
1292 |        \- DrawIndexed(666)                   | 238
     |     - defaultmaterial Cylinder              | 239
1298 |        \- DrawIndexed(666)                   | 239
     |     - defaultmaterial Cylinder              | 240
1304 |        \- DrawIndexed(666)                   | 240
     |     - defaultmaterial Cylinder              | 241
1310 |        \- DrawIndexed(666)                   | 241
```

```
     |      - defaultmaterial Cylinder                    | 242
1316 |        \- DrawIndexed(666)                           | 242
     |      - defaultmaterial Cylinder                    | 243
1322 |        \- DrawIndexed(666)                           | 243
     |      - defaultmaterial Cylinder                    | 244
1328 |        \- DrawIndexed(666)                           | 244
     |      - defaultmaterial Cylinder                    | 245
1334 |        \- DrawIndexed(666)                           | 245
     |      - defaultmaterial Cylinder                    | 246
1340 |        \- DrawIndexed(666)                           | 246
     |      - defaultmaterial Cylinder                    | 247
1346 |        \- DrawIndexed(666)                           | 247
     |      - defaultmaterial Cylinder                    | 248
1352 |        \- DrawIndexed(666)                           | 248
     |      - defaultmaterial Cylinder                    | 249
1358 |        \- DrawIndexed(666)                           | 249
     |      - defaultmaterial Cylinder                    | 250
1364 |        \- DrawIndexed(666)                           | 250
     |      - defaultmaterial Cylinder                    | 251
1370 |        \- DrawIndexed(666)                           | 251
     |      - defaultmaterial Cylinder                    | 252
1376 |        \- DrawIndexed(666)                           | 252
     |      - defaultmaterial Cylinder                    | 253
1382 |        \- DrawIndexed(666)                           | 253
     |      - defaultmaterial Cylinder                    | 254
1388 |        \- DrawIndexed(666)                           | 254
     |      - defaultmaterial Cylinder                    | 255
1394 |        \- DrawIndexed(666)                           | 255
```

```
     |      - defaultmaterial Cylinder                 | 256
1400 |        \- DrawIndexed(666)                        | 256
     |      - defaultmaterial Cylinder                 | 257
1406 |        \- DrawIndexed(666)                        | 257
     |      - defaultmaterial Cylinder                 | 258
1412 |        \- DrawIndexed(666)                        | 258
     |      - defaultmaterial Cylinder                 | 259
1418 |        \- DrawIndexed(666)                        | 259
     |      - defaultmaterial Cylinder                 | 260
1424 |        \- DrawIndexed(666)                        | 260
     |      - defaultmaterial Cylinder                 | 261
1430 |        \- DrawIndexed(666)                        | 261
     |      - defaultmaterial Cylinder                 | 262
1436 |        \- DrawIndexed(666)                        | 262
     |      - defaultmaterial Sphere 660 instances            | 263
1455 |        \- DrawIndexedInstanced(2880, 660)                 | 263
     |      - defaultmaterial Cylinder 1216 instances         | 264
1469 |        \- DrawIndexedInstanced(1332, 1216)                | 264
1475 |       - API Calls                              | 265
```

C.5 System Architecture Diagram

This is a reproduction of Figure 16 in section 3.7, provided here for clarity.


IReader
+ ReadLine: bool
+ BytesToString: FString
+ ReadStructure: void
FSSReader
+ Stride : FStrideInterface
+ readStructure(FString, AActor *) : void
+ GetLineType(uint8*) : SSLineType
+ ParseStructureType(char[600], AProteinData *)
SMainWindow
+ AppManager : AMolVizGameModeBase *
+ Proteins : AProteinData *
+ MainWindow : SWindow
+ SSReader : FSSReader
+ SelectedProtein : AProteinData *
+ ProteinListView : SListView *
+ ProteinRepWindow : SWindow *
+ OpenFileDialog() : FReply
+ Construct(FArguments&) : void
+ CreateListItem(...) : SharedRef<ITableRow>
+SelectionChanged(...) : void
SProteinRepConfigWindow
+ RepresentationView : SListView *
+ RepFactor : URepresentationFactory
+ ProteinData : AProteinData *
+ SelectedRep : URepresentation *
+ AddNewCpk() : Freply
+ AddNewVdw() : Freply
+ AddNewTube() : Freply
+ AddNewNCartoon() : Freply
UBackBoneComponent : USplineMeshComponent
+ SetStartingBackbone: void
+ SetBackbone: void
+ UpdateBackBond: void
UBetaSheetMiddleComponent
UBetaSheetEndComponent
UHelixStartComponent
UHelixMiddleComponent
UHelixEndComponent
UCoilComponent
FPdbReader
+ NewChainStartOffset : uint32
+ NewChainEndOffset : uint32
+ readStructure(FString, AActor *) : void
+ getLineType(uint8 *) : LineType+ ParseTer(uint
+ ParseAtom(uint8*, AProteinData*) : void
+ ParseHetAtom(uint8*, AProteinData*) : void
+ ParseEnd(AProteinData*) : void
FStrideInterface
+ Command : const TCHAR*
+ ProcID : uint32
+ RPipe : void*
+ WPipe : void*
+ Handle : FProcHandle
+ RunStrideCommand : FString
BaseHUD : AHUD
+ fileExplorer : TSharedPtr<SmainWindow>
+ container: TSharedPtr<SWidget>
+ BeginPlay : void
AMolVizGameModeBase : AGameModeBase
+ ProteinRepresentation: AActor *
+ ProteinData : AProteinData *
+ RepresentationFactory : URepresentationFactory *
+ OnLoadComplete: void
+ BeginPlay : void
ABenchmarkGameModeBase : AGameModeBase
+ ProteinRepresentation: AActor *
+ ProteinData : AProteinData *
+ RepresentationFactory : URepresentationFactory *
+ OnLoadComplete: void
+ BeginPlay : void
AProteinRepresentation: AActor
+ BeginPlay : void
+ Tick : void
+ ActivateRepresentation(Rep) : void
+ DeactivateRepresentation(Rep) : void
+ RemoveRep(Representation) : void
AProteinData : AInfo
+ Atoms: TArray<FAtomData>
+ Residues: TArray<Residues>
+ Bonds: TArray<FBondData>
+ BackBone: TArray<FAtomData *>
+ LoadCompleteDelegate: FLoadComplete
+ LoadComplete: void
+ CreateBonds: void
+ FindBackBone: void
+ AddResidue : void
+ AddAtom: void
InstancedAtomMesh : UInstancedStaticMeshComponent
+ AddAtom(FAtomData *, Color, Radius) : void
InstancedAtomMesh : UInstancedStaticMeshComponent
+ AddBond(FVector, FBondData) : void
UVDW
+ SphereRadius : float
+ AtomMeshComponen t : UInstancedAtomMesh *
+ ConstructRepresentation(AProteinData *) : void
UCPK
+ BondData: TArray<FBondData>
+ AtomMeshComponent : UInstancedAtomMesh *
+ BondMeshComponent : UInstancedBondMesh
+ ConstructRepresentation(AProteinData *) : void
UTubeRepresentation
+ ConstructRepresentation(AProteinData *) : void
+ AddTubeSection(CA, C, CA, C): void
UNewCartoonRepresentation
+ field: type
+ ConstructRepresentation(AProteinData) : void
URepresentation : USceneComponent
+ field: type
+ BeginPlay : void
+ TickComponent: void
+ ConstructRepresentation(AProteinData *) : void
URepresentationFactory : UFactory
+ CreateNewVdwRep(...): UVDW*
+ CreateNewCpkRep(...): UCPK*
+ CreateNewTubeRep(...): UTubeRepresentation*
+ CreateNewNCartoonRep(...): UNewCartoonRepresentatior
+ DoesSupportClass(...): bool
+ FactoryCreateNew(...): UObject *
UMolVizGauntletTestController : UGauntletTestController
+ OnInit: void
+ OnTick : void
+ StartTesting(): void
+ StopTesting(): void
+ StartProfiling(): void
+ StopProfiling(): void
FAtomData
+ Snum : int
+ Alt : uint8
+ Name : FString
+ Chain : uint8
+ Resnum: int32
+ Insertion_residue_code: uint8
+ position: FVectory
+ Occupancy: float
+ TempFactor: float
+ Element : FString
+ Neighbours: TArray<int>
+ IsHydrogen : bool
FBondData
+ AtomA : int
+ AtomB : int
+ Direction : FVector
FResidue
+ atoms : TArray<int>
+ bonds : Tarray<int>
+ Resname : FString
+ Resseq: int32
+ SSType SSResType
FChainData
+ StartIndex : uint32
+ EndIndex : uint32
+ SerialNumber : int32
+ Resname : FString
+ ChainID : uint8
+ ResSeq : int32
+ CodeForInsertionsOfResidues : uint8
+ ResidueOffsets : TArray<TPair<uint32, uint32>>
+ HetAtomIndices: TArray<HetAtomIndices>
+ StartBackBoneIndex : uint32
+ EndBackBoneIndex : uint32
VDW Radii : DataTable
AtomColor : DataTable
Shader Graph Material
1
*