Interacting with molecular models in virtual reality

Oriol Giralt Garcia

17/4/2019

Universitat Politècnica de Catalunya

Index

1. Context and scope of the project

1.1 Prelude and problem formulation

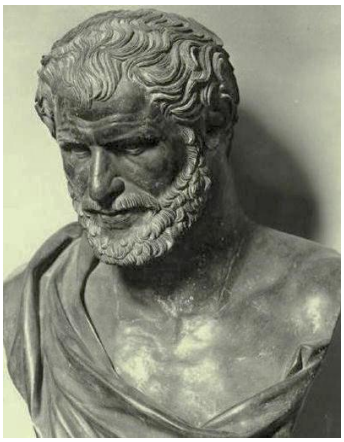


Figure 1 Democritus was an Ancient Greek pre-Socratic philosopher primarily remembered today for his formulation of an atomic theory of the universe.

Since immemorial times that humans have asked the question of what we are made of. The term atom comes from the Greek "atomon" which is the union of two terms; <<α>> (*a*) that means without and <<τομον>> (*tomon*) which denotes division, meaning than something is indivisible, that cannot be divided.[1] The concept of atom as a basic and indivisible block that composes all the matter of the universe was postulated in ancient Greece in the 5th century BC. As well as the modern concept of molecules. Later, in the early nineteenth, the concept gained a broad scientific acceptance when a big number of discoveries in the chemistry field demonstrated that the surrounding materials really behaves as if they were composed of atoms.

Thanks to these discoveries and the great technological advances that we have experienced in the recent decades, the molecular dynamics field was born. Molecular dynamics are methods of computing simulations of the physical movements of atoms and molecules, and the interactions between them. These simulations are used, among other things, in chemical physics, materials science, and the modelling of biomolecules.

One of the main problems that the molecular dynamics present is that the computation of dynamic molecular simulations can be a complex and time-consuming process. The results generated by molecular dynamics techniques are usually composed of thousands of elementary steps, which can reach the number of gigabytes of data from our memory

for each simulation and, despite of the techniques of optimization of these methods, analysing the results can take several days.[2]

Providing new tools and techniques to visualize and interact with simulations provided by molecular dynamics are key to understanding a bit more the atomic and molecular world. The main objective of this project is to build a molecular visualizer in virtual reality where the user can interact with the molecule and visualize it in the most common methods of molecule representation. Another aim of this project is to implement algorithms and optimizations than will enhance the performance of its rendering, allowing to visualize in a clear and stable way each part of the molecule without compromising the system. We will also build a parser than the visualizer will use to read any protein dat49se file format (.pdb) and will parse it to C# classes. The parser will read all the information about the molecule structures as well as the atomical information and will calculate its bonds. Furthermore, we will implement an attractive, consistent and clear user interface controlled by the virtual controllers in which the user will interact with the molecule in an attractive way. The virtual reality controllers will be working fluently and no molecule part will be unable to select. The user will be able to walk through the molecule and observe all the atomically parts and bonds of the molecule in a complete peripherical view. The designed user interface will be user-friendly and easy to use, facilitating the user's experience. There will not be any bug in the visualizer that could damage the user experience.

1.2. Introduction to the biomolecular world

During this chapter we will make a brief introduction to the biomolecular field in order to acquire some basic knowledges so the reader can understand and follow the methodology that will be taken on term during this project.

Firstly, we will define what a molecule is. A molecule is an electrically neutral group of two or more atoms held together by chemical bonds[3]. One of the most common

molecules are the proteins. A protein is a large macromolecule made by the union of amino acids through a peptide bond[4]. Those amino acids than make up the molecule are organic compounds that can be composed by carbon, oxygen, nitrogen, hydrogen and, in some cases, sulphur.[6]

We can classify the atoms of each amino acid into subgroups depending of its structure. Those groups are the group of the *amines*, the group of *carboxyl* and the group of *sidechains*. The sidechain is the only group that varies between amino acids and determines its several properties.[5] Analogously, we can classify the amino acids according to the type of interaction. The main groups are the *hydrophobic*, *hydrophilic*, charged amino acids and the amino acids without sidechain. The hydrophobic amino acids have carbon-rich side chains, which don't interact well with water. The hydrophilic or polar amino acids interact well with water. Charged amino acids interact with oppositely charged amino acids or other molecules.[6]

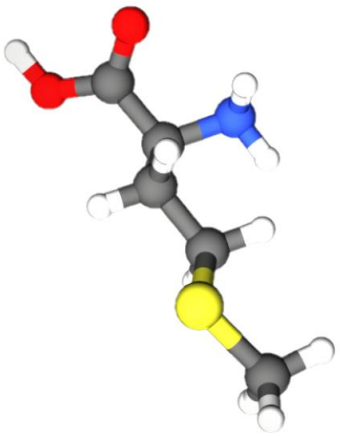


Figure 2 Graphical representation of an amino acid. The bound between the oxygen atoms (red) with the hydrogen atoms (white) represents the amino group while the bound between the nitrogen (blue) and the two hydrogens represents the carboxyl group. Both are connec-ted with the sidechain y a carbon atom (grey).

The major parts of the proteins can fold into unique structures allowing the free rotation of its atoms. The way a protein its fold on his natural state is known as native state. Although many proteins can fold thanks to the chemical properties of its amino acids, there are many than require help of the *chaperone proteins* in order to acquire his native states. The biochemists, usually refers to four different aspects of the main structure of a protein:[7]

· **Primary structure:** The primary structure of a protein is the linear sequence of amino acids as encoded by DNA. The amino acids are joined by peptide bonds, which link an amino group and a carboxyl group and a water molecule is released each time a bond is formed. The linked series of carbon, nitrogen, and oxygen atoms make up the *protein backbone*.

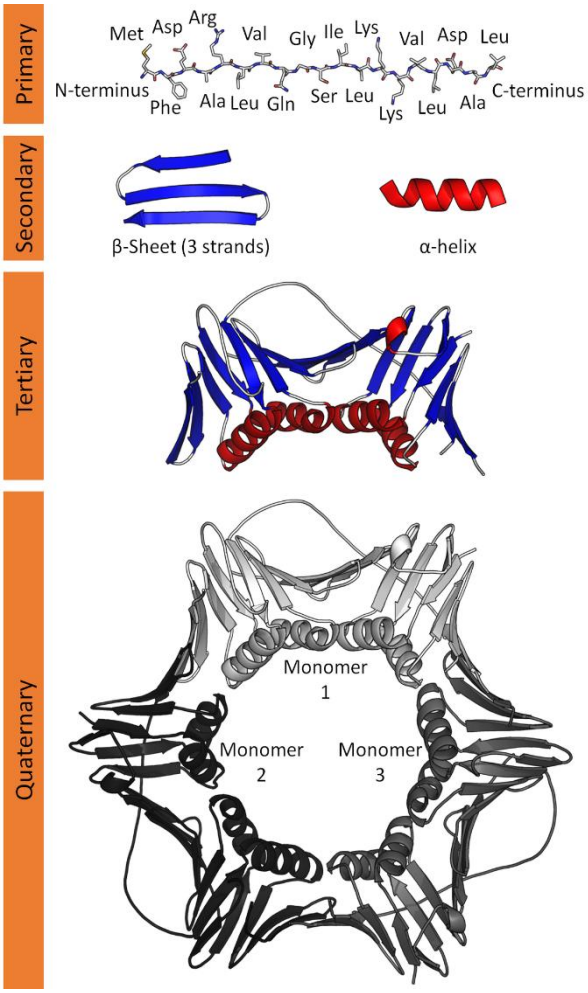


Figure 3 Diagram of the four levels of the protein structure.

· **Secondary structure:** The protein chains often fold into two types of secondary structures: *alpha-helices*, or *beta-strands*. An alpha-helix is a right-handed coil stabilized by hydrogen links between the amine and carboxyl groups of nearby amino acids. Those hydrogen links are called *hydrogen bonds*. Beta-strands are formed when hydrogen bonds stabilize two or more adjacent strands.

· **Tertiary structure:** The tertiary structure of a protein is the three-dimensional shape of the protein chain. This shape is determined by the characteristics of the amino acids making up the chain. Charged amino acids allow proteins to interact with molecules that have complementary charges. The functions of many proteins rely on their three-dimensional shapes. For example, haemoglobin forms a pocket to hold *heme*, a small molecule with an iron atom in the center that binds oxygen.

· **Quaternary structure:** Two or more polypeptide chains can come together to form one functional molecule with several subunits. For example, the four subunits of haemoglobin cooperate so that the complex can pick up more oxygen in the lungs and release it in the body.

1.3. Rendering techniques

In this chapter we will discuss and review the most notorious work made on the molecule's representation. Furthermore, we will explain the use of that models and will indicate its major shortcomings.

1.3.1. Space-filling

The Space-filling, also known as a calotte model, is one of the most common methods used to visualize molecules. This model represents each atom of the molecule with a sphere. The sphere size will be determined by the element's van der W35s radius of the atom, which radius represents the distance of closest approach for another atom[8]. Space-filling models can be also referred to as CPK models.

The good point of the Space-filling models is that they are useful for visualizing the relative dimensions of the molecule, and the effective shape the molecule might present. On the other hand, as we said before the radius of the sphere will be defined by a half of the distance between the closest approaches of the two non-bonded atoms of a given element. This means than the chemical bonds between the atoms are going to be masked and will make difficult to see the structure of the molecule. This is the reason why such models will have a better performance if they can be used dynamically [9].

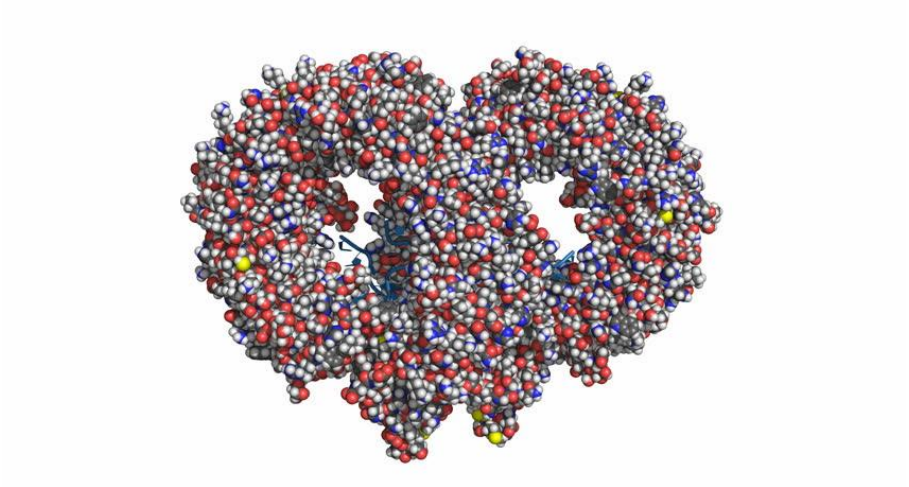


Figure 4 Space filling model with conventional atom color-coding of a Toll-like Receptor (TLR4).

1.3.2. Ball & Sticks

Despite the Ball & Sticks models presents similarities to the Space-filling model, the B&S model uses a smaller sphere radius, being able to represent with a set of cylinders, the bounds of the atoms. Thus, the B&S model will also have had to render the cylinders in order the represent that bounds. The ideal representation of the B&S model has to have the same angles between the rods as the angles between the bonds, and the distances between the centers of the spheres should be proportional to the distances between the corresponding atomic nuclei[10]. The B&S model also follows the CPK coloring. In order to provide a clearer view of the atoms and bonds throughout the model, the radius of the spheres has to be smaller than the rod lengths.

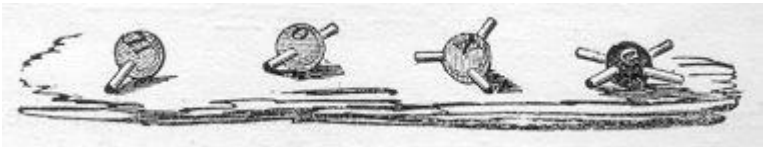


Figure 5. Hofmann's set of models. we distinguish the chlorine atom as univalent, the atom of oxygen as bivalent, that of nitrogen as trivalent, and lastly the carbon atom as quadrivalent.[24]

1.3.3. Ribbons

The Ribbons diagram model is a method than provides a higher level of abstraction of the underlying structure of biomolecules by rendering their backbone chain using a set of planes and tubes. Ribbon diagrams are generated by interpolating a smooth curve through the polypeptide backbone. Alpha-helices are shown as coiled ribbons or thick tubes, whereas betta-strands as arrows lines or thin tubes for non-repetitive coils or loops. The direction of the polypeptide chain is shown locally by the arrows.

This representation method is one of the favorite methods of the biologist and other scientist and researchers because it presents a simple, but complete and powerful view of the basics of a molecular structure meaning we can see in an easy way the overall structure of the molecule appreciating if it's twisted, folded or unfolded.[11]

1.4. Scope and methodology

The scope of this project will consist of creating competitive molecular visualizer ready to use by the researchers for de study of molecules. The project will be completed by June, 2018. The modules of the visualizer will include a protein dat49se reader than will read any .pdb file and represent it graphically, a complete selector mode in which the user will be able to group the atoms of the molecule by different aspect such as the residues, the element or the chain group, and several ways to interact with the surrounding molecule.

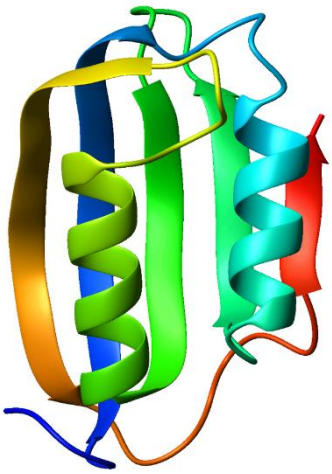


Figure 6 Ribbon diagram of acylphosphatase , which adopts a ferredoxin fold with an extra β-strand at the C-terminus (shown in red). The ribbon is colored from blue (N-terminus) to red (C-terminus).[25]

During the realization of this project, there are several technical competences than we want to take into consideration. The aim of this project is to fully accomplish satisfactory all the objective and technical competences that we have established. Therefore, we will need to develop and evaluate an interactive system that show complex information, and its application to solve person-computer interaction problems. In our project, we will try to reach a new level of person-molecule interaction, trying to solve the difficulties than the researches have when they need to visualize in an ease way all the molecules parts and residues groups and focus in particular groups or parts of the molecule.

Another very important scope will be to evaluate the computational complexity of the problem, learn the algorithmic strategies which can solve it and recommend and develop and implement the solution which guarantees the best performance according to the established requirements. One of the main ways to solve computational complexity of the problem will be developing strategies and algorithms which has the best computational performance and tries to reduce as much as possible the computational cost in order to get the best performance.

If we want to archive all the scope goals, we will need to follow a strict methodology. The appropriate scheduling and agile working methods will provide us with the flexibility we need to get satisfactory results faster. So now we will go over the main possible obstacles and which methodology we will take in order avoid or solve them.

The first point we will talk is about the timetable. Despite this project will take a whole semester, a bad scheduling and organization can be a big obstacle that could lengthen the development period. To avoid delivery problems, a strict timetable will be followed which involves working daily and with constant routines in order to not lose the workflow. Furthermore, the project developer is going to meet the director weekly and will talk about the project development and progress.

Another important point is the Bug testing. Considering than molecular dynamics is a complex field the number of bugs in the project software can be overwhelming. Because we will need to implement thousands of steps, we could do a mistake really easily, carrying it over the whole execution. In order to solve the problem of the bugs a series of unit tests will be run during each phase of the project in order to ensure that no more bugs are present. This unit tests will consider each possible case than an input could contain. One of the main ways to solve it, will be evaluating the computational complexity of the problem and develop an algorithm which has the best performance and tries to reduce as much as possible the computational cost.

Last but not least, the use of monitoring tools during all the programming steps will monitor the evolution of the project development. One of the main things that we want to control is the computational cost. Knowing every time, the computational cost of our code is really important for the correct development of the practice. As we discussed in the last chapter, the steps to render a molecule can be overwhelming if they are not computed correctly. In fact, we want a real-time rendering with a stable framerate and a convenient usage of the GPU in order than all the Virtual Reality ready computers could run the program without any issues.

1.5. Stakeholders

On this section we will explain the stakeholders than will take part in the project:

1.5.1. Project developer

The project developer will be the person in charge of the research of the required data and resources in order to solve the problem, implement the software and all the mathematical calculations needed and document each steps of the project. The project developer is also responsible of the project management and is the person in charge of the accomplishment of all the deadlines. At the end of the deadline period, the project developer has to have implemented the molecular viewer as well as the final report and all the required documentation. Furthermore, the developer will have to make a presentation of the project in front of his Director and co-director showing the results of the developed project.

1.5.2. Director and co-director

The director and de co-director are the persons in charge of guiding the project developer, helping him in any question or matter and providing any advice than the project developer could require. Their roles are fundamental in the project not only because they will follow and supervise each steps of the projects, but they will determine and correct any possible errors in the project and will help the project developer to solve it. The director of the project will be associate professor of Facultat d'Informàtica de Barcelona (UPC) and ViRVIG Group member, Pere-Pau Vázquez.

1.5.3. Beneficiaries

As well as we mentioned before, molecular dynamics is the main field of this project, so this work will try to help in the development of the optimization and rendering of the molecular projection and also improve the user interaction between them. Other areas than will be beneficiated of this project will be in biochemistry and biophysics, where the 3-dimensional representations will help the researchers and scientists to visualize the molecules in a more clear and comfortable way making his work more ease.

2. Project planning

The correct planning and scheduling of the academical semester is essential for the correct realization of the project. A good planification of your dedicated time can make you achieve all the goals and needs of your project, as well as giving you an action plan and alternatives in case of any problem appears. The initial task organization and schedule planning could be updated and revised during the course of the project.

For the planification of this project, firstly we are going to talk about the task description and all the steps than the developer will make in order to develop the project, then we will talk about the project management and the Gantt chart and finally we will talk about the alternatives and action planning in case any trouble appears.

2.2. Task description

2.2.1. Acquire knowledge about molecular dynamics and Unity engine

The correct learning and understanding of the molecular concept and structure as well as its representation and codification methods is one of the main aims of this point. Furthermore, ending to familiarize with the Unity engine and its shading methods would allow us to perform more efficient representation methods.

The main source than we will be using to learn about the molecular and protein environment will be the RCSB Protein Data Bank (PDB) website. This website developed by the Worldwide Protein Data Bank Foundation [12], is builder upon the data and resources of researches made by investigators about molecular structures and information than is coded in in a specifically developed programming language (.pdb) format. The website also provides a specific domain for its learning [13]. We will also use the resources of different pdf files of books chapters where the different representation methods are described in a detailed and rigorous way. [14] [15] [16]

In order to acquire the Unity engine background and usage we will make use of the main Unity documentation site [17] that provides the full documentation and some tutorials for its understanding. We will also take a look at the *catlikecoding* tutorials [18] that provides many useful tutorials for advanced rendering. Learning the Cg language will be a very important part. Cg is a graphical programming language based on C, that was developed by Nvidia and it is used for programming the Shaders of Unity. We will use the documentation provided by the Nvidia Developer zone [19] in order to acquire the basics of this language.

2.2.2. Build a PDB file parser

Once we have acquired all the background about the molecules and the basics about the PDB file format we will need to build a parser than will interpret the .pdb files and parse it to C#, then is the main language used by unity scripting. Once a PDB is read by our parser, we will be able to extract atom coordinates, align proteins, and compute its residue value among other things. The parser will be provided with many utility libraries than will ease our work. This utility libraries will split .pdb files into separate models, chains, or domains, align two conformations of the same protein or compute the distances between corresponding atoms in the two candidate structures (cRMS) or their corresponding intra-molecular distance matrix entries (dRMS), where measures are defined as the root mean square (RMS). [20]

2.2.3. Use of sphere Impostor and GPU rendering techniques in Unity

In order develop a fluent an efficient visualizer and acquire a good GPU performance we will need to take use of many techniques in order to reduce the number triangles and vertices and its drawing calls. The impostor technique makes use of the lighting computations to make an object appear to be something entirely different from its geometry. To attempt this, we will use the vertex shader to compute the vertex positions of a square in clip-space. This square will be in the same position and width/height as the actual sphere would be, and it will always face the camera. In the fragment shader, we will compute the position and normal of each point along the sphere's surface. By doing this, we can map each point on the square to a point on the sphere we are trying

to render. The idea is that we're actually drawing a square, but we use the fragment shaders to make it look like something else [21]. To implement this technique, we will follow a reference of this technique implemented in OpenGL [22] and will try to apply it in Cg Language in Unity as well. We will also apply some rendering techniques such as the Phong Shading. Additionally, it would be appropriate to build a Tester scene where a boatload of impostors is rendered and implements GPU instancing in order to render the same mesh multiple times in one go [23]. The Scene will also show the graphical statistics, allowing us to get a clear knowledge of our capacities and limitations.

2.2.4. Implementation of the basic visualizer

When the parser and render techniques tasks are over, we will be ready to build the visualizer. This task will consist on building a molecule visualizer that will be able to read any .pdf file from the computer and represent it Space-filling and Ball & Sticks graphical representations, as well as other molecule properties such as its residues.

2.2.5. Incorporation of Virtual Reality's interaction tools

This task consists of implementing and providing the tools than the User will use to interact with the molecule projection. The developer will have to upgrade the first version of the visualizer and make it VR ready, allowing the usage of the VR glasses and the VR controllers in order to interact with the molecules allowing him to rotate, expand, mask or visualize each atom and chain of the molecule. The visualizer will present a well implemented an easy to understand users' interface and will allow us to interact with the three different molecular representations, each with its unique characteristics. The controllers will be fluent and will facilitate the selection of all the molecule parts and components. For this task we will require of the HTC Vive glasses resource as well as VR ready graphical card.

Figure 7. An HTC Vive Headset with two wireless handheld controllers and two base stations.

2.2.6. Final implementations and optimizations

The final task workload will be conditionate by the amount of remaining time than we will have. A bigger timeframe will allow to the developer to try to implement some extra techniques to the project such as ambient occlusion or other shading techniques such as toon shading. Also, some we will try to implement some final GPU rendering optimizations. Furthermore, a series of test programs will be implemented in order to debug the project from possible errors and bugs, so the current code is refined. Additionally, the profiling of the data of each project step and test will be captured in order to see the performance improvements of our project.

2.3. Gantt chart & Estimated time

The Gannt chart is a graphic tool that illustrates your project schedule and the relationships between the activities and the current project status. We have used the free source software *teamgantt*[24] in order to make our Gantt chart:

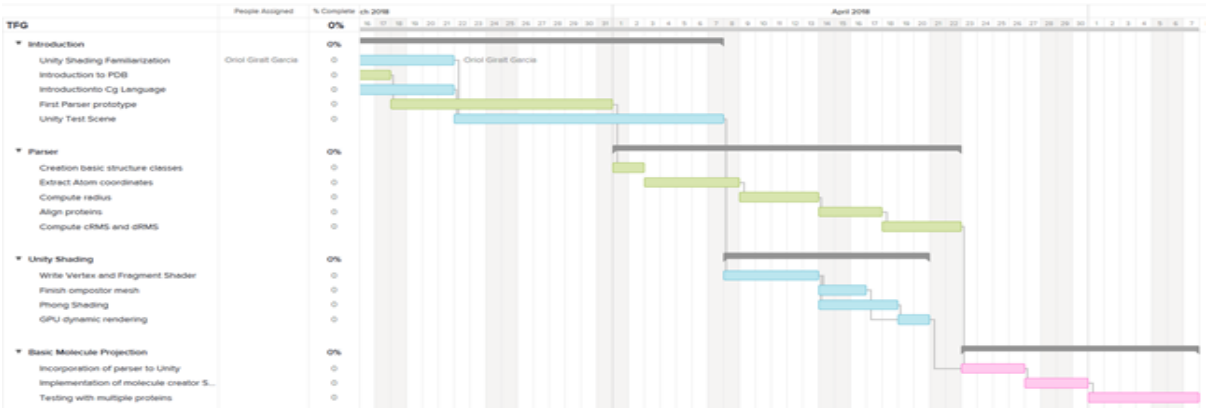


Figure 8. Screen capture of teamgantt web source software. The interface shows a calendar where each task is represented by colored bars. The width of the horizontal bars in the graph show the duration of each activity and its dependencies are marked with connected segments.

If we establish an approximated duration in hours of each task represented in our Gantt chart, we could estimate the overall duration of our project:

	Estimated duration (h)
Background in molecular dynamics	50
Background in Unity	30
Background in Cg Language	15
Parser Setup	50
Molecule Impostor Setup	70
Molecule Projection Script	120
Molecule Halo Script	25
VR interaction tools	90
Final optimization / implementations	50
Documentation and presentation	60
Total	560

Table 1. Estimated duration in hours of the proposed tasks based on the Gantt chart.

2.4. Obstacles and action plan

The methodology that we have chosen in order to realize the practice allow us to have a great planification of our time and have a good range view of our schedule. We could adapt our change any of our tasks if required and modify dynamically our schedule. However, there is the possibility than a hypothetical task lasts more than we expected, delaying the others. In this case we need an action plan in order to find an alternative solution so we can shorten the working time and we could be able to start with the upcoming task on time.

In this section we are going to talk about which are the potential problems and deviations we could experience during the project realization and which is the best action plan to solve it.

2.4.1. PDB complexity

Considering the great complexity than some proteins present, there is a possibility that we could not be able to parse correctly the information of the molecule. If there are only problems with the most complex proteins, we could ask for help to our director but, if not, there is the alternative plan to parse another PDB file which is coded in XML. An appropriate XML parser could convert this information for us with more ease.

2.4.2. Impostor rendering problems

It could be than, for some reason, we are not able to implement the impostor technique in the unity shaders. In this case, we could build up a sphere out of a cube. The implementation of the shader could be easier.

2.4.3. Bad code implementation

During the implementation of the project, we have to be care with the code optimization. A bad optimization of our resources could increase exponentially. An

exhaustive series of tests will be done in order to constantly check that our code is refined and efficient.

2.4.4. Unavailability of the HTC Vive Glasses

The high demand of the Google glasses at the University could delay our project debug and test during specially during our last stages. The lack of the glasses could cause a great delay to our project schedule. In this case, we will need to try to have always some HTC Vive glasses available and a computer with a VR ready graphic card.

3. Budget and sustainability

3.1 Current domain of sustainability

Before starting this section, we are going to make a brief self-analysis of the sustainability of our project. In order to achieve a complete self-analysis, we will make a survey1 which include a broad definition of sustainability that includes its three dimensions: social, environmental and economic.

To understand more clearly the dimensional impacts of our project we have to look back on who are the beneficiaries of it; the biologists and investigators. The development of a new way of visualize the molecular structures that are registered by the whole community of biologists and investigators, could mean a big impact in the social and economic. Firstly, the improvements of the investigation tools would potentiate the possibility of new discoveries and inventions, providing a better acknowledgement of ourselves and the world composition and giving us new methods to develop new tools and ways to cure many different diseases that would decrease the population mortality while would improve the welfare state. Another important point to analyze is the economic impact that our project will have. The economic management knowledges provided during the lecture course of the degree, will give us the tools in order to execute a properly management of our resources and our fuds so we that we have the most optimal financial amortizations and benefits, allowing us to maximize the economic benefits.

Despite the social and economic strength of our project, we must not forget the environmental impact. The fossil fuels and chemicals that are used in order to build the chipsets have a huge negative impact in our environment, putting in danger the well-being of living creatures of the planet. One we conclude the survey we can conclude that we should take a brief look of the contamination and impact of the modern computers and technologies on the world.

3.2. Budget estimation

In this section we are going to estimate the budget cost of the project. We are going to divide the budget in different sections, depending on the type of resources we are taking into account, and we will estimate the costs of each one. Finally, we are going to estimate the final budget of the project by combining the previous budgets.

If we take into account the fact than our project will last for six months approximately and we estimate a useful life for each resource, we could calculate an approximate amortization of the project.

3.2.1. Hardware resources budget

The hardware resources budget is strictly composed by the hardware components than we will use for the realization of this project. The estimated useful life of each one is established to four years. After that period the hardware will mean to be obsolete.

Product	Useful Life	Price (€)	Amortization (€)
HTC Vive set	4 years	699,00 €	87,38 €
VR-Ready PC	4 years	877,24 €	109,65 €
Hardware peripherals	4 years	75,00 €	9,35 €
TOTAL		**1.651,24 €**	**206,38 €**

Table 2 Hardware resources budget.

3.2.2. Software resources budget

The hardware resources summarize the costs of the software we are going to need in order to develop the project. The estimated useful life of the life and the established price of each license software will be estimated to four years as well.

Product	Useful Life	Price (€)	Amortization (€)
Windows 10 home	4 years	145,00 €	18,13 €
Github	4 years	Free	Free
Visual Studio Professional	4 years	1 756,30 €	219,54 €
Unity Software	4 years	Free	Free
TeamGantt	4 years	Free	Free
TOTAL		**1.901,3 €**	**237,67 €**

Table 3 Software resources budget.

3.2.3. Human resources budget

The human resources budget is composed by the group of people required to develop the project. Despite the entire project is fully developed by only one person, we will suppose than the team is composed by four main roles; the project manager, responsible only of the project planning. The software designer is going to do the task related to designing software, taking into account the user's needs and the algorithm complexity of the program. The software programmer will be the responsible of implementing all the required implementations proposed by the software designer. Finally, the software tester will be the responsible of elaborating stupas and performance and usability during the developing and final stages of the project. The total elapsed time of human resources work will be the same than the established time in the *Table 1*.

	Time	Cost per hour (h)	Cost (h)
Project manager	95 h	40,00 h	3.800,00 €
Software designer	145 h	30,00 h	4.350,00 €
Software programmer	150h	20,00 h	3.000,00 €
Software tester	100 h	15,00 h	1.500,00 €
TOTAL		**490,00 h**	**12.650,00 €**

Table 4 Human resources budget.

3.2.4. Total budget

The total budget cost will be composed by the sum of the results of the *Table 2, Table 3* and *Table 4*.

Concept	Cost
Hardware resources	206,38 €
Software resources	237,67 €
Human resources	12.650,00 €
TOTAL	**13.094,05 €**

Table 5 Total budget cost.

3.3. Sustainability analysis

Nowadays, a sustainability report is a must in any technological project. There are many different ways we could proceed, but in our project, we will use the sustainability matrix, where the project production (PP), the useful life and the dangers of our FDP will be contrasted with the three sustainability dimensions: the environmental dimension, the economical dimension and the social dimension. The following figure shows the results of this analysis:

	PP	Exploitation	Risks
Environmental	Consumption design	Ecological footprint	Environmental risks
	6 : 10	11 : 20	-9 : -20
Economical	Project Bill	Viability plan	Economic risks
	6 : 10	15 : 20	-3 : 20
Social	Personal impact	Social impact	Social risks
	7 : 10	18 : 20	0 : 20
Sustainability range	19 : 30	44 : 60	-12 : -60
	51/90		

Table 6 Sustainability matrix of the project. This matrix is based on the ideas of "The economy for the common good" by Christian Felber.[26]

3.3.1. Environmental impact study

The consumption design represents the impact of the environment along the realization of the FDP providing the energetical costs as well as the number of residues generated. For the correct development of the consumption costs, we will estimate the Kw consumed in the elapsed time of our project. We should have in mind that the consumption values will be approximated and the results could vary depending on the computer hardware or the elapsed time of work during the day. Therefore, to get an estimated value of the consumption, we will say that, during the approximated 140 days that will lasts the project, the 25% of the time the hardware will be working, the 15% will be on power safe mode and the rest of the time the hardware will be turned off. Additionally, we will suppose that for each hour, our hardware consumption (composed by the computer setup and the VR setup) will be 80.6W when they are turned on, 29,4W when they are on power mode and 5,1W when they are off 2. If we suppose this value, we could make an approximated estimation of how many W h we will spend during the project:

Estimated cost day(W*h) = 24h*(0.25*80.6W+ 0.15*29,4W+0.6*5.1W) = **662.88 W h**
Estimated cost of the project (W*h) = 140 days * 662.88 W h = **92.8 kW h**

The ecological footprint means the consumption impact that the FDP will have during his useful life. The estimators used to measure the footprint will be the same of the consumption design. If we take the previous results and we suppose a utility life for the hardware setup of 4 years, we will get the approximated consumption cost:

Utility life cost (W*h) = 4 years * 365 days * 662.88 W h = **966,649 MW h**

The environmental risks are the set of eventualities that could aggravate the environmental impact. Computers and its components are made of heavy metals and dangerous chemicals. These metals and chemicals contribute to global warming and also it causes water contamination and air pollution. Furthermore, there is a lot of people than makes a living of collecting these materials and exposing themselves to

dangerous working conditions and, in worst cases, initiating wars for economic interests. When we are buying new technologies such computers or VR headsets, we have to constantly have in mind that we are fomenting the social inequality and the degradation of the earth.

3.3.2. Economic impact study

In the last section, we have estimated the total budget cost of our project while we were taking into account the software, hardware and human resources. The total cost value stated in *Table 5* represents only the strict cost spent in realization of the project and since our main objective was to create a working visualizer, we will not estimate the costs of maintenance and improving versions. Implementing new versions of the current project would mean and increasement of the current development cost.

This project will also have low economical risks because of its affordable price and also because of his low number of human resources members. If we want to optimize the total budget costs of the project, we will need to take some considerations into account. The first one is that we cannot reduce the hardware resources costs. The established bill estimates the price for a VR-Ready Personal Computer with the required VR Glasses required to implement the visualizer. The price for the PC is an approximated value of the minimum cost than a PC will has, having the minimum required technical specs to run the VR Headset without compromising the system performance. The other consideration is than we will not be able to reduce the human resources bill. If we don't certainly know the exact hours than the project would take, it would be unwise to reduce the project working hours, because it could compromise our project objectives and scope. Finally, we could also try to reduce unnecessary costs such as the cost of any impression of documentation or the use of free Clouds and free licensed software so we can reduce the bill as much as we can.

We cannot certainly affirm than the project cost would make the project competitive but the fact than Virtual Reality Headsets are relatively new and that there is no current

molecular visualizer of the most notorious enterprises that includes VR interaction, makes the current project a good candidate for a paid application. Despite this project is not developed with the collaboration of any institution, there are many chemical and biological institutions that will be really interested in our project development and, because they don't dispone any molecular visualizer with virtual reality interaction methods, they would be potential buyers of our product.

3.3.3. Social impact study

In order to analyze the social impact of this project we will split this section in two parts: the first one is going to be the internal impact of the realization of this FDP while the other part will be the collective impact that will have this project on the society.

If we analyze the internal impact, we can conclude that the project developer not only will get a fundamental base in the biomolecular and biodynamics fields, but also will learn many different optimization techniques in order to represent them. The developer will also learn about parsing fundamentals and will get the skills to build a complete .pdb to C# parser. Finally, the developer will learn the fundamentals of Unity Software and will be able to map and adapt any project to virtual reality as well as will get the skills to design an interactive and user-friendly interface.

When we talk about the external impact on the society, we could number many different targets that will be related, either from direct or indirect way. Firstly, the biomolecular researchers and investigators. The fact of providing them with such a powerful visualization tool could make them more attentive and insightful when they have to analyze any molecule, giving them the chance to visualize molecular residues in a complete and clear way than the monitor screen doesn't presents. Moreover, this project will be a base for any developer or future student than wants to learn about molecular dynamics and projection and also for any person than wants to learn about optimization techniques and get a basis on the development of an interactive interface for virtual reality.

Finally, if we analyze the socials risks of this project, we can conclude that they are null. The aim of this project is to expand the present knowledges and the only usage of this FDP would only be for educational purposes.

4. Rendering molecular geometry

4.1. Complexity of rendering a sphere

The main geometrical form for molecular representation is the sphere. Geometrically talking, a sphere is a round solid figure, or it, with every point on its surface equidistant from its centre[30]. This implies that a sphere has infinite number of vertices. Therefore, if we want to render a quality sphere, we will need a really high number of vertices. Comparing the sphere with other basic geometrical forms such as the quadrilateral, the cube or the triangle, for example, the sphere is by far the more complex to render.

When graphics programmers face the problem of creating a geometry mesh for a sphere, trade-offs must be made between quality and construction, as well as memory and rendering costs. Now, we will briefly introduce four different methods, and then will make a brief comparison of the methods in order to see which method suits our needs.

- **UV Sphere**: The UV sphere is the most common used technique to render a sphere. This method divides the sphere using meridians (lines from pole to pole) and parallels (lines parallel to the equator). While we are near the equator, it will produce faces with bigger area near the equator, whereas on the poles it will produce faces with smaller area. All the faces will be made of quads.

- **Normalized Cube:** This method uses a uniformly subdivided cube, where each vertex position is normalized and multiplied by the sphere radius. This creates a non-uniformly subdivided sphere where the triangles closer to the centre of a cube face are bigger than the ones closer to the edges of the cube.

- **Spherified Cube:** As well as normalized cube technique, this method is based on a subdivided cube, but it tries to create more uniform divisions in the sphere. The area of the faces and the length of the edges suffer less variation, but the

sphere still has some obvious deformation as points get closer to the corners of the original cube.

- **Icosahedron:** The last technique is based on the icosahedron geometric figure. An icosahedron is a solid figure with twenty plane faces, especially equilateral triangular ones[31]. To get a higher number of triangles we need to subdivide each triangle into four triangles by creating a new vertex at the middle point of each edge which is then normalized, to make it lie in the sphere surface. Sadly, this breaks the initial properties of the icosahedron, the triangles are not equilateral anymore and neither the area nor the distance between adjacent vertices is the same across the mesh. An added problem with this method is that we can only increase the number of faces by four each time[33].

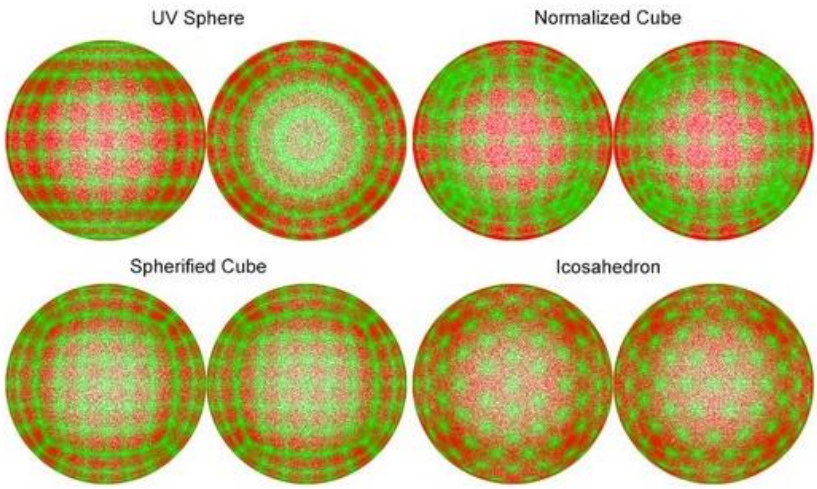


Figure 9 Visualization of the surface distance error. The red part means than the distance from the sphere center it's not unitary, thus there is an error on the surface distance. The green part means the surface perfectly fits the theorical postulated sphere.

Despite we use a simple UV sphere for our atom representation, rendering a sphere mesh will not work for us since our performance would decrease drastically. Therefore, we must need to find another sphere rendering techniques of sphere representation with lesser render cost in order to improve our overall performance, as well it still looking like a rounded theorical sphere.

4.2. Billboard Impostor

4.2.1. What you see is a lie

As we saw in the previous chapters, rendering a sphere for each atom of the molecule can be really expensive in terms of graphical rendering, not even mentioning if we add the cost of rendering the cylinders for each bond. At this point, we need to find a technique in order to try the render calls as much as possible. One of the most common techniques used to reduce the number of tris on the scene is the impostor. The impostor tricks the viewer in order to simulate a three-dimensional object while we only render a simple two-dimensional polygon form, typically a quadrilateral. Thus, before we continue, we will introduce some concepts in order to get some knowledge about how impostors are made of and the we will proceed to deepen to the vertex and fragment shader technical implementations.

The first concept we will introduce the *sprite* term. In computer graphics, a sprite is a two-dimensional image or animation that is integrated into a larger scene. Initially used to describe graphical objects handled separate from the memory bitmap of a video display, the term has since been applied more loosely to refer to various manner of graphical overlays[26]. Originally, sprites were a method of integrating unrelated bitmaps so they appeared to be part of the normal bitmap on a screen, such as creating characters and forms that can be moved on a screen without altering the overall screen data. The problem of a sprite in 3D graphics is that the extra space makes it a non-viable, since a sprite has no profundity. However, if we constrain the alpha channels to face the camera, we could partially solve the depth problem.

This previous case allows us to introduce the second term, called *billboards* or *Z-sprite*. A billboard is a two-dimensional polygon in three-dimensional space that is always rotated to face the viewer and that has an image texture-mapped onto it so that the image on the polygon seems to be a three-dimensional object in the scene[27]. Typically, a billboard is a partially transparent, textured quadrilateral. The texture map is an image of the object represented. The quadrilateral is partially transparent in some cases, since

the object's image does not entirely cover the quadrilateral. We could list the different types of billboards based on the three vectors of interest; the up vector, normal and rotation vector (perpendicular to up and normal). In our case we are interest on the axial billboards, in which the textured object does not normally face straight toward the viewer. Instead, it is allowed to rotate around the whole world-space axis and align itself to face a viewer as much as possible within this range[28].

Now that we got some acknowledgements, we can procedure to properly define what is an impostor. An *impostor* is a billboard that is created by rendering a complex object from the current viewpoint into an image texture, which is mapped onto the billboard. The impostor can be used for a few instances of the object or for a few frames. The impostors are useful for rendering distant and numerous objects rapidly, since a complex model is simplified to a simple image. A possible alternative could be to instead use a minimal level of detail (LOD) model. However, such simplified models often lose shape and color information. Impostors do not have this disadvantage, since the image generated can be made to approximately match the display's resolution.

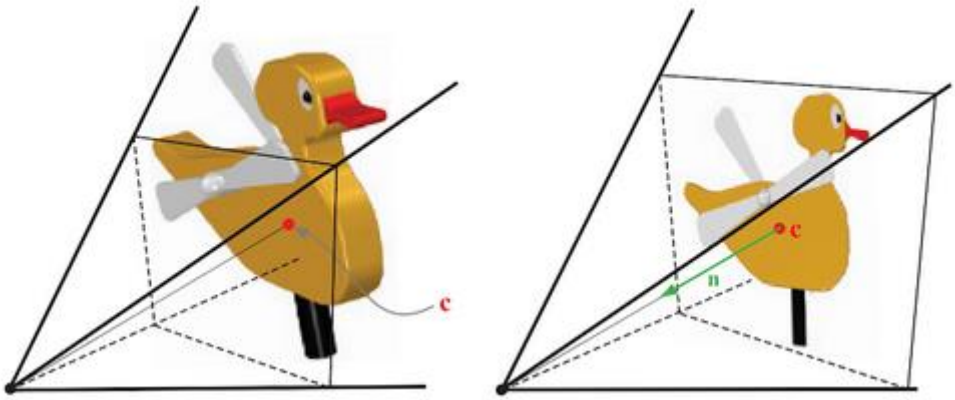


Figure 10 Impostor example. At the left, an impostor is created of the object viewed from the side by the viewing frustum. The view direction is toward the center, c, of the object, and an image is rendered and used as impostor texture, shown at the right of the figure. At right, the texture is applied to a quadrilateral. The center of the impostor is equal to the center of the object, and the normal (emanating from the center) points directly toward the viewport[aad].

Before we render the object to create the impostor image, the viewer is set to view the center of the bounding box of the object, and the impostor rectangle is chosen so that it points directly towards the viewport. The size of the impostor's quadrilateral is the smallest rectangle containing the projected bounding box of the object. The alpha channel is cleared and set to zero, and wherever the object is rendered, alpha is set to 1.0. When the camera or the impostor is moved, the resolution of the texture may be magnified, which may break the illusion. Thus, the impostor image needs to be updated.

4.2.2. Impostors

As we know, the two geometrical forms that we will need to represent the molecule and must be impersonated, are the sphere and the cylinder. For spheres and cylinder rendering, a custom vertex and fragment shaders are used in order to achieve it.

In case of the spheres, four identical vertices are sent to the GPU corresponding to the center of the sphere, as well as four extra mapping coordinates that represent the corners of the sphere impostor square (-1, -1; 1, -1; -1, 1; 1, 1). The vertex shader then takes each of the four coordinates, transforms them according to the model view and orthographic matrices (to handle rotation and scaling of the model, as well as the rectangular nature of the OpenGL scene), and then displaces them relative to the viewer using the impostor space coordinates so that the square is always facing the user.

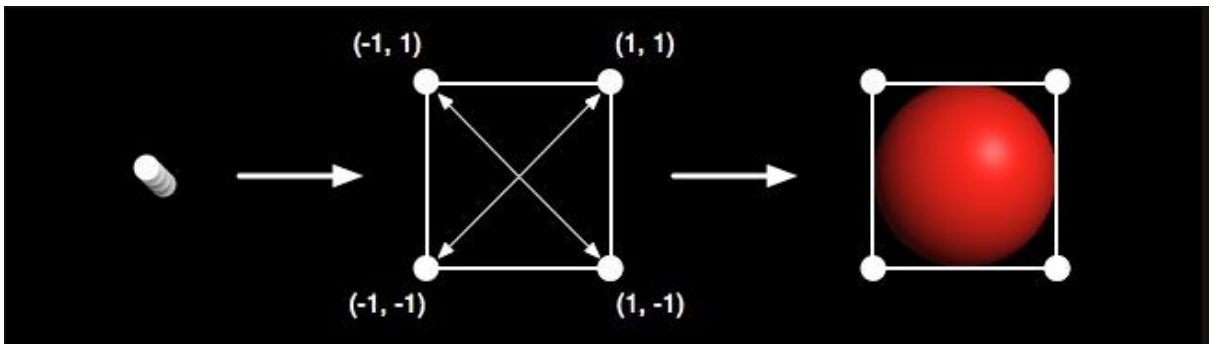


Figure 11 Process of rendering a sphere impostor[29]

Once the square has been generated, every fragment within that square (roughly, every pixel) needs to be colored as if it were from a lit sphere behind that point. For this, the normal of the sphere at that point is calculated as the vector (impostor space X, impostor space Y, normalized depth). The calculation and use of the depth component

are discussed later in the previous chapter. The dot product of the normal and the light direction is calculated and used to determine the strength of the illumination at that point for both ambient light and the specular highlight. The resulting color is written to the screen at that point, except for fragments that lie outside of the sphere, which are output as transparent[29] and therefore, discarted.

Cylinders are a little more complicated, but the same general process applies. Four vertices are fed to the GPU (two for the starting coordinate and two for the ending coordinate), along with four impostor space coordinates and four direction vectors that point from the beginning to the end of the cylinder center. The beginning and ending points are transformed at each vertex, then by using the transformed directions the vertices at each end are displaced perpendicular to the axis of the cylinder as viewed by the user. Additionally, the vertices at one end of the cylinder are displaced along the axis to account for the curving out of the cylinder at that end[29]. This is shown in the below *figure 12*:

Like the spheres, the normal at each fragment on the cylinder is calculated to use in determining illumination, but the calculations here aren't as simple as those for the spheres, since the illumination of the cylinder bases are not trivial. Thus, since the bases will be covered by the sphere impostors representing each molecule, we will skip this redundant process in order to simplify the as much as possible this calculus. So, in order to render a cylinder, many values are calculated in the vertex shader for points on the

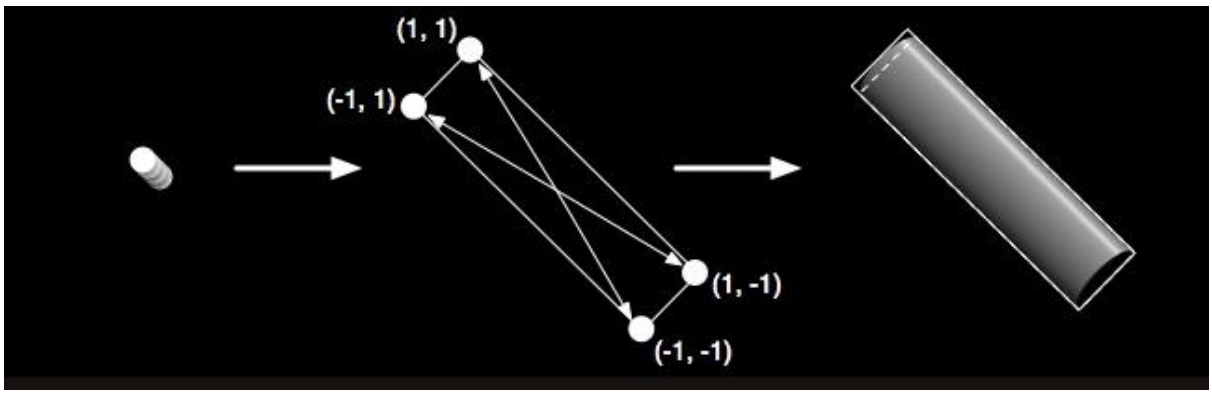


Figure 12 Process rendering a cylinder. See that this render example takes into account the illumination of the cylinder plane bases and our rendered cylinder impostor will not.

center axis, then adjusted in the fragment shader as a function of the distance from the axis.

In order to understand the processed previously described, we need to deepen to a more technical description of the processes. Thus, we will analyze the vertex and fragment shader and will talk about the events that are taking part into the process of building a basic sphere impostor. For first version of the vertex and fragment shader, we will try to build the most basic impostor form, keeping the shaders as simple as possible. Moreover, we will talk about its lacks and problems and what we can do to solve it.

4.2.2.1. Vertex shader overview

In order to keep the vertex shader as simple as possible, we will not declare any input variable. However, it will still use an input variable: `gl_VertexID`. This is a GLSL built-in input variable and it contains the current index of this particular vertex, starting the index with zero value and pursuing.

Since we are rendering a quad, we render 4 vertices as a `GL_TRIANGLE_STRIP`. This means the `gl_VertexID` will vary from zero to three. Because we're trying to render a square with a triangle strip, the order of the vertices needs to be appropriate for this. Thus, we will need a switch case statement that determines which vertex we are rendering and which corner of the impostor we are on. Note that the current index of the vertex and the mapping coordinate of the impostor have to be equivalent to the same corner of the impostor because we will pass to the fragment in which mapping corner coordinate, we are on. Later, on the fragment shader overview we will see why we need it.

After determining which vertex to render, we use the radius-based offset value of the sphere as a bias to the camera-space sphere position. The Z value of the sphere position is left alone, since it will always be correct for our square. After that, we transform the camera-space position to clip-space as normal.

4.2.2.2. Fragment shader overview

A basic lightning shader equation must need the position and normal value in camera-space, and the job of the fragment shader it's to provide them. However, the position and normal of the billboard will not have the same values than the position of normal of the impostor. In order to compute the position and normal, first we need to find the point on the sphere that corresponds with the point on the billboard that we are currently on and, in order to do that, we need a way to tell on which part of the impostor quadrilateral we are. Using `gl_FragCoord` will not help, as it is relative to the entire screen and we need a value that is relative only to the impostor square. That is the purpose of the mapping variable. When this variable is at (0, 0), we are in the center of the square, which is the center of the sphere. When it is at (-1, -1), we are at the bottom left corner of the square.

However, before we start computing the impostor point, we will require of a simple distance check. Since the size of the square is equal to the radius of the sphere, if the distance of the mapping variable from its (0, 0) point is greater than 1, then we know that this point is off of the sphere.

If the point is not under the sphere, we will just discard it. The `discard` keyword is a command exclusive for fragment shaders. It tells OpenGL that the fragment is invalid and its data should not be written to the image or depth buffers. This allows us to carve out a shape in our flat quadrilateral billboard, turning it into a circle.

Computing the normal is based on simple trigonometry. The normal of a sphere does not change based on the sphere's radius. Therefore, we can compute the normal in the space of the mapping, which uses a normalized sphere radius of 1. The normal of a sphere at a point is in the same direction as the direction from the sphere's center to that point on the surface. Computing the position is also easy. The position of a point on the surface of a sphere is the normal at that position scaled by the radius and offset by the center point of the sphere.

The computation process can be overwhelming in a three-dimensional space case, but it's easier to understand if we look at the two-dimensional case. To have a 2D vector direction, we need an X and Y coordinate. On our case If we only have the X, but we know that the vector has a certain length, then we can compute the Y component of the vector based on the Pythagorean theorem:

$$X^2 + Y^2 = R^2$$

$$\mathbf{Y} = \pm\sqrt{R^2 - X^2}$$

As we saw above, we just simply use and upgraded three-dimensional version. We have X and Y from mapping, and we know the length will be always 1.0. Thus, computing the Z value is easy enough. And since we are only interested in the front-side of the sphere, we know that the Z value must be positive.

If we take a look at the resulted impostor on the first capture of *Figure 14*, we can see that our sphere looks simply, but it has a spherical form. This impostor could be useful is someone wants to render a scene with a couple of spherical impostors on the background could be pretty useful, however, what if we try to make a first plane of the impostor or we want to position it on the sides of the camera viewport?

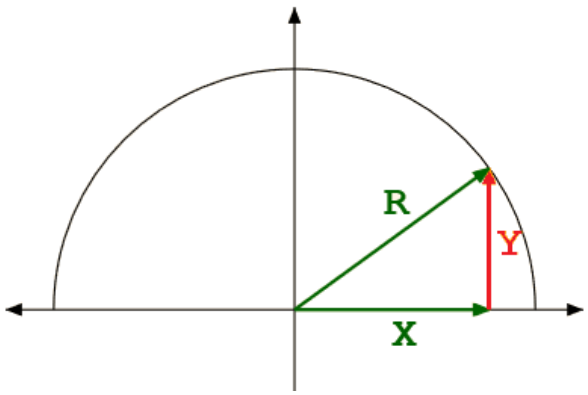


*Figure 13 Two-dimensional circle point computation schema. As we can see the **Radius** is equal to the triangle hypotenuse while the **X** and **Y** coordinates are the cadets.*

In those cases, the impostor is far from being a sphere. Apparently, its looks like the impostor is being clipped in perpendicular cuts from the scene. This makes the impostor useless for the representation of a complete molecule representation, since the size of the molecule could be such big than the atoms would be clipped from the viewport scene. Thus, we need to know why the impostor it's only portrayed properly if it's middle

centered from a middle plane, and gets clipped gradually if we start to move the impostor to the sides of the viewport or we try to make a first plane from it.

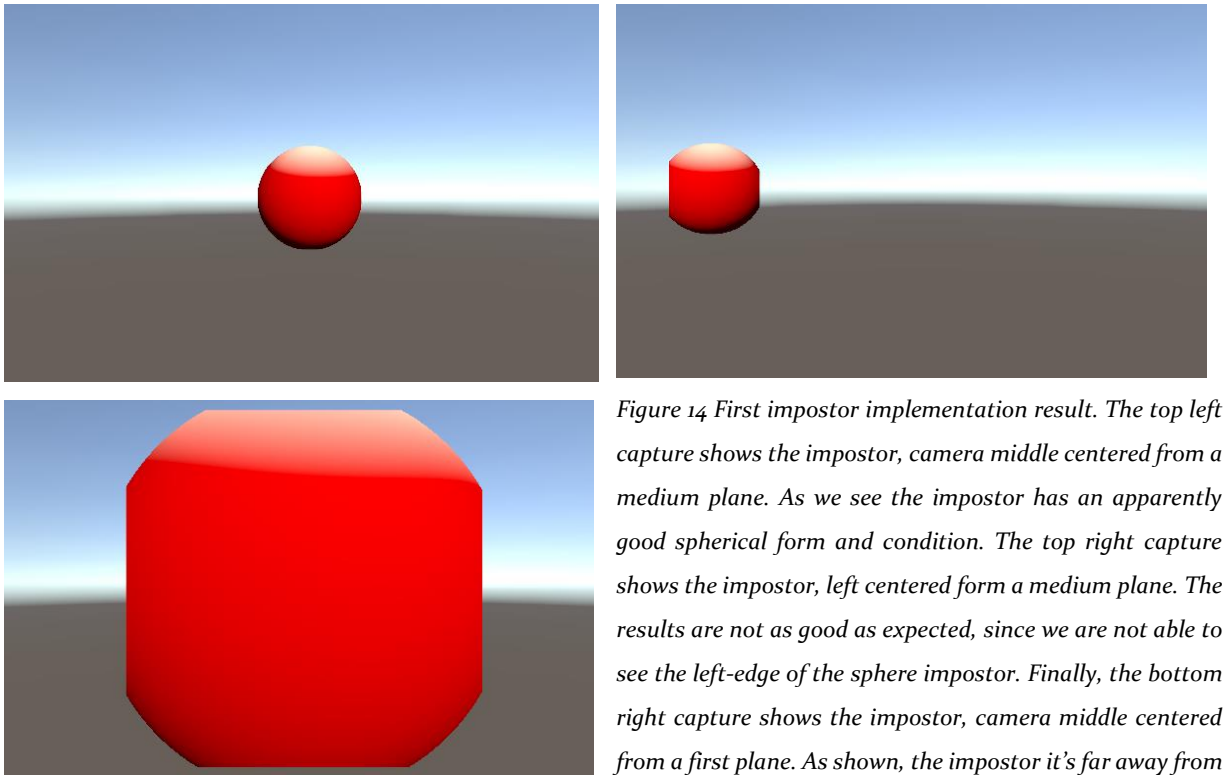


Figure 14 First impostor implementation result. The top left capture shows the impostor, camera middle centered from a medium plane. As we see the impostor has an apparently good spherical form and condition. The top right capture shows the impostor, left centered form a medium plane. The results are not as good as expected, since we are not able to see the left-edge of the sphere impostor. Finally, the bottom right capture shows the impostor, camera middle centered from a first plane. As shown, the impostor it's far away from having a spherical form. This could be a principle than our impostor is too simple.

In the next chapter, we will talk about why the impostor representation is being clipped and how we can solve it as well as future inconveniences they could appear during the representation of our molecule. Furthermore, we will discuss about the correct chicanery and methods we can use to improve our impostor in order to try to get a representation of a sphere as much real as possible.

4.2.3. Correct guidelines

As we know, the proper praxis and manners we employ to implement the algorithms and computations will be the key to reach our objectives as well as implementing an efficient and proper representation of a sphere.

The first problem that we had was that the impostor was being clipped as soon as is being displaced from de middle centered of the viewport as well as it need to be in a medium or far plane, as it starts to get a quadratically form as soon we start to get a first plane of the impostor.

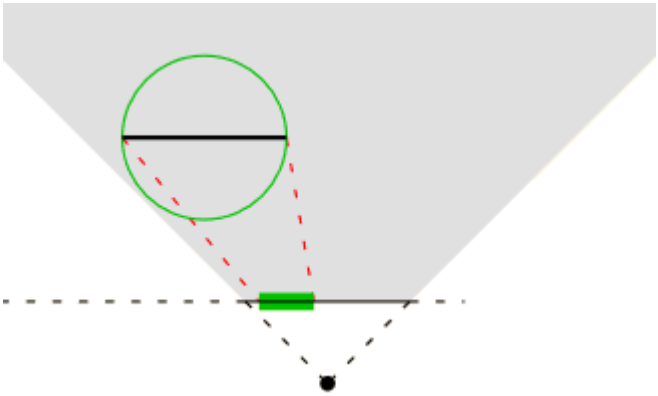


Figure 15 Two-dimensional representation of our scene. The dark line through the circle represents the square we drew. The green rectangle represents the camera viewport. Finally, the dashed red line represents the camera viewing angle

Let's try to analyze the purpose of our impostor algorithm implemented; we are trying to render a sphere down a flat quad. It could be than the sphere is wider than a simple quad. We did some computations to get the proper Z value of the represented sphere, and we did it in using the camera-space Z direction but, we forgot that the mapping between the impostor surface and the sphere is static; it does not change based on the viewing angle[32] and this is why our computations fail.

The problem can look overwhelming, but let's consider the two-dimensional case represented in the *Figure 15*. As we see, the sphere representation it's not completely inside our camera viewing angle. Thus, when we are viewing the sphere off to the camera side centered, such as in this case, we should not be able to see the left-edge of the sphere facing perpendicular to the camera because it's being clipped. Furthermore, we should see some of the sphere on the right that is behind the plane.

Rather than start computing the exact extent of the sphere's area projected onto a square, we can take easier ways such as, for example, it would be much easier if we just

make the square bigger. Thus, if we increase the square size a 50%, as we assure that each sphere point will be inside the range. Of course, we will end up rasterizing more than the strictly necessary, but it's overall will be much simpler.

However, even though the box correction is going to avoid the sphere impostor clipping, our impostor appearance still looks quite simply, very color saturated and with a poor lightning reflection effect.

At this point, we should take a look at our implemented code, and reformulate algorithm math's. As we know, our algorithm implementation works nice if the spheres are somewhat small represented in the viewport. But if the spheres are reasonably close to the camera, they don't look like a sphere. This is the most important fact the algorithm has to take into account, since in our viewer, we will want to approach as much as possible to our molecules.

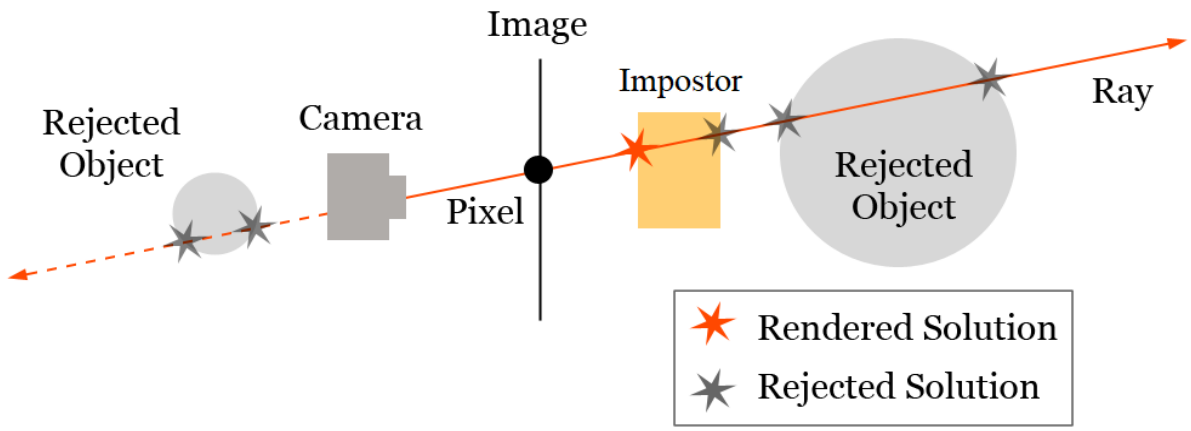


Figure 16 Two-dimensional representation of our scene. The pixel point will be the pixel represented in the window's image. The ray traced is represented in full gradient orange for the frontwards cast and a doted orange line for the backwards cast. Each scene object will be intersected twice. The rendered solution will be the first impostor hit.

Therefore, we will try to *ray trace* the position and normal of a sphere at a certain sphere point in our algorithm implementation. In optics, a *ray* is an idealized model of light, obtained by choosing a line that is perpendicular to the wave fronts of the actual light, and that points in the direction of energy flow[34].

In computer graphics, it works by tracing a path from an imaginary eye (camera) through a particular pixel in a virtual screen, and calculating the color of the object visible through it. In our case, we will not be implementing a full ray tracing algorithm; instead, we will use it only to get the position and normal of a sphere at a certain point.

As we have defined before, a ray is a line, therefore it will have a position and direction. In addition, it also means that this line will have an infinite number of points. Thus, each point can be expressed with the following equation:

$$\vec{P}_{(t)} = \vec{D}t + \vec{O}$$

$$\vec{D} = Ray\ Direction$$

$$\vec{O} = Ray\ Origin$$

For each fragment call, we want to detect the point which it hits the sphere, if any. If the ray intersects the sphere, then we use that point and normal for our lighting equation. Given a sphere radius $\boldsymbol{R}$, we know every point $\vec{P}$ will be from a distance $\boldsymbol{R}$ from the sphere center. Therefore, the equation for computing the points located on the sphere could be like:

$$\left\|\vec{P} - \vec{S}\right\| = R$$

$$R = Sphere\ Radius$$

$$\vec{S} = Sphere\ Center$$

At this point, we can substitute our ray equation for $\vec{P}$:

$$\left\|\vec{D}t + \vec{O} - \vec{S}\right\| = R$$

Our ray traced goes from the camera into the scene and, since we're in camera space, the camera will be at the origin coordinates. Therefore, $\vec{O}$ will have a value of zero and

can be eliminated from the equation. Furthermore, we will also want to get rid of that length norm. One way to do it is to re-express the sphere equation as the length squared[35]. So, we get the following equation:

$$\left\|\overrightarrow{\boldsymbol{D}}\boldsymbol{t} - \vec{\boldsymbol{S}}\right\|^2 = \boldsymbol{R^2}$$

The square of the length of a vector is the same as that vector dot-product of itself, and because the dot product follows the distributive rule, we can get the quadratic equation:

$$(\overrightarrow{\boldsymbol{D}} * \overrightarrow{\boldsymbol{D}})\boldsymbol{t^2} - \boldsymbol{2}(\overrightarrow{\boldsymbol{D}} * \vec{\boldsymbol{S}})\boldsymbol{t} + (\vec{\boldsymbol{S}} * \vec{\boldsymbol{S}}) = \boldsymbol{R^2}$$

As far as t is the equation variable, we can get a lot of useful information about the ray casted if we solve the equation:

$$\boldsymbol{ax^2 + bx + c = 0}$$

$$\boldsymbol{x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}}$$

$$\boldsymbol{a = (\overrightarrow{D} * \overrightarrow{D}) = 1}$$

$$\boldsymbol{b = -2(\overrightarrow{D} * \vec{S})}$$

$$\boldsymbol{c = (\vec{S} * \vec{S}) - R^2}$$

The discriminant (under the square root) is the first function part we will observe; If the discriminant is negative, the equation will have no solution. In terms of our ray traced, will mean the ray misses the sphere. However, if the discriminant is positive will mean it will hit the sphere. As we know, the discriminant will be positive, if and only if, the polynomial has two real roots.

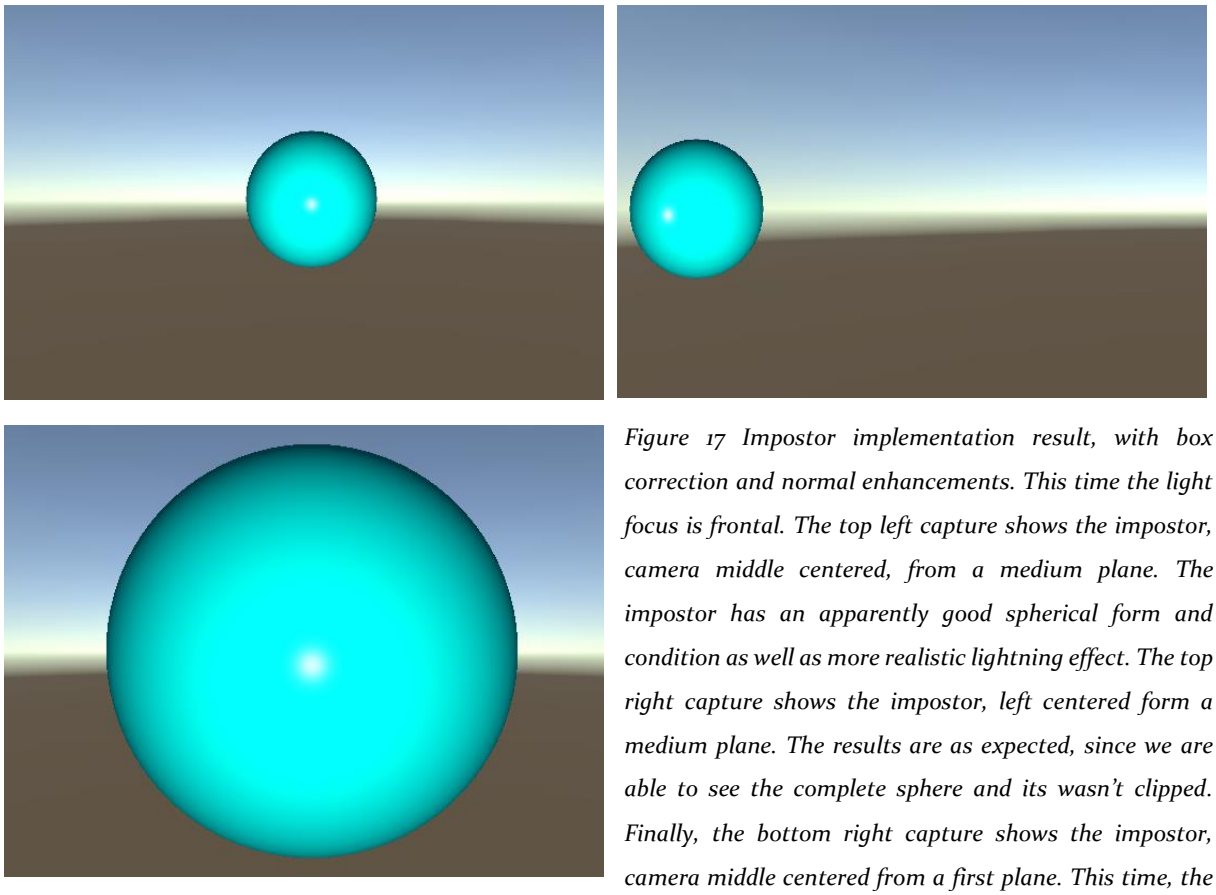


Figure 17 Impostor implementation result, with box correction and normal enhancements. This time the light focus is frontal. The top left capture shows the impostor, camera middle centered, from a medium plane. The impostor has an apparently good spherical form and condition as well as more realistic lightning effect. The top right capture shows the impostor, left centered form a medium plane. The results are as expected, since we are able to see the complete sphere and its wasn't clipped. Finally, the bottom right capture shows the impostor, camera middle centered from a first plane. This time, the impostor has a perfect rounded spherical form.

As you may recall, the square root can be either positive or negative. This gives us two t values, which makes sense; the ray hits the impostor in two places (*See Figure 16)*: once going in, and once coming out. The correct t value that we're interested in is the smallest one (since it's the one we will be observing). Once we have that value, we can use the ray equation to compute the point. Finally, with the point and the center of the sphere values, we can compute the normal[32].

Despite we solved the perspective and lightning problem of our sphere impostor, there still missing an issue to solve. When we render an atom sphere, the geometric element that take part into the representation, are constantly colliding each other. Even though we've made it look like a mathematically perfect sphere, it does not act like one into the depth buffer. As far as it is concerned, it's just a mere circle, not sphere (see *Figure 18*).

Part of the fragment shader's output is the depth value. This value is used on the depth buffer. In computer graphics, *depth buffering*, also known as, *z-buffering* is the management of image depth coordinates in 3D graphics, usually done in hardware and sometimes in software. It covers the solution of visibility problem, which is the problem of deciding which elements of a rendered scene are visible, and which are hidden [36].

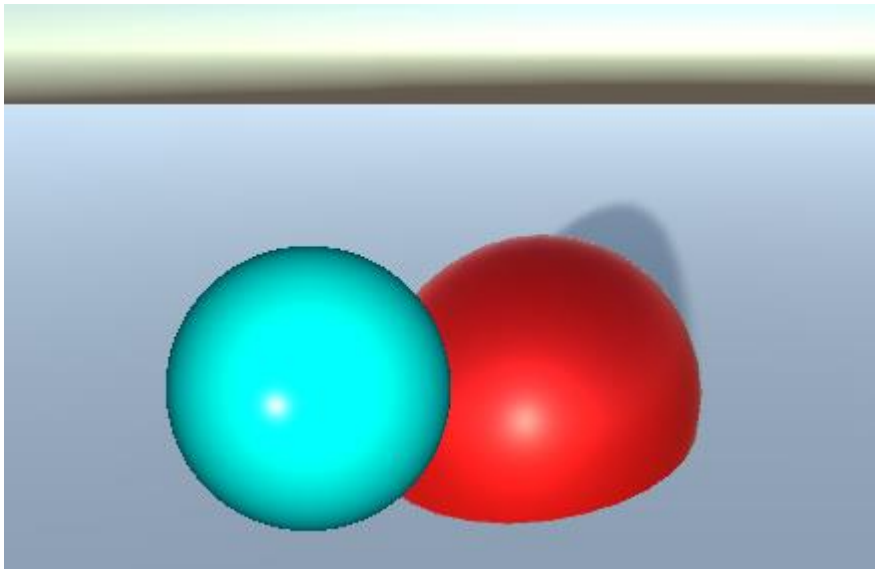


Figure 18 Scene capture of two spheres intersecting each other and also intersecting with a default Unity plane; The left sphere is a standard sphere mesh of Unity rendered with a standard Unity surface shader. The sphere intersects correctly, resulted by a correct depth buffer rendering. The right sphere uses the sphere impostor's technique previously described. As we see, the depth buffer values of the sphere impostor are completely wrong, thus the impostor not even intersects with the plane, but also it doesn't intersect with the Unity's default sphere mesh. The depth value of our impostor tell that will be always nearer than any other mesh.

If you don't specify, the output depth in your shader, then OpenGL will freely use `gl_FragCoord.z` as the depth output from the fragment shader. This value will be depth tested against the current depth value and, if the test passes, written to the depth buffer. However, we have the ability to write a depth value by ourselves.

Basically, we will need to go through the process OpenGL normally does in order to compute the depth. Fortunately, we just compute it in camera-space position while we were computing our ray tracing function. Therefore, we will just need to transform the position to clip space. When the perspective division happens, the value it's transformed to normalized device coordinates (NDC) space. An addition, when the depth range

function is applied, it forces the range [-1, 1] in the fragment shader to the range that the user provided with `glDepthRange`. Finally, we write the final depth to the built-in output variable `gl_FragDepth`.

Once we assign the depth range value, we can compile again our shader and check if the new depth value is correct. As shown in *Figure 19*, our impostor has a properly depth value and intersects perfectly with other meshes as well as it doesn't screen other objects that are in front of it.

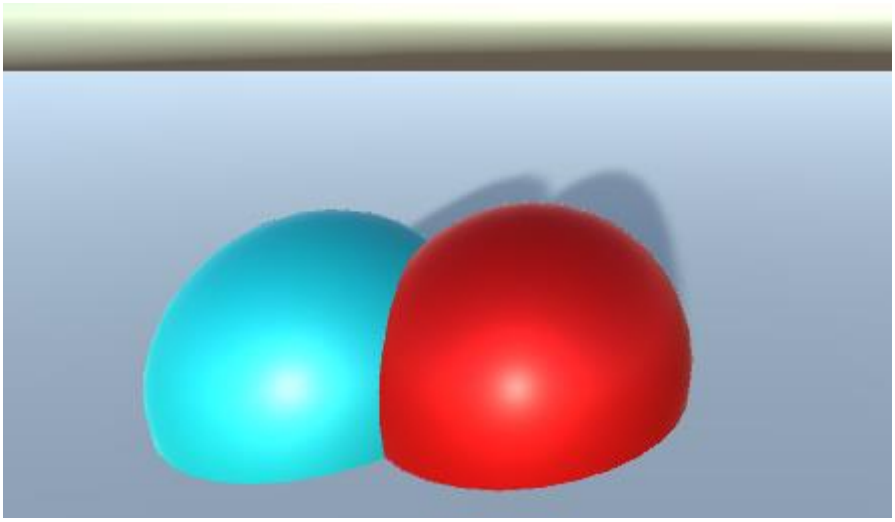


Figure 19 Scene capture of two spheres intersecting each other and also intersecting with a default Unity plane; The left sphere is a standard sphere mesh of Unity rendered with a standard Unity surface shader. The sphere intersects correctly, resulted by a correct depth buffer rendering. The right sphere uses the sphere impostor's technique with the fragment's depth range computation. as can be seen, the depth range value is computed

4.3. Other rendering techniques: GPU Instancing

4.3.1. Overview

As we mentioned before, rendering some geometrical forms can be quite expensive, especially if we try to render a proper sphere. In the last chapter we talked about the billboard impostors; they were one way to reduce the render time and draw calls of our molecule representation. Now, we will introduce another technique called *GPU instancing*. GPU Instancing draws multiple copies of the same Mesh at once, using a

small number of draw calls. This technique is done at runtime for visible objects. However, it has some restrictions, especially for the dynamic meshes.

In order to properly instantiate our molecular geometry meshes, we will apply some main rules and restrictions according to Unity's documentation [37][38]:

- Batching dynamic meshes has certain overhead per vertex, so batching is applied only to meshes containing fewer than 900 vertex attributes in total.
    - If your Shader is using Vertex Position, Normal and single UV, then you can batch up to 300 verts, while if your Shader is using Vertex Position, Normal, UV0, UV1 and Tangent, then only 180 verts.

- Meshes are not batched if they contain mirroring on the transform; this means we cannot it will not batch meshes with opposite transform values (for example mesh **A** with **+1** scale and mesh **B** with **–1** scale cannot be batched together).

- Using different Material instances causes meshes not to batch together, even if they are essentially the same. The exception is *shadow caster rendering*.

- Multi-pass Shaders break batching.
    - Almost all Unity Shaders support several Lights in forward rendering, effectively doing additional passes for them. The draw calls for "additional per-pixel lights" are not batched.

4.3.2. Proper Instantiation

In order to reduce as much as possible, the draw calls of our molecule mesh, we will have to execute some previous steps. By default, GPU instancing isn't enabled. The shader that we are going to design will have to meant to support it. Even then, instancing has to be explicitly enabled per material.

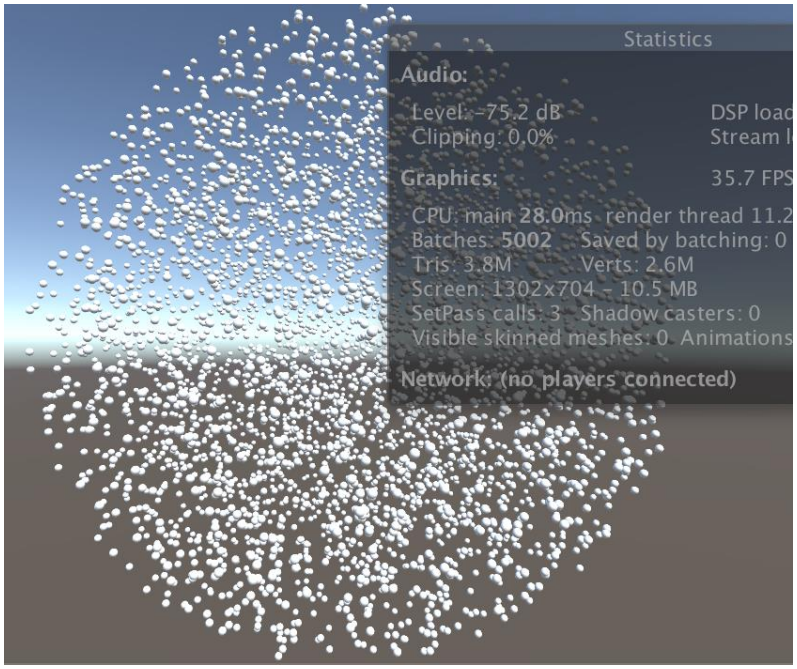


Figure 20 Scene capture with the camera focus centered to 5000 spheres. The sphere materials have the GPU instancing disabled. As we can see in the statistic, the batch call number is 5002, meaning it rendered just one sphere per time. Furthermore, we can observe than it made 3.8 million tris calls and 2.6 million vertices calls. This sphere rendering would be unsuitable four our molecular geometry representation purposes.

Unity's standard shaders have a material's toggle for this. The toggle can be added by invoking the `MaterialEditor.EnableInstancingField` method. The toggle will only be shown if the shader actually supports instancing. We can enable this support by adding the `#pragma multi_compile_instancing` directive to at least one pass of a shader [8].

At this moment, we are now sending the matrices of all spheres meshes to the GPU as an array. A single matrix consists of 16 floats, which are four bytes each. So that's 64 bytes per matrix. Each instance requires an object-to-world transformation matrix. However, we also need a world-to-object matrix to transform normal vectors. In the end, we end up with 128 bytes per instance. This leads to a maximum batch size of $\frac{64000}{128} = 5000$, which means it could render 5000 spheres in only 10 batches.

However, we are not telling to the shader which array index has to use. Without telling the shader which array index to use, it always uses the first one. Thus, we will only be able to see part of the batched spheres, the first sphere of each batch call (see *Figure 21*).

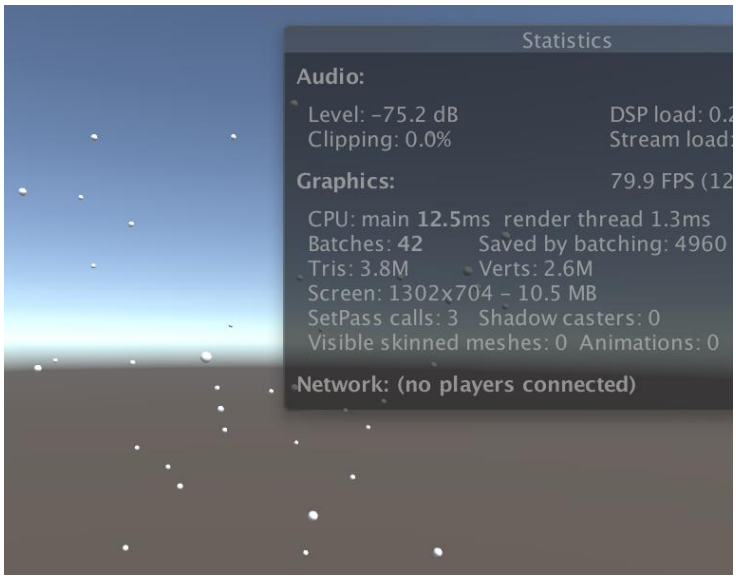


Figure 21 Scene capture with the camera focus centered to 5000 spheres. The sphere materials have the GPU instancing enabled. No array index is assigned. As we can see in the statistics table, the batch call number is 42, meaning it rendered 125 sphere per time. Furthermore, we can observe than it made 3.8 million tris calls and 2.6 million vertices calls. Despite we wanted 5000 spheres shown in scene, we can only see a few, concretely 42 spheres. Because we haven't assigned any index, the shader will only render the first mesh of each batch call.

The array index corresponding to an instance is known as its instance ID. The GPU passes it to the shader's vertex program via the vertex data. It is an unsigned integer named `instanceID`. We can simply use the `UNITY_VERTEX_INPUT_INSTANCE_ID` macro to include it in our VertexData structure. It is defined in *UnityInstancing*, which is included by Unity. It gives us the correct definition of the instance ID, or empty value when instancing isn't enabled [8]. We now have access to the instance ID in our vertex program, when instancing is enabled.

However, to make the macro work and get the array index, the instance's array index has to be globally available for all shader code. We have to manually set this up via the `UNITY_SETUP_INSTANCE_ID` macro, which must be done in the vertex program before any code that might potentially need it. Now, the shader can access the transformation matrices of all instances, so the spheres are rendered at their actual locations.

As we mentioned before, one limitation of batching is that they are limited to objects that have identical materials. This limitation becomes a problem when we desire variety

in the objects that we render. This is our case, since each molecule will have different color types, such by the molecule chain, the residue or the atom type.

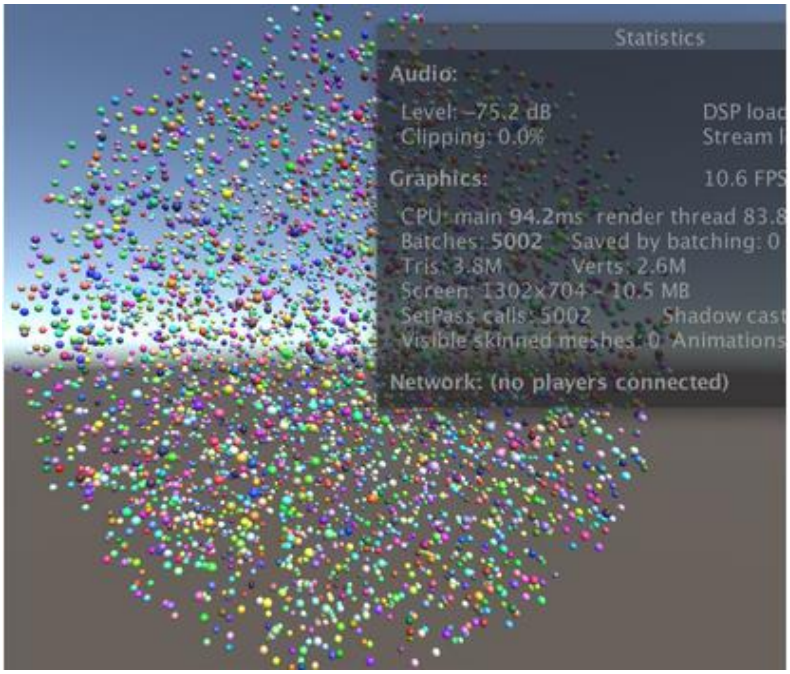


Figure 22 Scene capture with the camera focus centered to 5000 spheres. The sphere materials have the GPU instancing enabled. The array index is assigned. Each sphere will have assigned one color randomly. As we can see in the statistics table, the batch call number is 5002, meaning it rendered 1 sphere per time. Furthermore, we can observe than it made 3.8 million tris calls and 2.6 million vertices calls. Even though we have enabled batching for our material, it no longer works. as each now has its own material, the shader state has to be changed for each sphere as well.

Instead of creating a new material instance per sphere, we can use material property blocks. These are small objects which contain overrides for shader properties. Alternatively, instead of directly assigning the material's color, set the color of a property block and pass that to the sphere's renderer. This allows us to reuse one block to configure all of our instances.

Additionally, the GPU has to know about which property is overriding. When rendering instanced objects, Unity makes the transformation matrices available to the GPU by uploading arrays to its memory. Unity does the same for the properties stored in material property blocks. Like the transformation matrices, the color data will be uploaded to the GPU as an array when instancing is enabled. The `UNITY_DEFINE_INSTANCED_PROP` macro takes care of the correct declaration syntax for us.

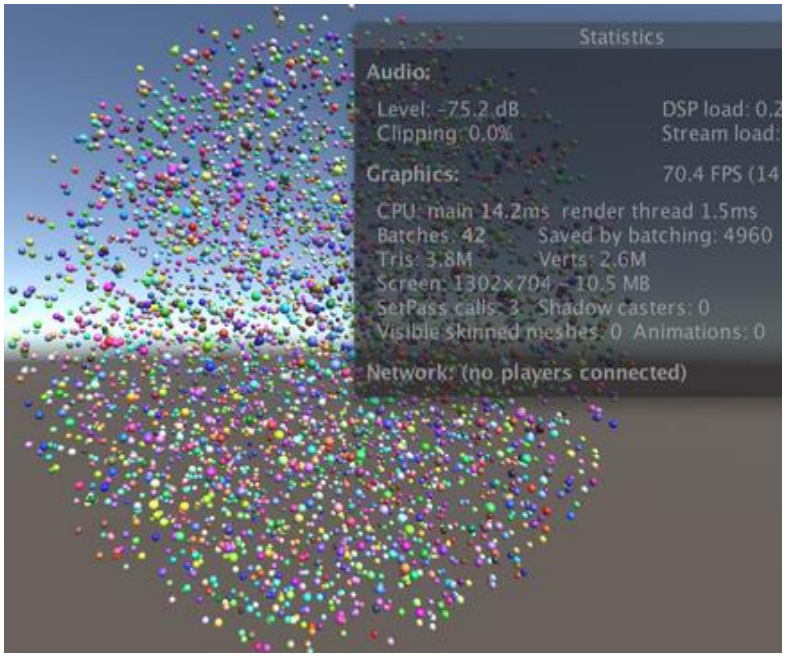


Figure 23 Scene capture with the camera focus centered to 5000 spheres. The sphere materials have the GPU instancing enabled. Each sphere will have assigned one color randomly. Unlike the previous scene of ***Figure 22****, material property blocks has been created and color data will be uploaded to the GPU. As we can see in the statistics table, the batch call number is 42, meaning it rendered 1 sphere per time. Material property block batching has been enabled. As we can see, our colored spheres are batched again.*

4.4. Techniques performance & comparison

4.4.1. Testing the rendering techniques

Once we have implemented our molecular geometry rendering techniques, will proceed to perform a series of rendering tests in order evaluate the performance of our previously defined techniques.

In order to execute the tests, we will create a new scene with an empty mesh prefab. This mesh will contain a script that will instantiate *N* number of molecular geometries meshes, each mesh with a random color. The instantiated meshes will not be further than *M* unity's scale units from the prefab center. *N* and *M* will be variable inputs introduced by the user.

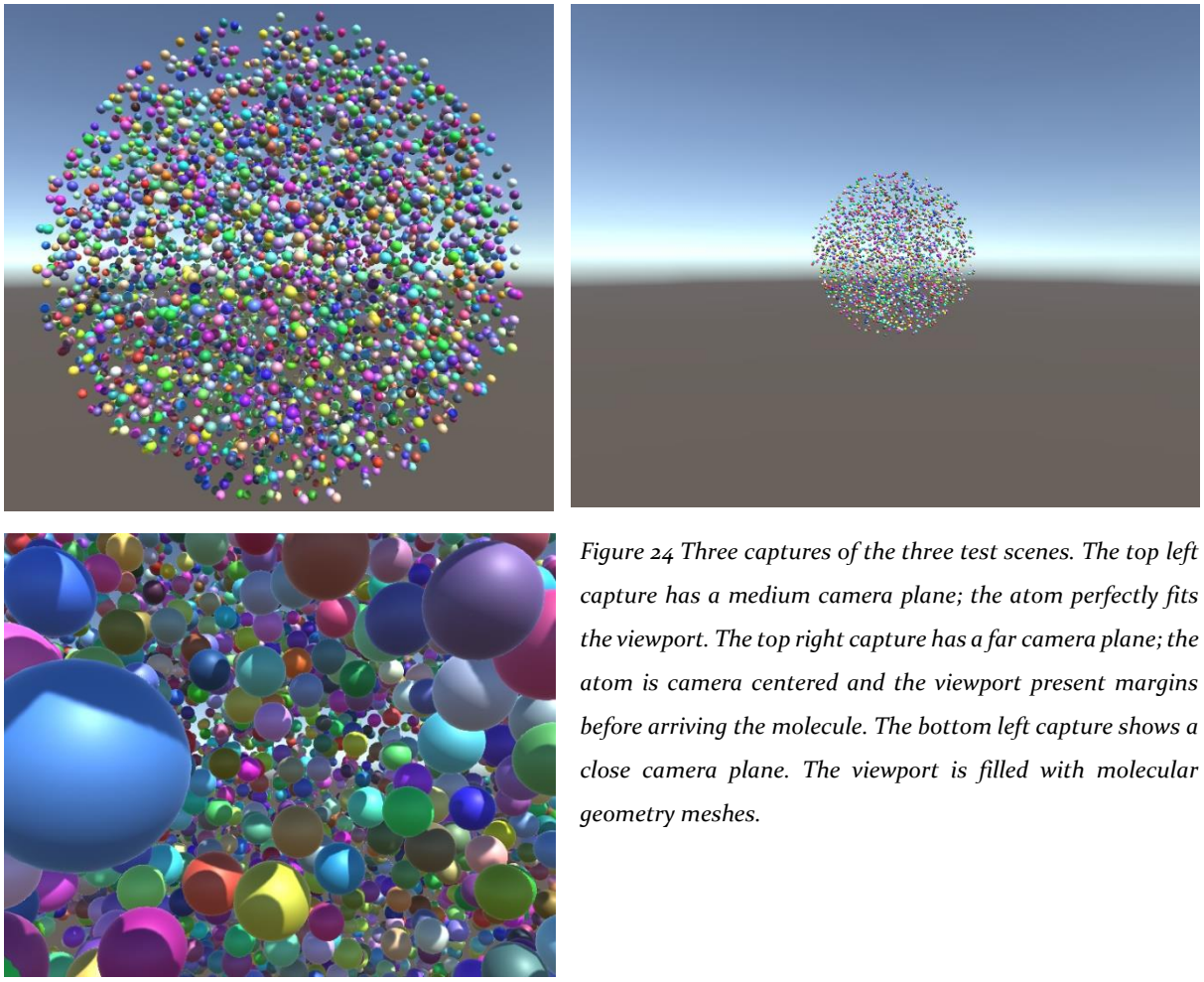


Figure 24 Three captures of the three test scenes. The top left capture has a medium camera plane; the atom perfectly fits the viewport. The top right capture has a far camera plane; the atom is camera centered and the viewport present margins before arriving the molecule. The bottom left capture shows a close camera plane. The viewport is filled with molecular geometry meshes.

We will execute three different frame-rate performance tests, each test with a different camera plane. The planes will be medium close and far. In all the three tests, the camera will be focusing at the center of the prefab. With these three differenced planes styles, we will be able to see if the bottleneck resides in the vertex or in the fragment shader.

We will test the UV-sphere, the billboard impostor sphere, the GPU instancing of UV-Spheres and the GPU instancing of the billboard impostor. We will instantiate 4 different sets of meshes, each with 100, 1000, 5000 and 10000 respectively. This way we will be able to compare them and see which obtains the best performance and which is the best candidate our molecular viewer.

The objective of this test is to evaluate the average frame rate (*fps)* that we obtain in each proposed scene. In order to get the average frame rate, we will capture the scene

frame rate every second, 100 times. Then, we will compute the obtained average frame rate value.

Therefore, the resulted values of each test will be the following ones:

4.4.1.1. Medium plane performance test

(fps)	100	1 000	5 000	10 000
UV-sphere	218.1	104.0	34.7	21.4
Impostor	225.5	145.3	60.1	43.8
UV-sphere + GPU Instancing	108.0	141.2	119.4	41.6
Impostor + GPU Instancing	109.7	142.6	121.9	55.5

Table 7 Frame rate table from the medium plane performance test

4.4.1.2. Far plane performance test

(fps)	100	1 000	5 000	10 000
UV-sphere	218.9	105.6	45.8	23.5

Impostor	228.	147.6	66.1	49.7
UV-sphere + GPU Instancing	110.3	146.0	130.1	53.2
Impostor + GPU Instancing	109.4	146.7	135.4	68.1

Table 8 Frame rate table from the far plane performance test

4.4.1.3. Close plane performance test

(fps)	100	1 000	5 000	10 000
UV-sphere	220.0	104.5	44.1	18.8
Impostor	224.3	142.9	68.4	41.1
UV-sphere + GPU Instancing	116.6	141.3	124.6	42.0
Impostor + GPU Instancing	108.1	141.8	120.1	50.5

Table 9 Frame rate table from the close plane performance test

4.4.2. Performance test conclusions

If we take a look at the obtained results, we can see that the billboard impostor gets a clearly better performance than the UV-sphere every aspect. However, despite the impostor gets stunning performance result in low number of instanced meshes, the GPU can obtain better performance than the billboard impostor in large instantiation quantities.

Nonetheless, the performance technique test that obtained the best result in high instantiation number was the combination of the billboard impostor with the GPU instancing rendering technique. Despite the performance in fewer instantiation number is worse than the expected, the molecule proteins from the .pdb don't tend to have such a lower number of atoms and tent to be higher values (closer to the 5000 to 10 000 units). Furthermore, the frame rate still enough to display it fluently and without compromising the system performance or the user experience.

Moreover, if we compare the obtained results of each camera plane, we can see than the far plane tests have a better performance than the closer plane tests, which have a worse performance. The difference can be insignificantly in the lower instantiation numbers, but, if we pay attention, we will able to see than this difference on the performance value increases while the number of instances increases. This differentiation increase between the distances of planes can give us a very important hint about whether it would be the bottleneck; in the vertex or in the fragment shader.

Since the lower frame rates are obtained in the closer camera plane, we can conclude than the bottleneck will be located at the fragment shader:

The closer camera plane will capture a scene where the full viewport will be full of instanced meshes and no landscape will be appreciated on the margins. This means than our viewport screen will be fully painted with our rendered meshes. If we compared with the far camera plane viewport, the far camera viewport will just draw our rendered scenes in the center of our viewport screen, and resulted meshes bounding box will not

even be a quarter of the viewport. Therefore, the closer plane will consume more of fragment shader than the farther plane, since it requires more mesh painting.

If we percept a downgrade of the framerate while we are filling the viewport with our instantiated meshes, it means than the bottleneck is situated in the fragment shader and not in the vertex shader.

5. Large molecular models design

5.1. Rendering molecular models on Unity Engine

5.1.1. Data model overview

In order to build a reusable and a future expandable project, we will create hierarchical data model that abstracts and formalizes all the information of a molecule and divides it in hierarchical classes. When we are building the data model, we will try to decompose each concept as much as possible as well as we encapsulate each one in order to achieve the maximum concurrence.

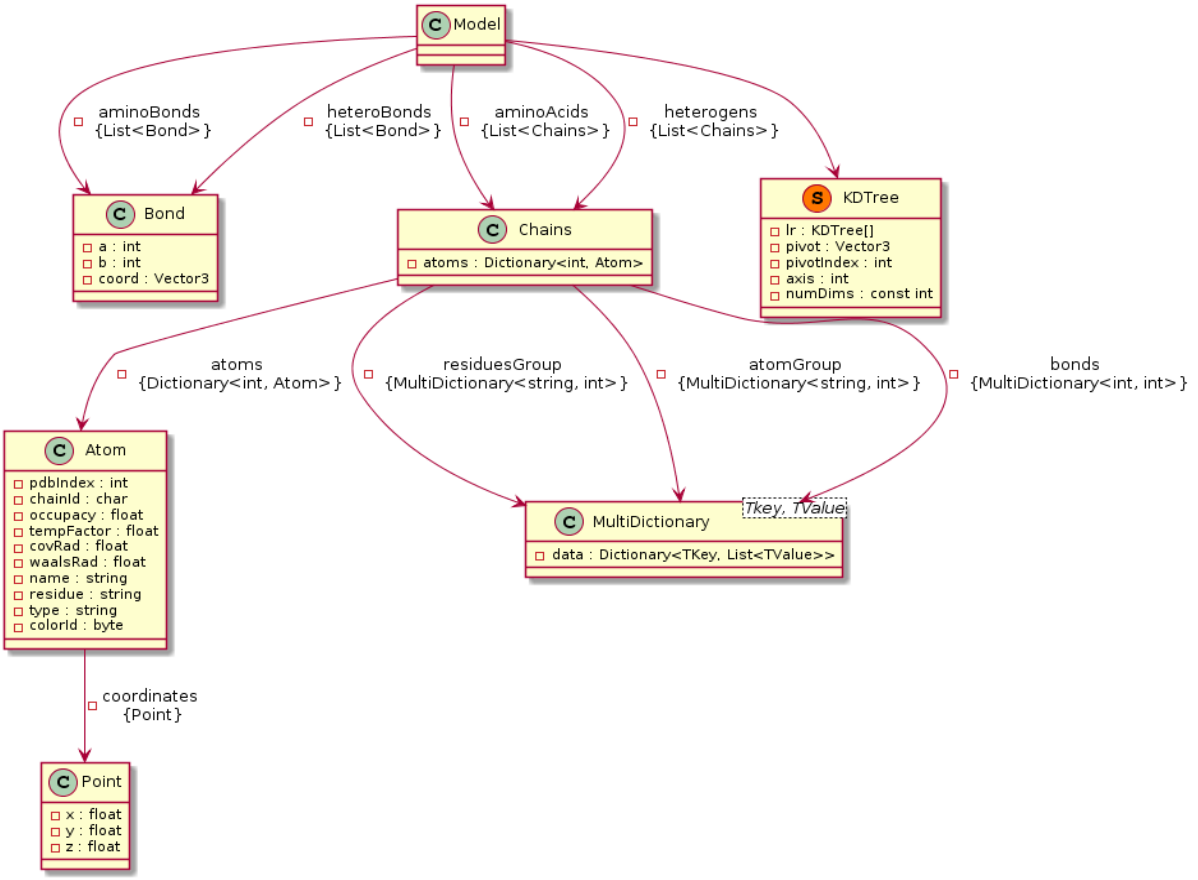


Figure 25 Diagram of the most relevant classes in the molecular viewer's data model. As we can observe, the data follows a hierarchical structure form as well as it isolates each class component. The component class diagram representation is sketched with plantUML open source. [aax]

Since explaining each model class and method would be overwhelming, the *Figure 25* shows an UML class diagram of the data model in which we will be able to observe each class attributes, as well as the dependency between components.

5.1.2. Building the PDB parser

The Protein Data Bank (PDB) is a dat49se for the three-dimensional structural data of large biological molecules, such as proteins and nucleic acids. Biologists and biochemists from around the world submit its research data, making it freely accessible for anyone. The Worldwide PDB (wwPDB) organization manages the PDB archive and ensures that the PDB data is freely and publicly available globally [46].

The file format initially used by the PDB was called the .PDB file format. This original format was restricted to 80 characters per line. Later, an XML version of this format called *PDBML*, was launched in 2005. However, we will strictly stick to the original .PDB file format.

The primary information stored in the PDB archive consists of coordinate files for biological molecules. These files list the atoms in each protein, and their 3D location in space. A typical PDB formatted file includes a large "header" section of text that summarizes the protein, citation information, and the details of the structure solution, followed by the sequence, composed by long list of the atoms and their coordinates. The archive also contains the experimental observations that are used to determine these atomic coordinates.

When we start exploring the structures in the .PDB archive, we will need to know a few things about the coordinate files. In a typical entry, we will find a diverse mixture of biological molecules, small molecules, ions, and water. In our parser, we will use the names and chain IDs to help sort these out. In structures determined from crystallography, atoms are annotated with temperature factors that describe their vibration and occupancies that show if they are seen in several conformations.

```
ATOM   2967  N  BHIS A 312      23.145   4.021   7.514  0.35 13.33           N
ANISOU 2967  N  BHIS A 312     1363   1162   2539   -575   1000      5       N
ATOM   2968  N   GLU A 313      26.355   3.548   6.018  0.50 15.22           N
ANISOU 2968  N   GLU A 313      986   3420   1376   -842    489  -1006       N
TER    2969      GLU A 313
HETATM 2970  PA  NDP A 318      22.607  -4.733  26.909  1.00  3.03           P
ANISOU 2970  PA  NDP A 318      333    382    439      4    -37     21       P
HETATM 2971  O1A NDP A 318      23.524  -3.682  26.406  1.00  3.64           O
ANISOU 2971  O1A NDP A 318      375    431    576    -29     -8     26       O
```

Figure 26 Typical .PDB file format fragment. The 6 first characters will determine the executed command type. The following line characters will have a specified column assignment depending of the command in question.

While we are exploring the PDB archive, we may run into several challenges. For example, many structures, particular those determined by crystallography, only include information about part of the functional biological assembly. Also, many PDB entries are missing portions of the molecule that were not observed in the experiment. These can include structures with missing loops, structures of individual domains, or subunits from a larger molecule. In addition, most of the crystallographic structure entries do not have information on hydrogen atoms [46].

A typical PDB format file will contain atomic coordinates for a diverse collection of proteins, small molecules, ions and water. Each atom is entered as a line of information that starts with a keyword: either ATOM or HETATM. By tradition, the ATOM keyword is used to identify proteins or nucleic acid atoms, and keyword HETATM is used to identify atoms in small molecules [46].

5.2. Molecular bonds computing

5.2.1. Rule-based algorithm for bond computing

Computing and assigning bond the atom bonds are a necessary and essential step for characterizing a chemical structure correctly in our molecular viewer. During the last decades, several methods have been developed to do compute it. All the methods have advantages. However, they all have limitations too. We have designed an automatic algorithm for assigning chemical connectivity and bond order. Now we will briefly discuss the main methodology that we will follow in order to compute the bond order of our molecule. We have divided in different steps, including the main rules among them. The *Figure 26* show a schematic representation of the steps that will take part in.

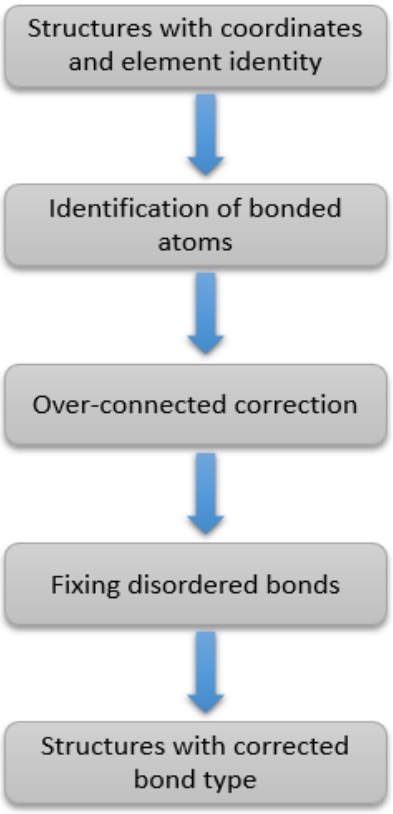


Figure 27 Flow chart of our methodology for the molecule bond order computing. Our step rules will be the key for a proper bond order computation.

○ Identification of bonded atoms

The first step we will make in our methodology will be detecting the connection of all possible atom bonds. This step will stick into a length distance equation and will not follow any other property or distinction, meaning it doesn't take into account the number of possible connections an atom type could have or any other rule or property.

$$\mathbf{0.8} < \boldsymbol{d_{ij}} < \boldsymbol{r_i} + \boldsymbol{r_j} + \mathbf{0.4}$$

Therefore, the equation shown on top describes if two atoms will be bonded or not. Being $\boldsymbol{r_i}$ and $\boldsymbol{r_j}$ the covalent radii of atoms $\boldsymbol{i}$ and $\boldsymbol{j}$ of a set of atoms $\boldsymbol{N}$, where $\{\boldsymbol{i}, \boldsymbol{j}\} \in \boldsymbol{N}$ and $\mathbf{1} \leq \boldsymbol{i} < \boldsymbol{j} \leq \boldsymbol{N}$. The distance between them is represented by the variable $\boldsymbol{d_{ij}}$.[39]

This process can be really expensive computationally talking, since checking for each atom if the distance between each of the remaining atoms asserts the equation or not is a NP-Complete problem.

Since this step process is undoubtedly the bottleneck of our algorithm, we will deep into the asymptotic cost of acquiring all the atom connections that follows our imposed equation rule. Later we will discuss an alternative technique that will drastically improve our asymptotic cost as well as will compare the algorithm results. Anyway, now we will continue describing the steps methodology in order to properly compute the bounds.

○ Over-connected correction

Once we have identified all the possible connections between atoms, we have to start discarding all the non-bonded atoms, and we will start with the over-connected atoms. Due to the intrinsic property of each element, each atom should have the maximum connections.

Therefore, the steps that will follow our algorithm will be the next one; given the atoms **C**(4), **N**(4), **P**(4) and **S**(4) (the value indicates its maximum connection number). The longest bond is removed and the atom is checked again until the number of the connections is no longer bigger than its maximum connection number.

Despite we could include more atom types to our methodology step (all the elements of the *CPK* table for example), we have selected the four types mentioned above because its tendency to bond with oxygen (**O**) and hydrogen (**H**) and also for its high number of possible connections. The other atom types will skip this rule and proceed to the following ones.

- Fixing disordered bonds

We already checked the atoms to make sure they are not over-connected. However, even after the application of the rules above, not all bonds connections are single bonds; some could be double or even triple bonds. This means that some atoms could be over-connected. Therefore, we will use the maximum valence to judge if this atom is over-connected. For atom given atom $\boldsymbol{A_i}$, the number of the connected bonds is calculated based on the following equation [39]:

$$\boldsymbol{iCon} = \sum_{\boldsymbol{k=0}}^{\boldsymbol{n}} \boldsymbol{O_{ik}}$$

Where if $\boldsymbol{iCon} > \boldsymbol{Valence}\ \boldsymbol{n}°$, the longest single bond is deleted, and the structure is checked again until $\boldsymbol{iCon}$ is no longer larger than the maximum valence. Here we just handle single bond, because for the bond with the bond order larger than $\mathbf{1}$, its bond length is reliable if this bond can be judged by the rules applied above.

5.2.2. K-d tree algorithm

As we mentioned in the previous chapter, the main bottleneck of our algorithm will be the determination of the possible connections of each atom. If $\boldsymbol{n}$ is the number of atoms of the molecule we would have to iterate $\boldsymbol{n}$ times $\boldsymbol{n-1}$. This means the asymptotic computational complexity of our algorithm will have an upper bound of $\mathbf{O}(\boldsymbol{n^2})$. This problem is critical for algorithm implementation, since the execution time will increase quadratically depending of the number of the atoms the molecule has. However, we know that finding the nearest neighbor problem is **NP-Complete** and every NP-Complete problem would also have a solution in polynomial time.

In computer science, a **k-d tree** (for *k-dimensional* tree) is a space-partitioning data structure algorithm for organizing points in a relative k-dimensional space [40]. In particular, is a binary tree in which every leaf node is a k-dimensional point, in our case,

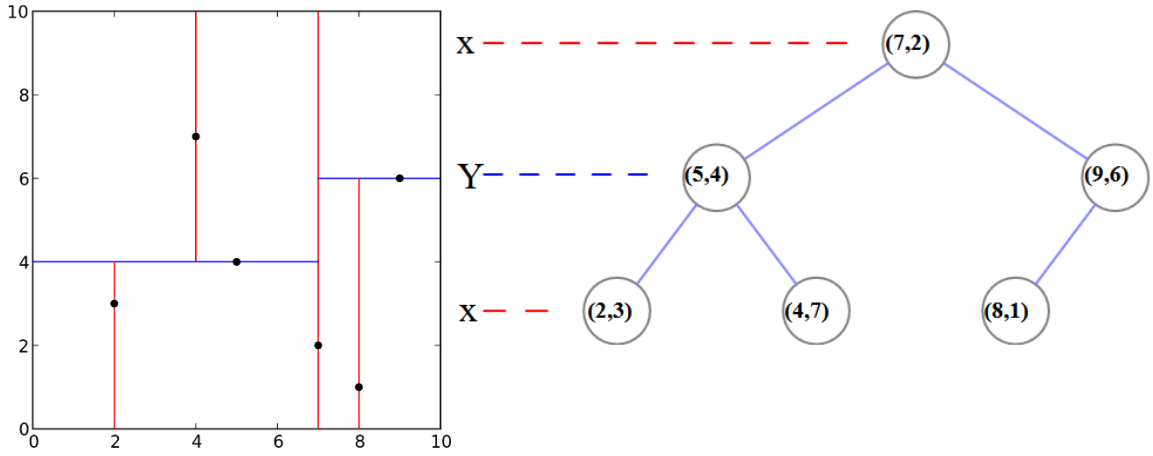


*Figure 28 Graphical decomposition of a **2**-dimensional tree. The "**x**" axis was the chosen one to start building the resulting tree. The root point is chosen by the median of medians algorithm.*

a 3-dimensional point. Every non-leaf node can implicitly generate a splitting hyperplane that divides the space into two parts, known as *half-spaces*. Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree.

The hyperplane direction is chosen in the following way: every node in the tree is associated with one of the k dimensions, with the hyperplane perpendicular to that dimension's axis. So, for example, if we choose to split the "x" axis, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with larger "x" value will be in the right subtree. In such a case, the hyperplane would be set by the x-value of the point, and its normal would be the unit x-axis [41].

In our algorithm implementation, each coordinate point will be equal to space-relative coordinate of each of the molecules. We have decided to start building our tree with the "x" axis and choosing the root element with the *median of medians* algorithm to select the median axis value at each level of the nascent tree.

Once we have built our 3-d tree, we can start finding which elements will have a possible connection. One of the main k-d tree operations is the *nearest neighbor search*. The nearest neighbor search (**NN**) algorithm aims to find the point in the tree that is nearest to a given input point. This search can be done efficiently by using the tree properties to quickly eliminate large portions of the search space.

As we are using the median of median algorithm with an asymptotic cost of $\mathbf{O}(\boldsymbol{n})$. The total average cost of building our tree will be $\mathbf{O}(\boldsymbol{n\ log\ n})$.[42] Furthermore, making a binary tree search, finding the nearest point will have an asymptotic cost of $\mathbf{O}(\boldsymbol{log\ n})$.

Additionally, the algorithm can be extended in several ways by simple modifications that, in our case, will improve our computational cost. We can extend it to provide the k nearest neighbors to a point by maintaining k current bests instead of just one. A branch is only eliminated when k points have been found and the branch cannot have points closer than any of the k current bests.

This search implementation can improve our methodology steps and overall performance; Instead of checking all the possible atom connections and then deleting the over-connected ones, we could fix the number of connections for each molecule to four, which will be the maximum number of bounds. Therefore, can use the 4-neareast neighbor search to get the 4 closer connections candidates of each atom. With this new implementation we are able skip the over-connected correction step and go through directly to fix the disordered bonds as well as we are reducing the total asymptotic cost from $\mathbf{O}(\boldsymbol{n^2})$ to $\mathbf{O}(\boldsymbol{n\ log\ n})$.

5.2.3. Runtime performance comparison

In order to observe the algorithm performance and visualize the improvements than the k-d tree brings to our bond computing algorithm, we will implement a simple performance test between the k-d tree algorithm and a basic brute force algorithm.

In order to perform this test, we will build a simple script that given the number of elements *n* as input parameter, it will build and array set if size *n* of random 3-d points, and will find the nearest neighbor of all the elements in the set. This process will be repeated *m* times, each time with a different set *n*. the final results will be a regression of each of the obtained execution's run time values.

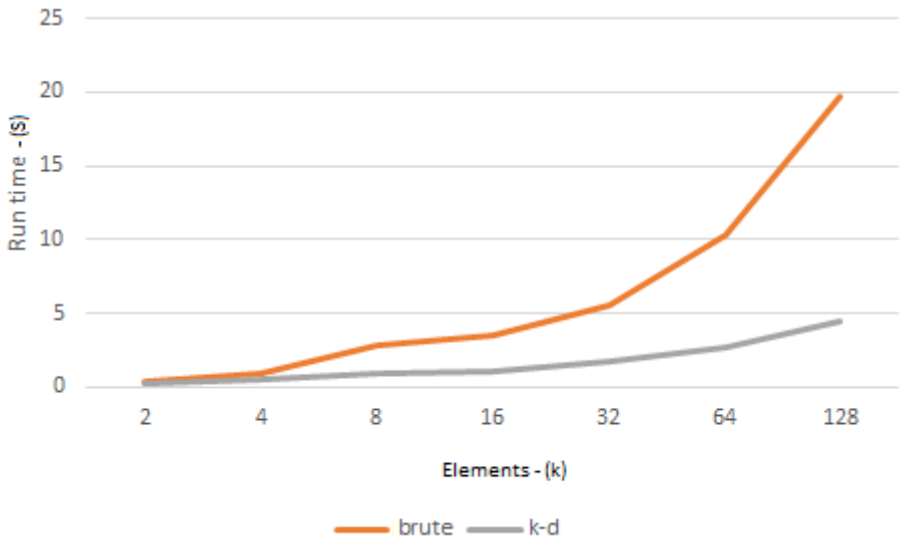


Figure 29 Performance results of the K-d Tree algorithm (grey) versus the Brute force algorithm (orange). The abscises axis shows the number of elements to perform, represented in thousands (kilos). The ordinates axis shows the run time execution of the algorithms in seconds.

If we observe the obtained graph representation of the *Figure 28,* we can see that the results where as we expected. The values have skyrocketed in run time execution when we reached the 64 thousand number of elements, bringing and exponential curve tot the representation. The k-d tree algorithm keeps bringing a stable performance and increments polynomial while we increase the elements number.

However, we have to take into account that the run time execution of the k-d tree not only includes the neighbor search, but also includes the tree structure creation. This will have a cost of $\boldsymbol{O(n\ log\ n)}$, the same of the overall cost of every neighbor search. This means the run time execution time of this tests will be a little bit higher since that in our project, the tree would only be built once.

Therefore, we can conclude than the k-d tree is a really good optimization for our algorithm implementation; Not only improves considerably the bond's computation time, but it also can be reusable, since we are building a tree structure than have more utilities rather than bond computing, but also can be used for other future methods and implementations such as group selection or a possible atom scrolling selection etc.

5.3. Silhouette rendering

5.3.1. Overview

One of the main techniques that will help the user to interact with the atoms of the molecule will be the silhouette rendering. Here, we will make the use of vertex transformations, which will move each vertex along its surface normal vector to create a larger outline of an object. While this vertex transformation is quite simple, it will be creating a reasonable rendering of the outline. It also requires use of the stencil buffer, which is also discussed.

Therefore, we will try to render a world space silhouette shader, which is such efficient that will barely impact our frame-rate. It will create an outline by moving vertices along their normal vector, as well as it will avoid occlusions of the outlined object by its outline. We will also be able to select the silhouette thickness as well as its color. Finally, we will talk about the visual results and will performance a stress test to analyze the impact of the silhouette rendering in our molecular visualizer.

5.3.2. World Space Silhouette rendering

As we mentioned before, one method to create a silhouette around an object is to enlarge the object by moving its vertices along their surface normal vectors. Given the position and the normal vector in object coordinates, this we could implement it easily in a vertex shader; The vertex shader will multiply the surface normal vector with the user-specified uniform of the *thickness* (since it's a uniform, it will allow an adjustable thickness of the outline) and will add it to the position of the vertex.

All these vertex steps will be computed in world space coordinates. One important thing to mention is that this vertex transformation works better with smooth surfaces. To give to the outline the color we will simple return the user-specified uniform of the *color* in the fragment shader. Such as the silhouette thickness we will be able to customize this color value.

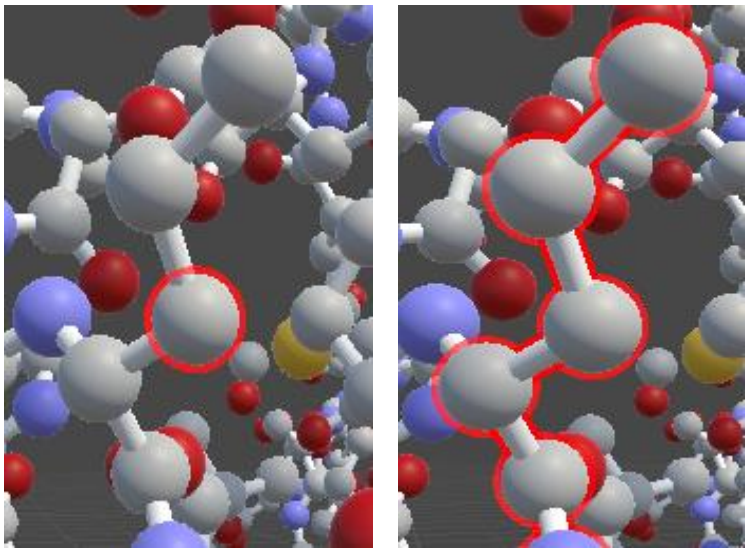


Figure 30 Two captures of the world space silhouette in a ball&sticks molecular representation. The right capture shows a single silhouette rendered in front of the other geometrical meshes. The left capture shows multiple silhouettes rendered behind the geometrical meshes

The more challenging part of rendering this kind of outline is to avoid occlusions of the outlined object, which are projected by the outline itself: since we have enlarged the object to create the outline, the larger outline will usually occlude the object that we want to outline. On the other hand, the outline should occlude other objects in the background and it should be occluded by objects in the foreground. Therefore, it should behave like any other opaque object, except when it is in front of the outlined object.

If we assume that we first we will render a regular version of the object that we want to outline, and after we will proceed to render the outline, we can state the problem with a couple of steps; the outline should only be rasterized for pixels that are not covered by the outlined object. This way, our problem can be solved with the use of the *stencil test*.

The stencil test is a per-sample operation performed after the fragment shader. The fragment's stencil value is tested against the value in the current stencil buffer; if the test fails, the fragment is culled [43]. The stencil test is used to limit the rasterization to certain parts of the framebuffer; in our case to the parts of the framebuffer that are not covered by the object that we want to outline.

Therefore, our strategy will be first marking all pixels that are covered by the object that we want to outline in the stencil buffer. After that, when rendering the outline, we can use a stencil test to rasterize the outline only in pixels that haven't been marked.

To mark all pixels that are covered by an object with a value of 1 in the stencil buffer, we can use the *ShaderLab* syntax provided by Unity in a Pass block before the GLSL implementation starts [43].

5.3.3. Silhouette performance test

As we did in *chapter 4.4*, we will execute a series of performance tests in order to evaluate the performance of our silhouette rendering technique. We will use the same test scene and the same manners we did in the last performance test. However, we will just execute one performance test; the center plane one.

We will not perform the other planes ones since we have determined than the bottleneck was in our fragment shader, and this technique is basically performed in the fragment shader, so there is no chance the bottleneck is moved to the vertex shader.

Therefore, the obtained table is the following one:

(fps)	100	1 000	5 000	10 000
No Silhouette	109.7	142.6	121.9	55.5
With Silhouette	107.4	101.0	18.2	8.6

Table 10 Frame rate table from the world space silhouette performance test

If we observe the resulted performance results, we can see than the than the frame rate barely decreases when we instantiate a low number of meshes. However, the framerate drastically decreases as we increase the instantiation number.

6. Interaction with Molecular Models

6.1. Introduction

6.1.1. Introduction to Unity framework and tools

Unity is a cross-platform game engine developed by Unity Technologies that was released in June 2005. The engine is mainly used to create videogames as well as simulations for many platforms, but the engine utilities can be infinite.

Unity brings us the ability to create three-dimensional and two-dimensional scenes, as well as a primary scripting API in C#, including Direct3D, OpenGL, OpenGL ES or WebGL. Unity previously supported other languages such as JavaScript, Boo, but both deprecated in order to prior C# [44].

Unity encompass the Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR) into the umbrella term of **XR** [44]. Despite we are exclusively using the VR tools, most of the features that Unity offers to us will be referred with the XR term. However, in order to not cause any confusion, we will just refer it as VR term.

Unity provides built-in support for a number of virtual reality devices. Its device availability is on a per-platform basis, meaning that not every device is available for every platform. Multiple devices are available on certain platforms, and you can choose which VR devices your app will support from the available list. At runtime, only one device can be active at any given time. However, it is possible to switch between devices if you have chosen to support multiple devices in your application.

6.1.1. GUI introduction

A graphical user interface (**GUI**), is a type of user interface which allows users to interact with objects and devices through graphical icons and visual indicators, instead of text-

based user interfaces, such as typed command labels or text navigation. GUIs were introduced in counterpart of the perceived hard steep learning curve of command-line interfaces (**CLI**s), which require commands to be typed [49].

Because a GUI is much more visually intuitive, we decided than the interaction with the molecular models will be made with a GUI rather than a typical command line interface. A GUI offers a lot of access to properties and features, in addition, a GUI its more user friendly than a command line, especially for new or novice users, which makes a GUI being used by more users and increasing. Moreover, a GUI has more colors and is more visually appealing, leading to a potential reduction in visual strain.

However, each GUI has a different design and structure when it comes to perform different tasks. Even different iterations of the same GUI, can have many different functions and changes, making it ambiguous sometimes. Furthermore, a GUI requires more system resources because of the elements that require loading, such as icons and fonts [50].

Most of the GUIs are rendered in camera space however, we will take advantage of the VR immersion to design a world space GUI which will make the interaction more reciprocal with the molecule and is going to create a more sensation of reality and scene awareness, which we cannot obtain with conventional GUIs.

6.1.2. Establishing a Three-tier Architecture

In order to define an efficient GUI, we will need to have the maximum level of concurrence and abstraction from the data model proposed in *chapter 5.1.2*. Therefore, we will take into practice the *Three-tier architecture* design pattern. A Three-tier architecture is organized into three major parts; the presentation tier (GUI), the logic or control tier, and the data tier. It follows a client-server structure and all the user, as well as each tier, is developed and maintained as independent modules.

Separating the application graphical user interface and the molecular data model will mean a better load balancing as well as it will make easier to modify or replace any tier without affecting the other ones. This way, any change performed by the GUI will be processed in the logic tier, and therefore forwards to the data model or vice versa. For example, if we want to change the data model of the molecule and add a new hypothetical parameter, it will not affect the control tier and neither the presentation tier

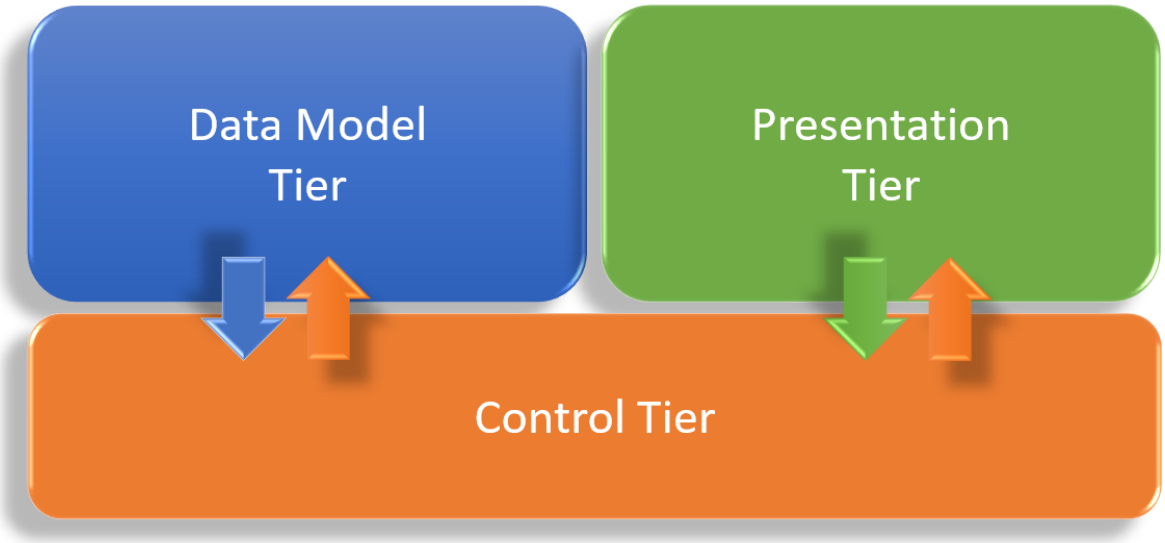


Figure 31 Three-tier Architecture diagram. As we observe, the data model and the presentation model are not communicated between each other. The control tier is the intermediary between the two other tiers.

Apart from the previously defined advantages, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently in response to future changes and upgrade implementations. This will allow us to implement future methods by modules and our application will still be functional and any future expansion will not be overwhelming or will require to reformulate the full data structure.

6.1.3. Multiple SDK supporting

Nowadays, there are many VR headsets available on the market. However, each headset uses a different SDK. When we are developing our molecular viewer application, we will try to reach the maximum number of developers and researchers. Therefore, we cannot assume that they are going to use the HTC Vive headset as we do. Our application will need to support multiple SDKs. In order to support multiple SDKs, we will make use of the *VRTK* toolkit [51].

The main feature of VRTK toolkit is the SDK abstraction layer feature; if we use the VRTK input components for our left and right controller's mechanics of our application, then it will just work on any supported SDK. Therefore, our application will work on SteamVR, Oculus or even PSVR without the need of any extra coding. [52]

Furthermore, VRTK offers a VR Simulator that works with the mouse and the keyboard controllers. Furthermore, the simulator provides a device simulation support of any third-parties SDKs. This means a user could test his project methods and implementations without the need of the VR headset.

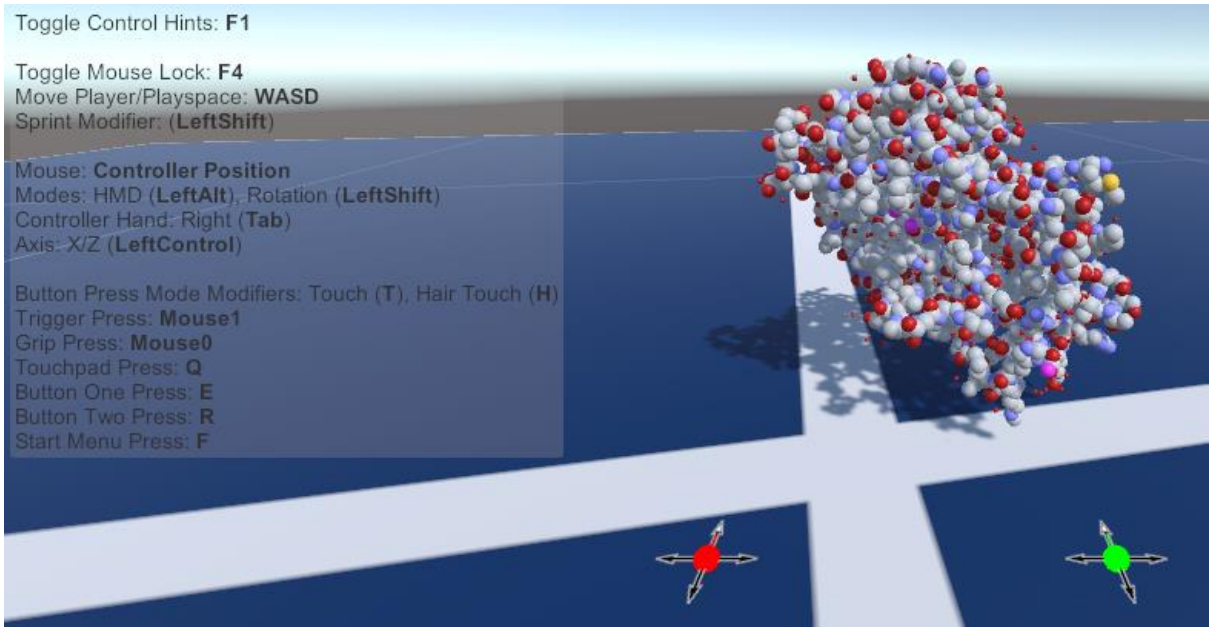


Figure 32 Scene capture of the VR Simulator provided by VRTK tools.

6.2. Defining our GUI

6.2.1. First GUI contact

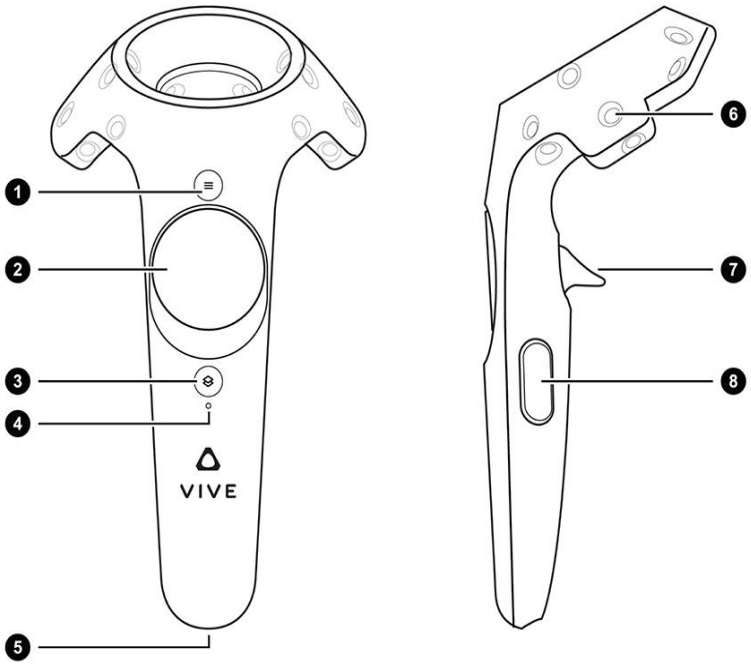
As we know, the VR give us a deeper sensation and awareness of the scene than a conventional screen can offer to us. However, it can also be more ambiguous and some controls and interactions can be confusing. When we are designing a virtual reality's GUI, the have to make sure than the user is connoisseur of the provided GUI indicators as well as it knows how to use them.

Therefore, when the user puts on the VR headset and initialize the application for the first time, he must know which the main interaction tools (haptic controllers) are and where positioned relative to the scene space are. The way we will make the user aware of his inputs will be with a graphical visual indicator, which is a faithful representation of our HTC Vive haptic controllers.

Additionally, those graphic controllers indicate which kind of input action the user is making; the graphical indicators will include from the track pad touch and press position to the buttons press & release events to the trigger pressed intensity.

Despite we provided to the user a graphical fact of the inputs and events that will take place with the haptic controllers, we still need to provide more information. The quantity of actions than the viewer provides are substantial and the learning curve may be harder than a conventional screen GUI.

Thus, we will have to show in a certain way to the user, which are the actions than each input handles. This way the user will perfectly know all the event actions and we will avoid some possible action unawareness or misunderstood. The way we will do it is by building a set of custom controller tooltips prefabs, whose will constantly indicate which action or event is doing each input. Those prefabs tooltips will not be interactable, but will show all possible methods and possibilities the haptic controllers will provide.



1 Menu Button
2 Trackpad
3 System button
4 Status light
5 USB port
6 Tracking sensor
7 Trigger
8 Grip button

Figure 33 HTC Vive controller inputs mapping representation. The GUI graphic controllers will be a loyal representation of the original controllers.

A prefab tooltip will consist of the following elements:

- A panel canvas: This canvas will contain a very brief description of the input event that's taking part. It will be rendered in world space and will have a transform angle always parallel to forward vector of the haptic controller in question. Furthermore, the canvas will have a reverse panel which have exactly the same information of the front panel. This way even if we are looking at the canvas from the front or from the back of the controller, he will always provide the event information. The canvas is set to a custom desired position, always relative to its parent, in this case, the haptic controller, and they will not occlude between each other. Moreover, in order to make it clear and visible, we will render the canvas in a black panel and with white text font.

- A segment: The main objective of the segment is to indicate which controller input the canvas is referring to. The origin of the segment will always be on the center of the canvas, while the destination of the

segment will be the center of the input in question. In addition, we will always make sure than the canvas is always in front of the segment, in camera space coordinates. This segment will be black as well.

Despite the notable utility of the tooltips, the fact that they are constantly floating around the haptic controllers could make it unpleasant and uncomfortable, thus it can obstruct our vision or even collide with the other scene object. In order to make it more efficient and satisfying we will add a final conditional; the prefabs tooltips of each haptic controller will only be shown if and only if the user's camera forward view vector is focusing the pertinent haptic controller, if not they will be hidden.

This way if the user is taking any event or action than strictly not requires to observe the controller, they will not bother or disturb. Furthermore, we will have the option to hide the canvas permanently with a trigger, located on the settings menu that we will discuss later.

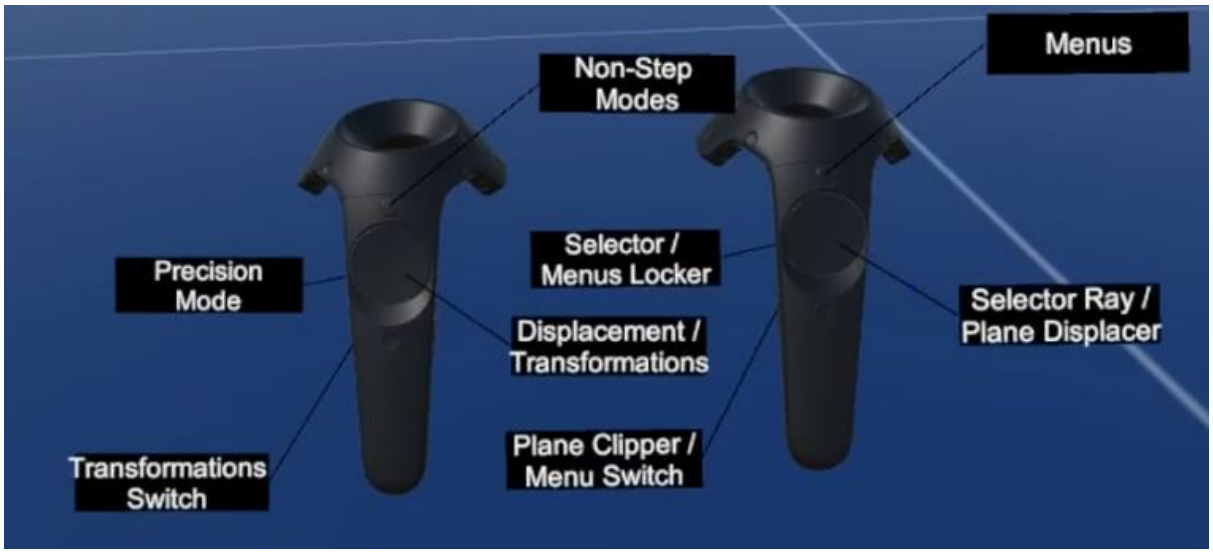


Figure 34 Scene capture of the controler's prefab tooltips

6.2.2. GUI events description

Once we are aware of each action for each controller input, we can proceed to discuss the GUI transformations and interactions it can take place.

For a better ease of use and a more comfortable molecule interaction we have split the events in two main parts. Therefore, if we take a look at the descriptions provided by each tooltip, we can observe than the left haptic controller will be responsible of the proper transformations of the molecule and the user, while the right haptic controller will be responsible of the proper molecule interactions and modifications. This way, while we are making the required transformations to approach to the desired section of the molecule with the left controller, we can interact with the right and no obstacles or misunderstood will take part.

Now, we will proceed to describe and explain each of the input events, starting from the left controller and continuing with the right one:

6.2.2.1. Left Controller event description

- Non-Step Movement -> *Trackpad Push*

The trackpad push will activate the non-step movement mode:

One of the most innovations of the virtual reality is the introduction of the step movement. With step movement, the user is able to freely move thought its located room and its displacement will be detected and, consequently reflected to the scene application.

However, the room space is limited and the application scene could be bigger than the actual room. This is one of the main drawbacks of the actual virtual reality, since it totally breaks the immersion and realism that VR offers to us. This way, the non-step

movement was introduced and keeps being one of the most investigated facts of the actual virtual reality development.

The way we will handle the non-step movement event will be through a ray traced along the forward vector of the left controller. The new user position will be the coordinates of the ground mesh ray collision. If we intersect with another mesh, the non-step movement is forbidden.

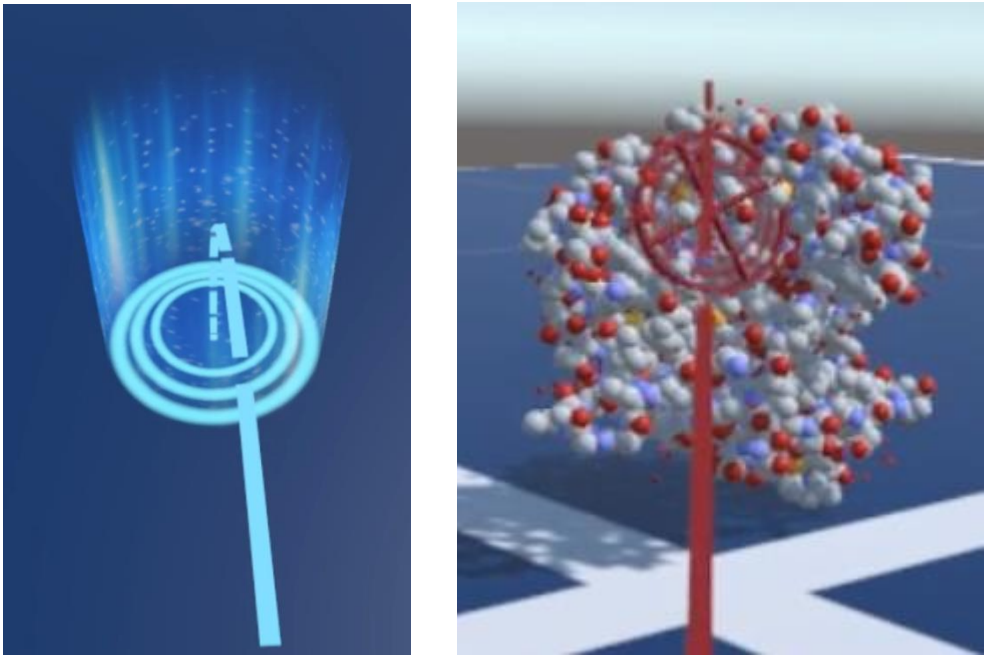


Figure 35 Scene captures of the non-step movement GUI prefabs. The left capture shows the visual indicator when the non-step movement is allowed. The right capture shows the visual indicator when the non-step movement is forbidden.

Therefore, it is really important to implement a proper non-step movement. It has to be precise and the visual indicators has to be clear and properly indicate how it will procedure. So that, we will design a custom set of GUIs indicators:

- Destination platform: This graphical indicator is composed by a circular platform that is projected perpendicularly to the first collided mesh by the ray trace vector. When the non-step movement is allowed, the platform will be rendered in blue color and will emit a blue glowing light. When the non-step displacement is

forbidden, it we be rendered in red color and it will have a red cross on the center of the platform.

- Bezier curve: The way we will indicate the displaced path than we will through a lineal Bezier curve. This curve goes from the haptic controller to the center of the destination platform. The curve will be represented with a dotted line, since it will obstruct lesser visual camp than a straight line. As the destination platform, when the non-step movement is allowed, the curve will be rendered in blue color whereas when the displacement is forbidden, the curve will be rendered in red.

- Teleport/Locomotion Modes -> *Menu Button*

The menu button will act as a trigger between blink teleport and dash locomotion mode:

- The first mode is the blink teleport mode. This is the default teleport mode. In blink mode, when you move you fade out the current location to black and then fade in at your new spot.

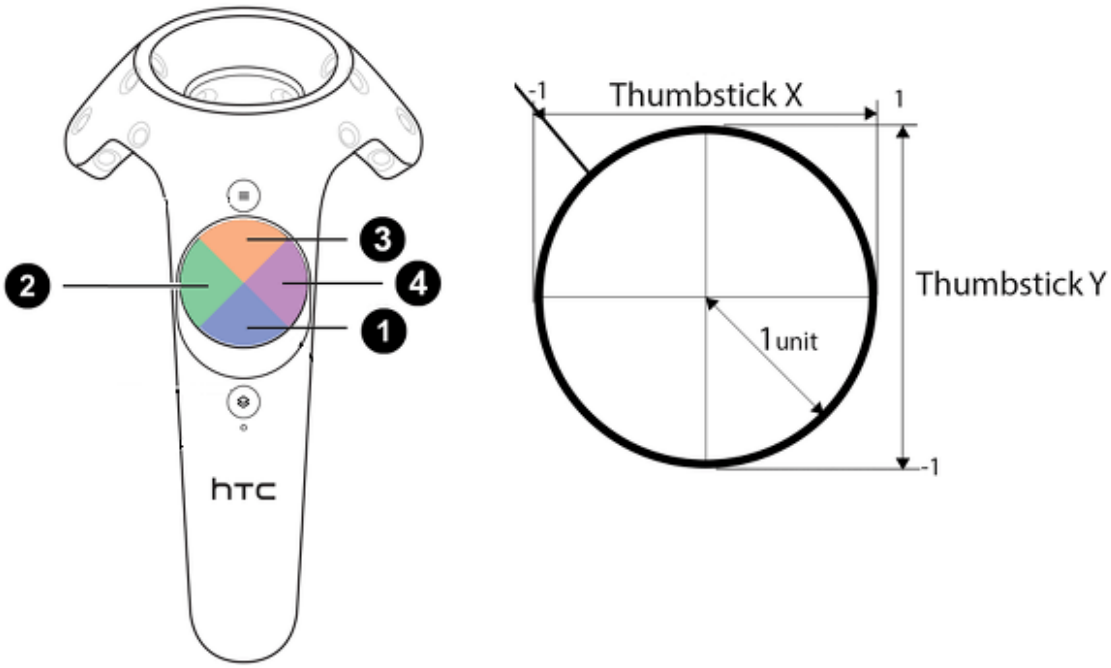
- The second mode is the dash locomotion mode. In dash mode, when you move you dash step forward to the new spot.

Providing different ways to combat motion sickness is really important for designing a thankful application. With this input, we provide two different ways to interact with non-step movement, which is one the dizziest and hardest problems of the virtual reality. Both methods are valid; the locomotion movement provides a more immersive and realistic non-step movement, while the teleport is more softly and can be more pleasant if you use the application for many hours.

- Molecule Transformations -> *Trackpad Touch*

The trackpad touch will manage the position and rotation transformations of the molecule:

The HTC Vive trackpad tool not only is sensitive to the user's input touches but also provides a two-dimensional position coordinates of the user's fingers relative to the trackpad. Each coordinate goes from -1 to 1. We will take advantage of The X and Y axis to provide many different kinds of molecule transformation that will describe in the molecule transformations switch mode section point.



1	Trackpad Down	3	Trackpad Right
2	Trackpad Left	4	Trackpad Top

Figure 36 HTC Vive trackpad sensor inputs mapping representation. It will present two thumbstick unitary coordinates

- Molecule Transformations Switch Mode -> *Grip Button*

The grip button will trigger between the rotation & altitude and the free translation mode:

- The first mode is the rotation & altitude mode. This is the default mode and is composed by two actions transformations, one action for each axis:

  - The first action, controlled with the X axis, is the rotation. The molecule will rotate to its left relative to the object coordinate's Y axis if the value is negative. Aversely, it will rotation to its right relative to the object coordinate's Y axis if the value is positive. The amount of value tracked by the trackpad touch X axis will affect the quantity of rotation degrees; if the value of the X axis is closer to the -1 or 1 values (left & right sides respectively) it will rotate more than a value close to the 0 value (center).

  - The second action, controlled by the Y axis is the altitude mode. This mode will transform the position of the molecule relative to the vector up (Y axis) in object coordinates. This way we can control the altitude of the molecule. The amount of value tracked by the trackpad touch Y axis will affect the quantity of rotation degrees; if the value of the Y axis is closer to the -1 or 1 values (left & right sides respectively) it will rotate more than a value close to the 0 value (center).

    This method is designed because the molecules proteins have a wide range of forms and, sometimes, they tend to be really tall, even more than the user. Hence, we can allocate the molecule where we desire and see the full parts and molecule compositions with a full variety of perspectives.

Furthermore, we will be able to observe the tallest parts of the molecule that could be too much higher to see by the user.

- The second mode is the free translation mode. In this mode we can freely displace the molecule around the scene. This mode will displace the molecule its X and Z coordinates and will ignore the Y coordinate since we don't to affect the molecule's altitude.

  The Y axis will translate the molecule around the vector forward of the haptic controller. If the value is positive the molecule will move forward. However, if the value is negative the molecule will move backwards.

  The X axis will translate the molecule around the vector cross of the haptic controller. If the value is positive it will move to the right. Aversely, if the value is negative it will move to the right.

- Precision Transformations Mode -> *Trigger Button*

The trigger press will activate the precision mode for molecule transformations:

The precision mode is no more than a functionality used along the touchpad touch, which will divide the transformation's input value to the half. This way we can obtain more precision in our molecule transformations.

6.2.2.2. Right Controller event description

- Options Menu / Stored Selection -> *Menu Button*

The menu button will activate the options menu and the stored selection modes:

- The options menu will perform the main interactions and modifications of the molecule's properties, as well as will be responsible of the application's settings. The menu canvas will be composed by two main panels; The top and the bottom panels:

    - The top panel will be composed by three horizontal layered buttons. Those buttons will be the load, properties and settings buttons. The button's click will alternate the bottom panel between the load, properties and settings panel menus.

    - The bottom panel will be composed by the load, properties and settings menus:

        - Load Menu: The load panel menu will be responsible of reading all the filenames of the .PDB folder and then, load them to a button scroll list. It will be composed by the next UI elements:

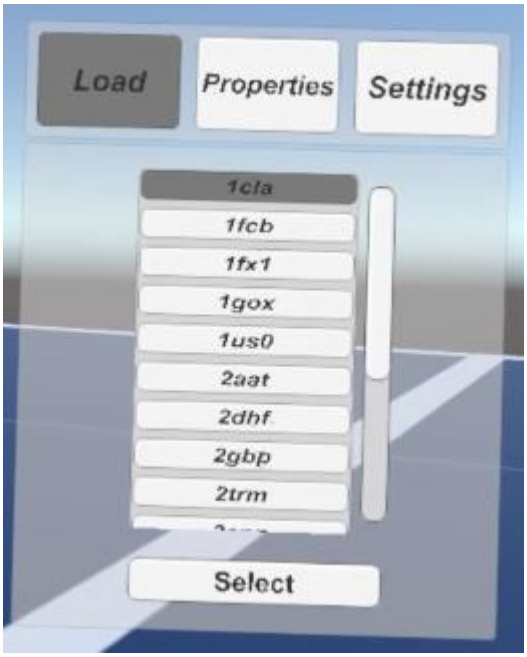


Figure 37 Scene capture of the load menu

        - Button Scroll List: The button scroll list will contain a set of buttons with the label names of all the .pdb filenames, one button for each file. It has two input interactions methods: the

first one, is using the GUI scroll bar presented at the right of the scroll list. The second one will be through scrolling the touchpad sensor. In the button scroll list, only one button is able to be selected at time.

- Select Button: This button, found on the bottom of the scroll list, will be the responsible of loading the .pdb models to our molecule mesh. When the button is clicked, the actual molecule mesh will be cleared and it will be loaded with the molecule with the same filename as the last scroll button is selected. This action will be ignored if the selected model is the same as the actual model.

- Properties Menu: The properties menu will be responsible of modifying the molecule's traits and properties. It will be composed by the following UI elements:

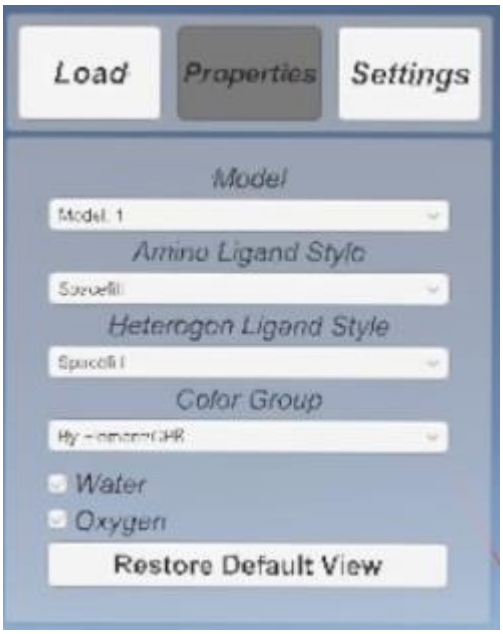


Figure 38 Scene capture of the properties menu

- Model Dropdown: Some .pdb files can contain more than one protein models. The model dropdown will load and number all

the models that the molecule has. The default loaded model will be the first one.

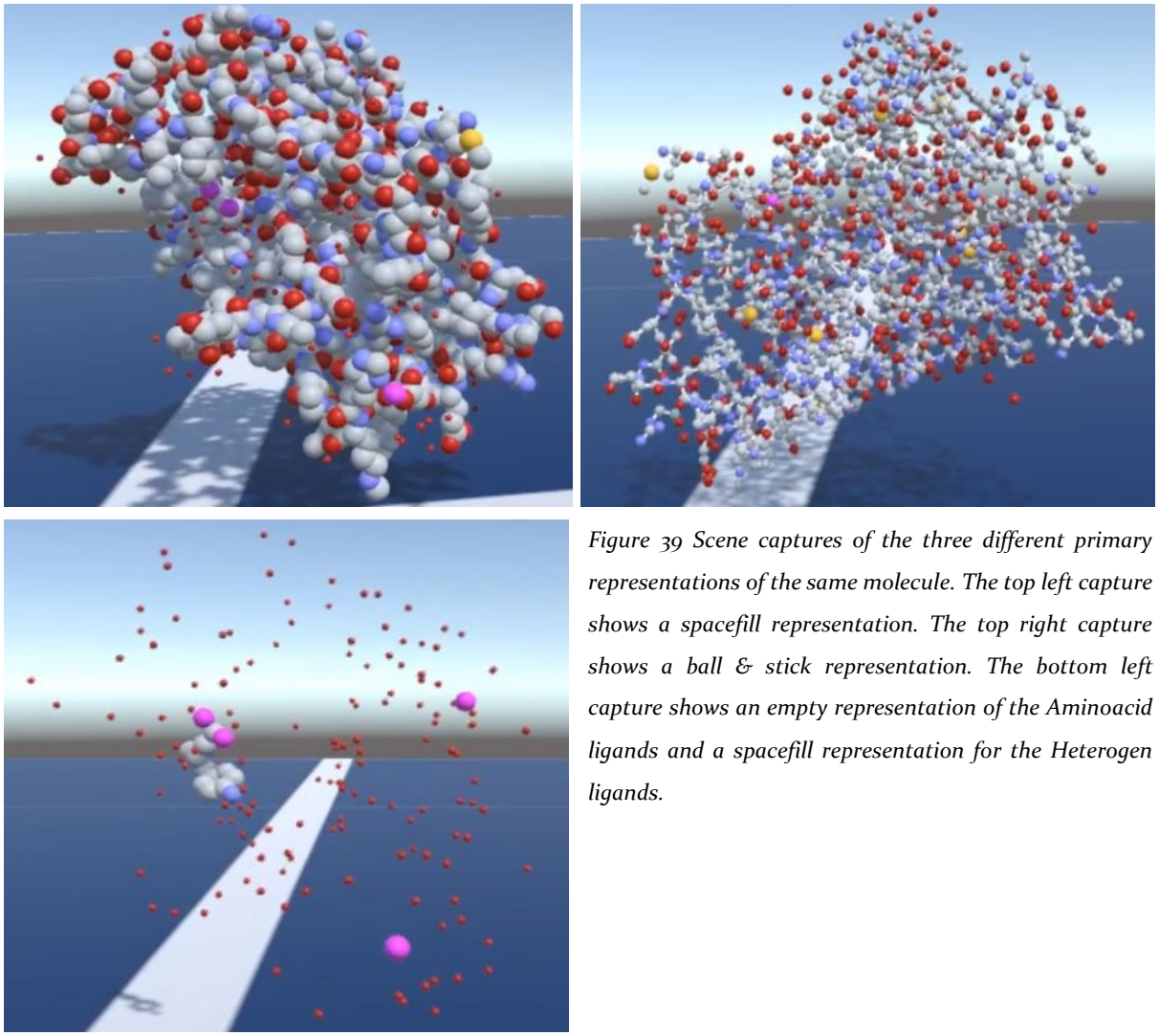


Figure 39 Scene captures of the three different primary representations of the same molecule. The top left capture shows a spacefill representation. The top right capture shows a ball & stick representation. The bottom left capture shows an empty representation of the Aminoacid ligands and a spacefill representation for the Heterogen ligands.

- Amino Ligand Style Dropdown: The main functionality of the amino dropdown will be to alternate through the available primary structure's representations of the Amino Acids. It has the following possibilities:
  - Spacefill Primary representation. (Default mode)
  - Ball & Stick Primary representation.
  - None Primary representation (Empty).

- Hetero Ligand Style Dropdown: As the amino style dropdown, the functionality of the hetero dropdown will be to alternate through the available primary structure's representations of the Heterogens molecule atoms. It has the following possibilities:
  - Spacefill Primary representation. (Default mode)
  - Ball & Stick Primary representation.
  - None Primary representation (Empty).
- Color Group Dropdown: The color group dropdown will be responsible of grouping the atoms in colors depending of the selected clustering mode. The different possibilities are the following:
  - By element/CPK: This mode will color the atoms following the CPK coloring table. (Default mode)
  - By Residue: The atoms will be colored depending of its residue value. There are 25 different colors and they will be alternated making the module of the residue's order value.
  - By Chain: The atoms will be colored depending of the chain number. There are 25 different colors and they will be alternated making the module of the chain's order value.
  - Amino/Hetero: The atoms will be colored depending if they are amino or hetero. The amino atoms will be

colored in red, while the hetero atoms will be colored in yellow.

- Water Toggle: The water toggle will toggle the water atom's visibility. If the toggle is on, the water atoms will be active whereas, if the toggle is off, the water atoms will be hidden. The default mode will be ON.

- Oxygen Toggle: The oxygen toggle will toggle the oxygen atom's visibility. If the toggle is on, the oxygen atoms will be active whereas, if the toggle is off, the oxygen atoms will be hidden. The default mode will be ON

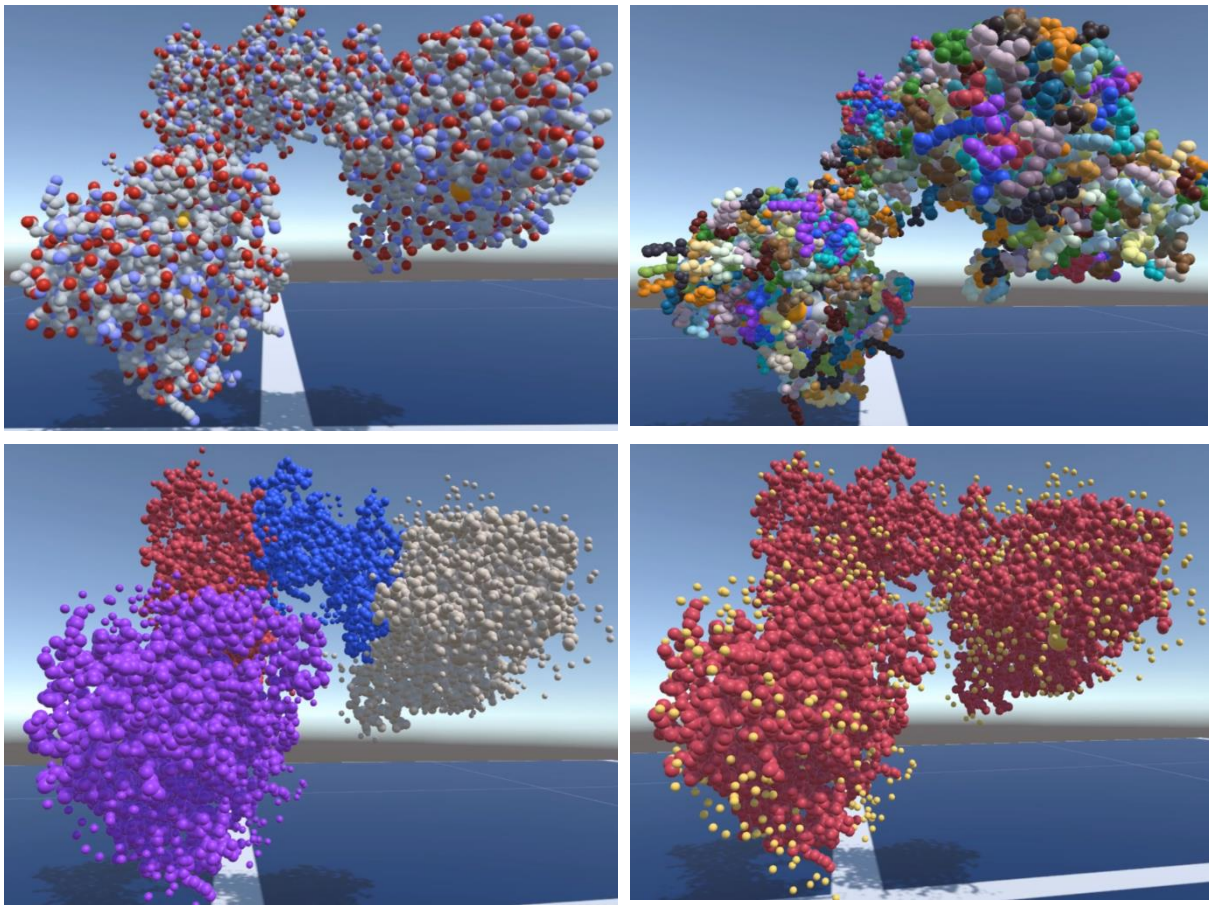


Figure 40 Scene captures of the same molecule with different coloring set groups. The top left capture presents the CPK color set. The top right capture presents the Residue color set. The bottom left capture presents the Chain color set. The bottom right capture presents the Amino/Hetero color set.

- ❖ Restore Default View Button: This button will be the responsible of restoring the default settings of the properties panel. If the button is clicked, the elements will be restored to its default value.

- ➢ Settings Menu: The settings menu will be responsible of managing the settings value of the molecule & plane transformations. It will be composed by the following UI elements:

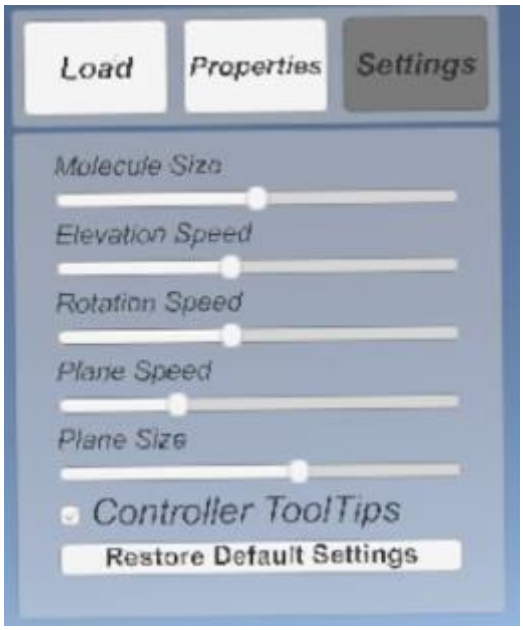


Figure 41 Scene capture of the settings menu

- ❖ Molecule Size Slider: The molecule size slider will be responsible of managing the molecule's mesh transform size. If the slider value is on the right, the molecule will be bigger whereas, if the slider is on the right, the molecule will be smaller.

- ❖ Elevation Speed Slider: The elevation speed slider will manage the amount of distance that will take part on the molecule's altitude displacement. If the slider is on the right, the displacement will be greater whereas, if the slider is on the left, the displacement will be lesser.

- Rotation Speed Slider: The rotation speed slider will manage the amount of rotation the molecule will perform. If the slider is on the right, the rotation will be greater whereas, if the slider is on the left, the rotation will be lesser.

- Plane Speed Slider: The plane speed slider will manage the amount of distance displaced by our plane culler. If the slider is on the right, the displacement will be greater whereas, if the slider is on the left, the displacement will be lesser.

- Plane Size Slider: The plane size slider will manage the size of our plane culler. If the slider is on the right, the plane will be bigger whereas, if the slider is on the left, the plane will be lesser.

- Controller Tooltips Toggle: The controller tooltips toggle will toggle the tooltips' visibility. If the toggle is on, the controller's tooltips will be active whereas, if the toggle is off, the controller's tooltips will be hidden. The default mode will be ON.

- Restore Default View Button: This button will be the responsible of restoring the default settings of the properties panel. If the button is clicked, the elements will be restored to its default value. The controller tooltip toggle will not be affected.

When the menu is active, it will always be located on front of the camera user. If we turn the camera, the canvas will move accordingly. However, when the controller receives a trackpad touch input, the canvas position is blocked and will not be reposition

back to the user's camera's the controller until it stops receiving the controller's input. This way the menu will be displaced while we are changing the parameters of the menu.

- The stored selection canvas will be responsible of giving the proper information of the actual molecule's selection. There are three different possible canvas:

  - No Selection Canvas: The no selection canvas indicates that no atom or bond is selected. This canvas is composed by a single text label. The text is in bold and with red font color.

Figure 42 Scene capture of the no selection canvas

  - Atom Properties Canvas: The atom selection canvas will be responsible of showing the selected atoms properties. It will have the following label parameters:

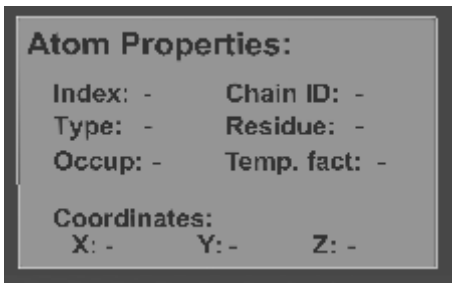


Figure 43 Scene capture of the atom properties canvas

    - ❖ Index Label: The index label displays the index of the selected atom.

    - ❖ Type Label: The type label displays the element type of the selected atom.

- ❖ Occupancy Label: The occupancy label displays the occupancy of the selected atom.

- ❖ Chain ID Label: The Chain ID label displays the chain number identifier of the selected atom.

- ❖ Residue Label: The residue label displays the index of the selected atom.

- ❖ Temperature Factor Label: The temperature factor label displays the degree value of the selected atom, in which the atom electron density is spread out.

- ❖ Coordinates Label: The coordinates label displays the three-dimensional coordinates of the selected atom. The coordinates are the crystallography values and no the unity's mesh coordinates.

- Bond Properties Canvas: The bond selection canvas will be responsible of showing the selected bonds properties. It will have the following label parameters:

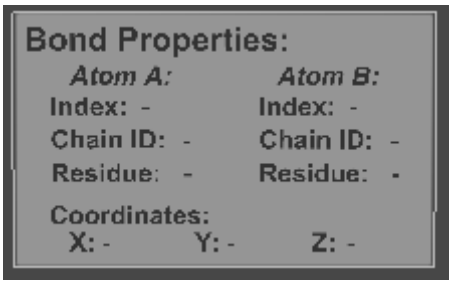


Figure 44 Scene capture of the atom properties canvas

- ❖ Index Label: The index label displays the index of the atoms which are bonded.

- ❖ Type Label: The type label displays the element type of the atoms which are bonded.

- ❖ Chain ID Label: The Chain ID label displays the chain number identifier of the atoms which are bonded.

- ❖ Coordinates Label: The coordinates label displays the three-dimensional coordinates of the selected bond. The coordinates are the crystallography values and no the unity's mesh coordinates.

As the menu canvas, when the stored canvas is active, it will always be located on front of the camera user. If we turn the camera, the canvas will move accordingly. The stored selection canvas will not be interactable, and the selector ray of the controller will ignore any collision kind.

- Selector Ray -> *Trackpad Press*

The selector ray will trace a ray function through the forward vector of the right haptic controller. The ray will be constantly traced until the trackpad sensor is pressed. It will be rendered as a red segment, which will act as GUI indicator. The collision of the ray will be marked with a red dot. It will have two main functionalities:

- The first functionality will be to manage and indicate the candidate's selection of atoms and bonds. When an atom or bond is ray traced by the selector ray, we will use again the world space silhouette technique as visual indicator. The silhouette will be rendered in turquoise color. The silhouette will have the same depth value as the rendered mesh.

- The second functionality will be to indicate the position of the ray on the canvas menu.

- Selector / Canvas Blocker Mode -> *Trigger Button*

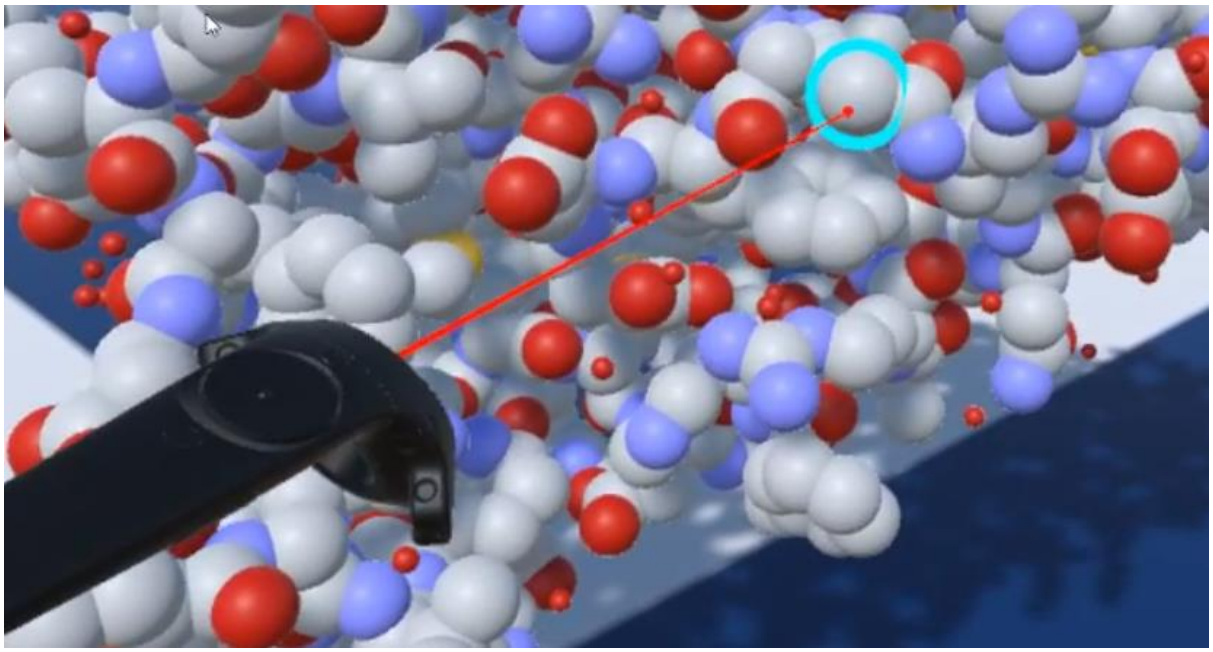


Figure 45 Scene capture of the selector ray collision in an atom. The ray will be rendered in red while, the geometry mesh will have a world space silhouette as visual indicator.

The trigger button will have two different modes, the selection mode and the canvas block mode:

- The first mode is the selector mode. This mode will be available while we are pushing the controller's trackpad sensor and a selection ray is tracing. When the trigger is released, the ray's collided mesh will be selected and the pertinent selection canvas will be activated. Otherwise, if the ray doesn't trace any molecule mesh, the selection will be set to null.

  When an element is selected, we will use the world space silhouette technique as GUI indicator. This silhouette will be in green color. Moreover, its silhouette will visible over any other atom or bond meshes that are in front of it.

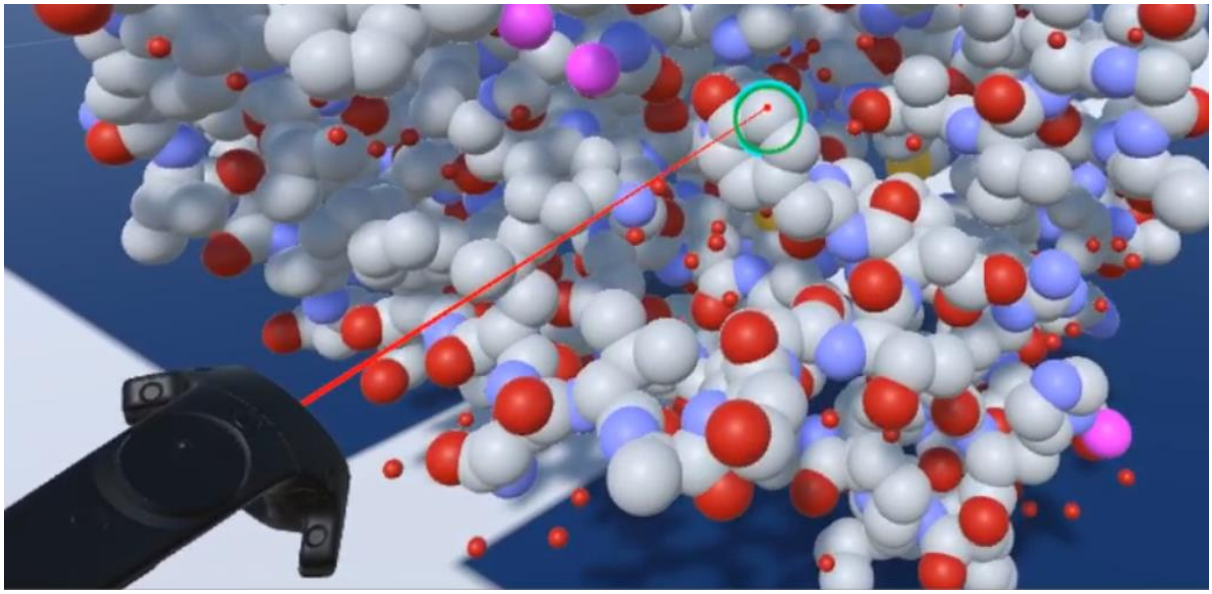


Figure 46 Scene capture of the selector ray selecting an atom. The selected atom will present a world space silhouette as visual indicator.

The selection method will be captured in trigger release mode because, sometimes the selection can be tricky and, select the desired atom or bond can overwhelming and complicated. If we use the trigger release mode, we are able to hold the trigger fully pressed and activate the trigger release input while we loosen the finger with a fast motion, making the selection more responsive.

Last but not least the selector mode will be also responsible of the menu's items inputs. These inputs will be emitted in trigger click mode.

- The second mode is the canvas blocker mode. The input will be captured on trigger clicked mode. This mode will only be available when any of the menu button's canvas are active and no trackpad sensor input is captured. When the trigger is clicked, the active canvas will toggle between the block mode, in which it freezes and remains to its position, or the forward camera mode, in which it stays in front of the user's camera forward vector.

- Masking Plane Toggle Mode -> *Grip Button*

The grip button will be responsible of activating and deactivating the masking plane mode:

- When the mode is on, a wireframe green plane will appear in front of the user and will active the molecule's masking mode, which will be discussed later, in *chapter 6.3*.

- When the mode is off, the masking mode will be reset and will be deactivated.

6.3. Masking Technique

6.3.1. Dynamic clipping plane

Once we have implemented the main features of our molecular viewer, we can proceed to implement one method that allows to the users to select and visualize the more inner atoms of the molecule. Usually, the molecule proteins atoms tent to have a thick form and we are only able to see the most external layer of atoms. Thus, it's essential to provide any technique to permit to visualize the full coats a molecule presents.

In this chapter we will introduce the dynamic clipping plane technique, in which we will mask every atom and bond that is behind the plane:

The clipping plane will have a GUI indicator, which will be composed by a green wireframe quad. Its size can be edited in the settings menu. In some cases, the plane's wireframe could be out of the camera's range. Therefore, we will also render the wired diagonals over the quad in order to constantly know where and in which distance the plane is.

An atom will be clipped and hence, hidden form the scene, if and only if the distance from the user's camera center to the clipping plane is bigger than the distance to the mesh in question. The Y space coordinate will be ignored and set to one, since we want to clip the molecule perpendicularly to the ground.

The masking plane will have to be responsible to the molecule transformations and act according to them. When the molecule rotates for example, there exists the possibility than some meshes are displaced in front of the plane and some others behind. The plane has to live update all the masked atoms and bonds without compromising the system performance.

We had two different possibilities to implement the masking method; one by GPU and the other by CPU. We have decided to implement via CPU since constantly updating the transformations can be a time-consuming process, and we don't want to create a bottleneck on our shader.

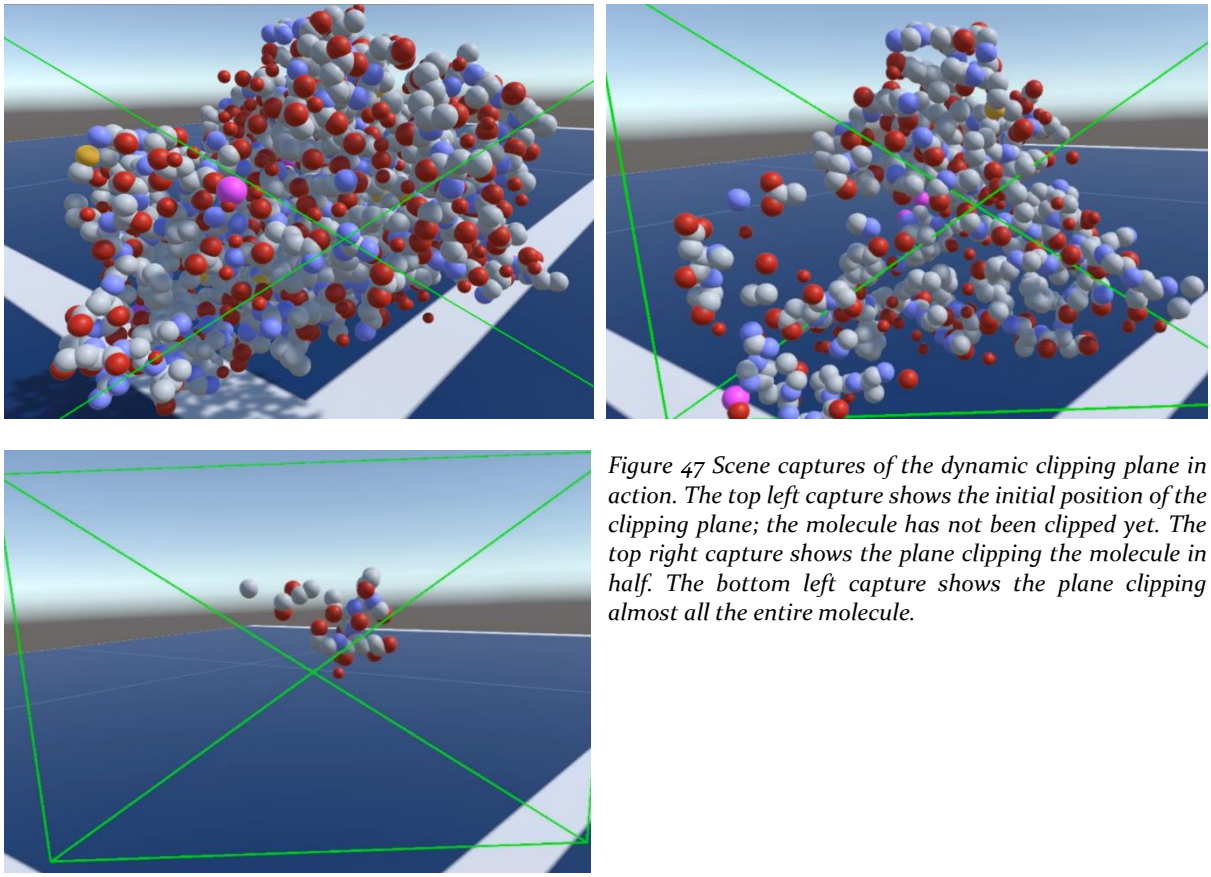


Figure 47 Scene captures of the dynamic clipping plane in action. The top left capture shows the initial position of the clipping plane; the molecule has not been clipped yet. The top right capture shows the plane clipping the molecule in half. The bottom left capture shows the plane clipping almost all the entire molecule.

6.3.2. Masking Events Description

Once we have toggled and activated the masking mode with the controller's right grip button, the wireframe plane indicator will appear in front of the user camera, perpendicularly to the forward vector. The plane will follow the user's displacements or camera movements and will keep its position.

The plane is set to a default position but, we can manage the plane's displacement thought the trackpad sensor touch. The X axis will be ignored. If the tracked value is positive, the masking plane will go forward whereas, if the value is negative, the masking plane will go backwards. The amount of value tracked by the sensor affects the quantity of rotation degrees; if the value of the Y axis is closer to the -1 or 1 values (left & right sides respectively) it will rotate more than a value close to the 0 value (center).

7. Conclusions

7.1. Summary of the project

Molecular visualization software is an upcoming and growing sector than it's tracing its path thanks to the incoming technologies. Not only allows to compute time-consuming CPU and GPU programs and renders in lesser time, but also introduces more complex and complete visualization methods that allow to research and investigate the molecules and proteins in a more interactive and comprehensible way.

The recent introduction of the VR to the market is a very important point to remark, since the launched VR technologies are such recent than they lack a reference or even an open source visualizer in the biomolecular and biochemist world that would provide such visualization methods. The VR interaction methods are still on develop and investigation processes and no techniques or standard methods are properly defined yet.

However, in this project, we have taken advantage of the recent launch of the commercial VR glasses in order to construct an alternative way to visualize the molecule and protein models. This visualizer not only allows the user to read any molecule from the most common and used codification molecular file types from the biggest molecular dat49se, but also allows to control and modify the molecule properties with a pair of haptic controllers in which the controller ray trace will be the main interaction method.

During the process we have developed a strictly project planification that allowed us to achieve our objectives satisfactorily; we have built a complete molecule parser and data model which can read thousands of the mainly three-dimensional structural data for large biological molecules. We have introduced and implemented three different atom and bond geometry representation methods, and therefore, discuss its advantages and drawbacks of each method as well as finding its bottlenecks. We have designed a complete three-dimensional graphical user interface that interacts with the represented

molecule and follows a Three-tier architecture. We have implemented a complete ray-based selection technique. Finally, we implemented a dynamic clipping plane than activates and deactivates the meshes based on the relative position of the camera.

For the proper accomplishment of the project tasks, we have developed a series of methods and techniques that reduced considerably the render and computing process time. For each technique, we have developed a series of performance and stress tests in order to compare it with other available methods and see its disadvantages and bottlenecks as wells as its strong traits.

During the realization of this project, we have accomplished our proposed project objectives; we presented molecular viewer, in which any user is able to properly interact with ease with the molecule while it keeps a well-defined and rendered mesh that reacts to the graphical interface transformations and properties modifications as well as provides comfortable selection and masking techniques. This viewer also provides a fully customizable menu options sets that not only englobes interface properties but also model projection properties.

However, we found some troubles during the realization of the project. We encountered really hard problems with the implementation of the billboard impostor technique through Cg Language; it's basic syntax and depth buffer's capabilities didn't allowed us to properly instance the depth value of our impostor mesh. Furthermore, the rupture of the computer's graphics card that required a replacement of another capable of running the VR system and lately, the temporal unviability of the VR headset has taken us away from our planning and forced us to take alternative path in order to succeed with our objectives.

7.2. Future research

Before we initially started our project, we determined a strict schedule for our guidelines to follow. During each step plan, we set a series of restrictions in order to accomplish or goals and objectives. However, we there still certain limitations that impede to our molecular viewer to bring the full potential of our project.

If we truly want to offer a complete visualizer, that squeezes all virtual reality's immersion capabilities as well as brings a full visualization of the biomolecular structures of the actual reference techniques, there are still many methods and implementations and future researches we could make.

In this section we will introduce and discuss some possible enhancements that we could implements to our molecular viewer as well as we will deliberate about some possible researches, we could make in order to improve our render and computational performance;

Representation of other protein visualization structures: during the realization of this molecular viewer, we have only represented the primary structure of proteins. However, there still many others three-dimensional representations we could render and recreate in our viewer. Those methods are also really important in the biomolecular and biochemical world, and are frequently used by the researchers to interpret and interact with the catalogued molecules. Even though the primary representation was made by some of the most knowing and basic geometrical forms, the other structures requires more complex forms and meshes. We could generate the structure meshes based on the atom's coordinates and information than we have stored in the file and in the data model. Even we could try to generate an impostor of the hypothetical mesh.

Introduction of other selection techniques: In our project we have introduced two different selection techniques. Though, we could still include more methods to make different precision selections. One possible way could be to take

advantage of the k-d tree and implement, for example, to implement a nearest neighbor selection which guided with our trackpad. This obviously means to perform a series of enhancements and operations to or k-d tree algorithm, which will also take into account the slide direction to select the incoming element, taking into account its camera to object transformations, etc.

Investigate more molecular geometry rendering techniques: In *chapter 4* we introduced and subsequently analyzed and compared many different renderings of molecular geometries that allowed us to make possible visualize huge molecular models such as the proteins. We have discussed the good points and the limitations of each one and, with the results and analysis extracted, we could improve our render performance to increase our frame rate. For example, we could research and investigate how to expand the limitations of the GPU instancing in order to allow to the buffer to include more instanced meshes in order to allow the instantiation of bigger molecular models. Unity has an open source version and it supports a huge active community of developers. As the same way we have implemented the pertinent methods to allow multiple color instantiation, we could research for possible methods we could implement to improve our performance.

These are only a few of the possible enhancements and investigations we could perform to our project in order to build, step by step, a more complete molecular viewer. Furthermore, the VR tools and sources are continuously evolving and the incoming community and research investigations are on top of the day. If we keep looking for new articles that are related to the biomolecular and biochemical world, as well as the introduction of new VR interaction techniques, we could take advantage of it and implement it to our project.

7.3. Concluding statements

Not all geometry rendering techniques are able to represent the molecular geometries. Molecules has a very high number of atoms and bonds and can be really expensive if we talk about its computational cost. The most well-known used molecular representation technique is the billboard impostor technique.

However, there still other viable representation methods such as the GPU instancing. Each method has its strong points and weaknesses nonetheless, the proper implementation and working praxis allowed to our molecular visualizer to perform a real time rendering and visualizations that doesn't compromise our system performance.

The introduction of the VR tools allowed us to implement a deeper and more immersive molecular viewer. A world space GUI not only brings the advantage that doesn't partially occupy the user viewport, but also leads to a more potentially awareness of the changes produced on the represented molecule. Furthermore, the introduction of the VR trackpads replacing the conventional mouse and keyboard enables a full range of three-dimensional space selections as well as a more complete control of the mesh transformations.

8. Bibliography

[1] «Átomo», in Diccionario de la Lengua Española (22 ed.). Real Academia Española (2001). [Website]. Retrieved March 1, 2018.

[2] Kumari, I., Sandhu, P., Ahmed, M., & Akhter, Y. (2017, August 30). *Molecular Dynamics Simulations, Challenges and Opportunities: A Biologist's Prospective*. Retrieved April 07, 2018, from https://www.ncbi.nlm.nih.gov/pubmed/28637405

[3] McNaught, A. D., & Wilkinson, A. (2000). *IUPAC compendium of chemical terminology* (PDF), Cambridge, England: Royal Society of Chemistry. Retrieved March 1, 2018.

[4] "Proteïna". Anonymous. *Gran Enciclopèdia Catalana* (Website), Barcelona, Spain: Enciclopèdia Catalana (1965). Retrieved March 1, 2018 from https://www.enciclopedia.cat/enciclopedies.

[5] "Amino acid". *Cambridge Dictionaries Online* (Website), Cambridge University Press. 2015. Retrieved March 1, 2018 from https://dictionary.cambridge.org/dictionary/.

[6] Jessica Mayand and David S.Goodsell. (2017). *What is a Protein?* (Online Video) ,20 November 2017. Retrieved February 28, 2018 from http://pdb101.rcsb.org/learn/videos/what-is-a-protein-video.

[7] D. Voet and J.G. Voet. *Biochemistry* (Book), 4a ed. John Wiley & Sons, 2010. Retrieved March 1, 2018.

[8] S.S. Batsanov. *Van der W35s Radii of Elements* (PDF). Inorganic Materials, 2011. pg 1031-1046. Retrieved March 4, 2018.

[9] Corey, Robert B.; Pauling, Linus. *Molecular models of amino acids, peptides, and proteins* (PDF). Review of Scientific Instruments, (1953). pg 621–627. Retrieved March 4, 2018.

[10] Olmsted J, Williams GM (1997). *Chemistry: The Molecular Science* (PDF). Jones & Bartlett Learning. pg 87. Retrieved March 4, 2018.

[11] Richardson, Jane S. (2000), *Early ribbon drawings of proteins* (PDF), Nature Structural Biology, pg 624–625. Retrieved March 5, 2018.

[12][13] Bank, R. P. (n.d.). Homepage. [Website]. Retrieved March 10, 2018, from https://www.rcsb.org/

[14] Corey, Robert B.; Pauling, Linus. *Molecular models of amino acids, peptides, and proteins* (PDF). Review of Scientific Instruments, (1953). pg 621–627. Retrieved March 4, 2018.

[15] Olmsted J, Williams GM (1997). *Chemistry: The Molecular Science* (PDF). Jones & Bartlett Learning. pg 87. Retrieved March 4, 2018.

[16] Richardson, Jane S. (2000), *Early ribbon drawings of proteins* (PDF), Nature Structural Biology, pg 624–625. Retrieved March 5, 2018.

[17] Technologies, U. (n.d.). Unity Manual. Retrieved March 10, 2018, from https://docs.unity3d.com/es/current/Manual/index.html

[18] J. F. (n.d.). Unity C# Tutorials. Retrieved March 10, 2018, from http://catlikecoding.com/unity/tutorials/

[19] The Cg Tutorial. (n.d.). Retrieved March 12, 2018, from http://developer.download.nvidia.com/CgTutorial/cg_tutorial_chapter01.html

[20] Schawarzer F. A., & Lotan I. *IUPAC Approximation of Protein Structure for Fast Similarity Measures* (PDF), Computer Science, Stanford University, Stanford, CA. Retrieved March 11, 2018.

[21] P. R. (n.d.). *Learning Modern 3D Graphics Programming*. Retrieved March 8, 2018, from https://paroj.github.io/gltut/Illumination/Tutorial%2013.html

[22] P. R. (n.d.) (2012, February 07). Paroj/gltut. Retrieved March 12, 2018, from https://github.com/paroj/gltut/tree/master/Tut%2013%20Impostors

[23] Technologies, U. (n.d.). *GPU instancing*. Retrieved March 12, 2018, from https://docs.unity3d.com/Manual/GPUInstancing.html

[24] (n.d.). *Models and Structural Diagrams in the 1860s*. Retrieved April 08, 2018, from https://webspace.yale.edu/chem125/125/history99/6Stereochemistry/models/models.html

[25] (n.d.). *Proteins, Scientific figures*. Retrieved April 08, 2018, from http://www.wikipremed.com/image_archive.php?code=040101

[26] Allen Sherrod *Game Graphics Programming* (PDF), Course Technology PTR Retrieved August 21, 2018.

[27] Dr. Steve Cunningham *Computer Graphics: Programming, Problem Solving, and Visual Communication* (PDF), Computer Science Department California State University Stanislaus Turlock, CA, pg 274. Retrieved August 22, 2018.

[28] Moller, Thomas *Real-Time Rendering, Fourth Edition* (PDF), Computer Science Taylor & Francis Boca Raton, CA, pg 559. Retrieved August 22, 2018.

[29] Larson, Brad *Enhancing Molecules using OpenGL ES 2.0* (Website), published 11 May 2011. Retrieved August 18, 2018 from http://www.sunsetlakesoftware.com/2011/05/08/enhancing-molecules-using-opengl-es-20

[30] "Sphere". *Oxford Living Dictionaries* (Website), Cambridge University Press. 2015. Retrieved February 9, 2018 from https://en.oxforddictionaries.com/definition/sphere

[31] "Icosahedron". *Oxford Living Dictionaries* (Website), Cambridge University Press. 2015. Retrieved February 9, 2018 from https://en.oxforddictionaries.com/definition/sphere

[32] Jason L. McKesson *Learning Modern 3D Graphics Programming* (Website), published 2012. Retrieved March 23, 2018 from http://www.cse.chalmers.se/edu/year/2018/course/TDA361/LearningModern3DGraphicsProgramming.pdf

[33] Sebio, Oscar Cajaraville *Four Ways to Create a Mesh for a Sphere* (Website), published December 7, 2015. Retrieved February 10, 2019 from https:// medium.com/game-dev-daily/four-ways-to-create-a-mesh-for-a-sphered7956b825db4

[34] Moore, *"What is a ray?"* (Article), Zemax. published July 25, 2005. Retrieved February 14, 2019.

[35] "Proteïna", *Wolfram Math World* Weisstein, Eric W. (Website). Retrieved February 21, 2019 from http://mathworld.wolfram.com/Norm.html

[36] Straßer, Wolfgang *Schnelle Kurven- und Flächendarstellung auf grafischen Sichtgeräten* (PDF), Chapter 6 (page 6-1), published December 26, 1974. Retrieved February 20, 2019

[37] n.a. *GPU instancing* (Website), published February 18, 2019. Retrieved February 28, 2019 from https://docs.unity3d.com/Manual/GPUInstancing.html

[38] Dutra, Teofilo *Unleash Your Gpu Instancing* (Website), published January 2, 2018. Retrieved February 10, 2019 from https:// medium.com/game-dev-daily/four-ways-to-create-a-mesh-for-a-sphered7956b825db4

[39] J. Cheminford *A rule-based algorithm for automatic bond type perception* (Article), published October 31, 2012. Retrieved March 10, 2018 from https:// medium.com/game-dev-daily/four-ways-to-create-a-mesh-for-a-sphered7956b825db4

[40] Altman, N. S. *An introduction to kernel and nearest-neighbor nonparametric regression* (pdf), published 1992. Pg. 175-185. Retrieved March 7, 2019 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1010.2854&rep=rep1&type=pdf

[41] Bentley, J. L., N. S. *Multidimensional binary search trees used for associative searching* (pdf), published 1975. Pg. 509. Retrieved March 7, 2019 from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.160.335&rep=rep1&type=pdf

[42] Blum, M.; Floyd, R. W.; Pratt, V. R.; Rivest, R. L.; Tarjan, R. E. *Time bounds for selection* (pdf), published 1973. Pg. *448–461.* Retrieved March 7, 2019 from http://people.csail.mit.edu/rivest/pubs/BFPRT73.pdf

[43] n.a. *Stencil Test* (Website), published September 12, 2015. Retrieved March 10, 2019 from https://www.khronos.org/opengl/wiki/Stencil_Test

[43] n.a. *Cg Programming/Unity/Outlining Objects* (Website), published August 18, 2018. Retrieved January 5, 2019 from https://en.wikibooks.org/wiki/Cg_Programming/Unity/Outlining_Objects

[44] n.a. *Unity User Manual (2018.3)* (Website), published February 26, 2019. Retrieved March 5, 2019 from https://docs.unity3d.com/Manual/UnityManual.html

[45] n.a. *OpenVR* (Website), published March 3, 2019. Retrieved March 5, 2019 from https://docs.unity3d.com/Manual/UnityManual.html

[46] H.M. Berman, K. Henrick, H.Nakamura, J.L. Markley *The Worldwide Protein Data Bank(wwPDB)* (Website), published 2007. Retrieved April 08, 2018 from http://www.wwpdb.org/

[47] David S. Goodsell. *Introduction to PDB Data* (Website), published n.a. Retrieved April 07, 2018from https://docs.unity3d.com/Manual/UnityManual.html

[48] Roques, Arnau *PlantUML* (Website), published April , 17 2009. Retrieved March 14, 2019 from http://plantuml.com/

[49] n.a. *Graphical user interface* (Website), published March, 16 2019. Retrieved March 16, 2019 from https://en.wikipedia.org/wiki/Graphical_user_interface

[50] n.a. *Command line vs. GUI* (Website), published December, 13 2018. Retrieved March 16, 2019 from https://www.computerhope.com/issues/ch000619.htm

[51] n.a. *VRTK - Virtual Reality Toolkit* (Website) Retrieved April 4, 2019 from https://vrtoolkit.readme.io/docs/sdk-setup-switcher

[52] Ian Hamilton, *VRTK's Open Source Tools Help New Developers Get Started In VR* (Website)., published September, 28 2017 Retrieved April 4, 2019 from https://uploadvr.com/vrtk-stone-fox-unity-tool/