TÉCNICO LISBOA

Fast Photorealistic Rendering of Protein Representations

Nelson Miguel Matos de Oliveira Ferreira

Thesis to obtain The Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Hugo João António Madeiras Pereira

Dr. Francisco Jorge Dias Oliveira Fernandes

Examination Committee

Chairperson: Prof. Nuno João Neves Mamede

Supervisor: Dr. Francisco Jorge Dias Oliveira Fernandes

Members of the committee: Prof. Abel J.P. Gomes

November 2022

Acknowledgments

Firstly, and specially, I would like to thank my supervisors, Professor Doctor João Pereira and Professor Doctor Francisco Fernandes, for all the patience, guidance, wisdom, knowledge, support and kindness given throughout this past year, they were the main factor that I could deliver this.

I will never know how to thank my close friends for being there, by my side, every step of the way, you are the reason I could go through this journey with all my energy after all problems and struggles, in particularly Afonso, all my bachelors' ex-colleagues from Coimbra, my house roommates and my Masters colleagues that walked with me through this journey.

Margarida, you helped me to overcome some critical difficulties back in the time and for that you deserve my thanks.

To my Capgemini colleagues at work that were comprehensive enough and flexible with me, allowing me to conciliate work with thesis and at the same time giving me emotional support along with some chilling and fun moments.

A final and special thanks to my mom and stepfather for providing me with all the conditions necessary to be able to study and work.

Abstract

Proteins are biomolecules essential for life and responsible for many of the biological processes and reactions present in every living organism. The existence of computational supportive tools for viewing and presenting results, reasoning and formulating hypotheses related to their molecular structure is crucial for the development of scientific areas like chemistry, biology etc. However, this kind of visualization tools are very computational heavy and demanding since there are structures with hundreds of thousands of atoms. For this scenario, we have standalone tools that already present acceptable performance and a wide range of visualization features, but they are not very approachable for students or a more casual audience aiming for quick and simple investigations. On the other hand, there are web browser applications that are trying to achieve the same features as the standalone ones and at the same time be more accessible for everyone. Still, they are conditioned by the web programming languages' performance and their inability to deal with heavy amounts of data. Considering this scenario, the main objective of this work is to deal with these limitations on the web browser, loading the highest possible number of atoms using the least memory possible and implementing the same visualization features which are only available in standalone applications.

Keywords

JavaScript/WebGL
Web Application
Protein Data Bank
Image-Based Rendering
Impostors/Billboards
Ray-tracing
Global Illumination

Resumo

As proteínas são biomoléculas essenciais à vida e responsáveis por muitos dos processos e reações biológicas presentes em cada organismo vivo. A existência de ferramentas computacionais de apoio à visualização e apresentação de resultados, raciocínio e formulação de hipóteses relacionadas com a sua estrutura molecular é crucial para o desenvolvimento de áreas científicas como química, biologia, etc. No entanto, este tipo de ferramentas de visualização são muito pesadas e exigentes em termos computacionais, uma vez que estamos a falar de estruturas com centenas de milhares de átomos. Para este cenário, temos as aplicações *standalone* que já apresentam um desempenho aceitável e uma vasta gama de *features* de visualização, mas que não são muito acessíveis ou de fácil uso paras estudantes ou para um público mais casual, uma vez que esses utilizadores têm o objetivo de as usar para realização investigações rápidas e simples. Por outro lado, existem aplicações de navegação web que estão a tentar alcançar as mesmas características que as *standalone* e ao mesmo tempo ser mais acessíveis para todos. Ainda assim, são condicionadas pelo desempenho das linguagens de programação web e pela sua incapacidade de lidar com grandes quantidades de dados. Considerando este cenário, o principal objetivo deste trabalho que incide em lidar com estas limitações no browser, carregando o maior número possível de átomos utilizando o mínimo de memória possível e implementando as mesmas características de visualização que só estão disponíveis nas aplicações *standalone*.

Palavras-chave

JavaScript/WebGL
Aplicação Web
Base de Dados de Proteínas
Renderização Baseada em Imagens
Impostores/Cartaz
Ray-tracing
Iluminação Global

Table of Contents

List of Figures

List of Tables

Abbreviations

IBR – Image Based Rendering

MD – Molecular Dynamics

HM – Hybrid Methods

FOV – Field of View

GI – Global Illumination

PDB – Protein Data Bank

OC – Ambient Occlusion

1 Introduction

The chemical compounds that we humans ingest are mainly known by macronutrients, which provide us with most of our energy, being protein one of the three primary ones (the others being carbohydrate and fat). Every cell in the human body contains protein, which is an important nutrient, not just for athletes and bodybuilders but for everyone. Humans can't survive without all nine essential amino acids, and protein is essential to strengthen bones and body tissues (such as muscles), but it does much more than that, it participates in practically every process of a cell, playing a part in providing a source of energy, assisting in cellular repairing, metabolic reactions, form blood cells, acting as immune response, and more.

Massive biomolecular structures are being used and experimented daily setting up techniques such as electron microscopy (high resolution images of biological specimens) and crystallography (discerning the arrangement and bonding of atoms). Also, emerging integrative or hybrid methods (I/HM) are building structural models of vast macromolecular machines, at times containing more than hundreds of millions of atoms. Moreover, a file format was specifically in order to store all needed data from atoms and molecules about certain proteins, called Protein Data Bank (PDB) which is referenced with more detail in the sections below.

Having this situation, the interactive visualization of massive macromolecular complexes on the web is turning into a challenging issue as some techniques, such as those ones, advance at an unprecedented rate and deliver structures of increasing size, and they are a widely used tool in biological research.

Of course, displaying these molecular structures on the web and making them accessible to all educators, scientists, and students (not just experts with access to dedicated networking, hardware and software), is essential.

Despite the significant advances in molecular dynamics (MD) and biology research, there's still a lack of specialized bioinformatic tools. The difficulty lies in the efficient management of the data, in sending and processing 3D information for its visualization. In order to visualize such scenes at interactive rates, it is necessary to limit the number of geometric primitives rendered in each frame.

Molecular viewers are a vital tool for our understanding of protein structures and functions, because different 3D shape visual representations of proteins can give us visual clues about the protein structure and its functions. There are various types of protein representations and studying these adversities helps the biologists to better understand the protein behavior and to design proteins with modified properties. One of the most common approaches to these studies is to compare the protein structure with other molecules and reveal similarities and differences in their polypeptide chains (chain of amino acids).
One of the many challenges, in observing multiple protein representations at the same time and comparing each one of them, is that in some cases it is just not possible to scale the atoms enough in order to get the desired detailed information about them, because we are talking about hundreds of thousands of particles.

The general objective of all visualization modules is to reduce or avoid geometry data whenever possible, and recently, image-based rendering (IBR) techniques have emerged as an alternative to geometry-based systems for interactive scene display. With IBR methods, the 3D scene is replaced by a set of images, and traversing the scene is therefore independent of object space complexity.
Polygon meshes are a large field of computer graphics and a geometric modelling that simplifies rendering. Polygons is a collection of faces, vertices and edges defining a shape of an object and the respective faces usually consist of triangles or other simple convex polygons. With this, it is possible to apply a variety of operations like smoothing and Boolean logic or algorithms like ray tracing. In these approaches, 3D models are replaced by a small set of textured polygons that resemble the original geometry.
There are also imposters (Christiansen, 2005) (known as billboards) methods that are mainly used to reduce the time required to render a 3D scene, by caching images of 3D objects and using their images in place of the real objects in a scene that is only rendered as 2D objects when they are far enough from the camera. Presentation is tested in regards to the distance the camera has from the imposter and from what angle we look at the imposters from. In a few words, they are 2D elements incrusted into a 3D

world. This technique decreases the amount of work performed each frame results in less time spent rendering.

Basically, the web applications, to these days, have very basic visual features, they don't use ray tracing on the contrary of the standalone ones. Since, the atoms alongside with the polygons have a lot of information that occupy a lot of memory in the CPU (atom information loading) and in the GPU (when the spheres have a lot of vertexes, triangles and polygons etc.), in this work, it will be attempted to load the most possible quantity of atoms with the less possibly memory and optimize that process, that is crucial. For that, a parser and loader will be created to convert PDB files into the formats that the libraries eventually use, smoothing the all loading procedure. Also, a hybrid implementation will be done, that will consist in having impostors/billboards when the objects are too far away and ray tracing when the objects are close to the camera. This approach needs to be heavily considered, because ray tracing is a slow process and it's not bearable to have it working for all situations in a 3D scene with a lot of information going on simultaneously. This process will have as consideration the study and investigation performed in web ray tracing done in 2021 (Vitsas N. , 2021), alongside with the Rayground framework (Vitsas N. , 2020) that provides an easy way to test the algorithms and visualize their outputs right away.

2 Related Work

In this section the main aspects of the standalone and web tools for protein visualization will be addressed. A more detailed explanation and overview will be done to the web applications, since those ones will be the main reference for what will be implemented in the solution of the problem. In section 2.3 are referenced the most known and useful web tools and a brief summary will be presented for each one of them, mentioning their main aspects, advantages, principal results, limitations, weak points, how they evaluate, use, manage and test the data from the protein structures. Also, in section 2.1 a quick summary will be given about PDB file format which is the format that all these applications use to get the atoms data, alongside with a brief description regarding the various representation of proteins.

2.1 Protein Data Format and Representation

Protein Data Bank (PDB) format is a standard for files containing atomic coordinates, it is used for structures and is read and written by many programs. It is a text file consisting in lines of information having each line called a record. A PDB file generally contains several different types of records, arranged in a specific order to describe a structure.

As for describing molecular structures, PDB is the most commonly used way to store and share atomic coordinates, still its use is increasing every year, with over 600 million total downloads from the RCSB PDB (Berman, 2003). Although the data is kept in flat ASCII files, the PDB format is ubiquitous.

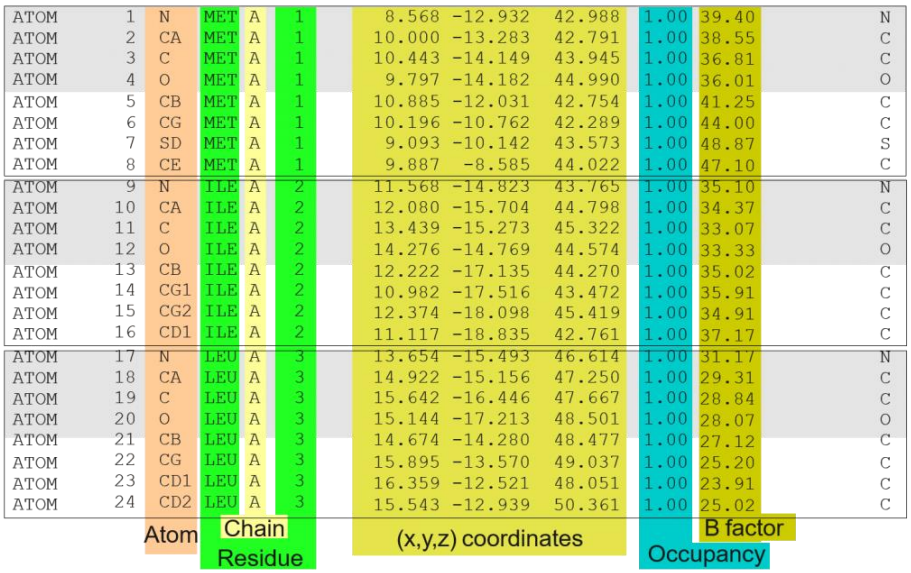

Fig. 1. Example of a PDB file format *(Botzki, 2021)*.

There are four types of protein representation: the space filling diagram that shows all atoms that are making up the protein, the ribbon/cartoon diagram shows the organization of the protein backbone and highlights the alpha helices, the surface representation shows the areas that are accessible to water molecules and finally the ball-and-stick model that displays both 3D positions of the atoms and the bonds between them.

One of many challenges in visualizing multiple protein representations and compare each one of them, is that such representations are very limited to its scalability and due to the occlusion problems, for example, the spatial representation is only possible for comparison in a few structures (Kocincová, 2017).

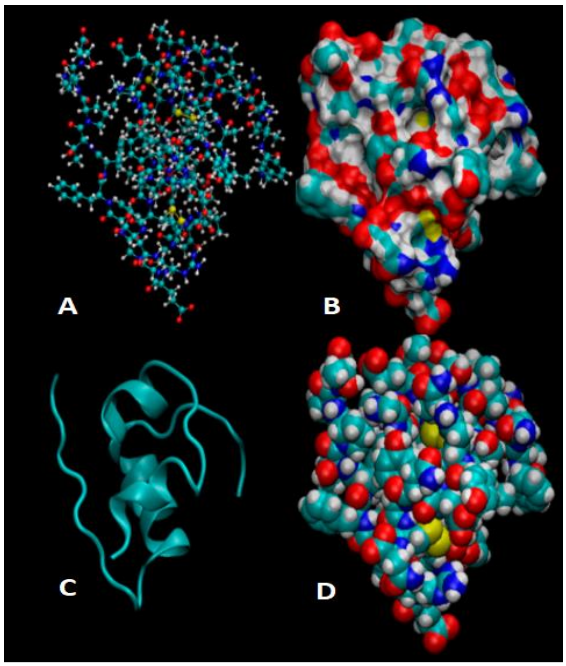

Fig. 2. Protein representations (2a - Ball and stick model, 2b – Surface model, 2c - Ribbon model, 2d – Space filling model) *(Mary, 2004)*.

The representation model that is going to be focused is the space-filling one. In this model, its envisioned the surface of the molecule as being determined by the Van Der Waals radius (radius of an imaginary hard sphere representing the distance of closest approach for another atom) of each atom of the molecule and crafted atoms as hard-wood spheres of diameter proportional to each atom's Van Der Waals radius. This representation reflects the electronic surfaces that molecules present, that dictate and show how they interact, one with another. The main difference between the space filling model and ball stick is that, in the ball and stick model, the molecular structures are depicted by spheres and rods, whereas, in the space filling model, the molecular structures are depicted by full-sized spheres without rods.

It will be important to know exactly which different elements may be present in the 3D environment, including all their specific characteristics in order to make things smoother and simpler. So, there are about 20 amino acids within a protein, and the most prevalent 5 atoms are: Carbon (C), Hydrogen (H), Oxygen (O), Nitrogen (N) and Sulfur(S) as shown in the figure 3 below. Each one of them have their size, Van der Waals radius of the sphere and color as represented in the table 1 below. The rendered 3D scene will only have 5 types of different elements, with their specific radius. Yet, there will be hundreds of thousands spheres in the proteins themselves.

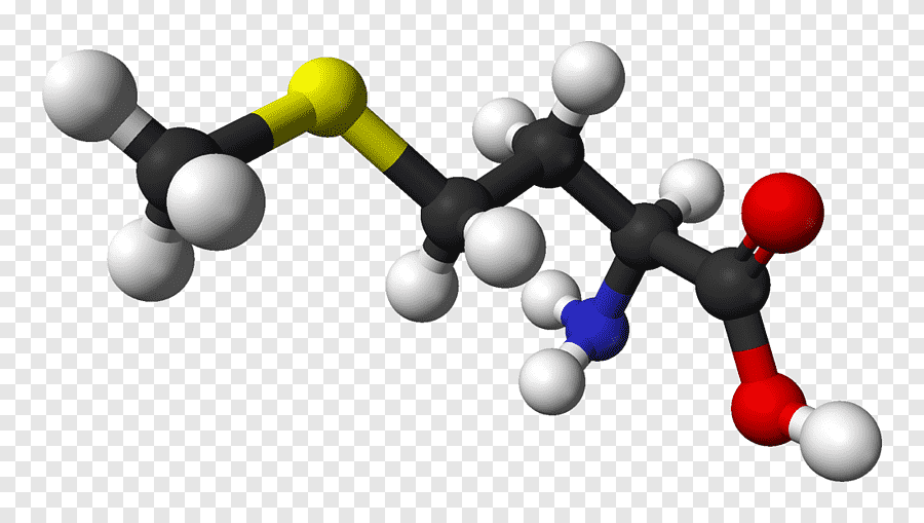

Fig. 3. Protein elements representation *(Methionine Essential amino, 2021)*.

Table 1. Elements that build a protein (Marcella Martos, 2021).

Atom Element	Chemical Symbol	Van Der Waals Radius (Å = 10^{-10} m)	Standard Color
Carbon	C	170	Gray
Oxygen	O	152	Red
Nitrogen	N	155	Purple
Sulfur	S	180	Yellow
Hydrogen	H	120	Blue

2.2 Standalone Tools

A standalone application runs entirely on the device and does not require any additional software to work, because the application contains all the logic, it does not require an internet connection or the installation of any other services, all the files will be included in the setup file itself. They use our PC resources, so generally these tools are more powerful allowing a more sophisticated studies and analysis taking place, which is handful for scientists, biologists, biochemistries, basically a shorter and more specific audience of users. As we can see in table 2, there's an example of a few of the main standalone applications for protein analysis alongside their type of image rendering (Imposters or Polygons).

Applications like Chimera (Pettersen, 2004), Chimera X (Goddard, 2018), VMD (Humphrey, 1996), Pymol (DeLano, 2002) or Yasara (Krieger, 2014) can make the use of new technologies power like *RTX* and *OptiX* providing all the visualization features needed for protein representation as ray tracing, shaders, lighting, reflections and many more, allowing a more detailed information about them. Being their strong advantages factor against the web applications that will be referred to in section 2.3. However, all this progression comes from a long way that can be marked with the arrival of QuteMol (Tarini, 2006), revealing one of the first of ambient occlusion implementations.

Table 2. Standalone tools with their respective main IBR technique.

Name	Last Update	Imposters	Polygons	Link
Chimera	2020		●	https://www.cgl.ucsf.edu/chimera/
Pymol	2021	●		https://pymol.org/2/
VMD	2014		●	https://www.ks.uiuc.edu/Research/vmd/
Chimera X	2021		●	https://www.cgl.ucsf.edu/chimerax/
Yasara	2021	●		http://www.yasara.org/index.html
QuteMol	2007	●		http://qutemol.sourceforge.net/

Table 3. Standalone tools with their respective visualization features.

Name	Shaders	Reflections	Light Source Positioning
Chimera	●	●	●
Pymol	●	●	●
VMD	●	●	●
Chimera X	●		●
Yasara	●		●
QuteMol	●	●	●

2.3 Web Tools

On the other hand, web tools are targeted for a more casual audience, they have more care about how the UI is presented and make sure if it's easy to use and understand its output and results, they have more focus on portability and guarantee that the main and essential algorithms and analysis are working.

The general steps for displaying a macromolecular structure on the web are: download file, decompress and parse, populate a data model, create geometry and render it.

A web application executes in the server side where it is hosted and mostly uses web technologies like HTML5, CSS, JavaScript, WebGL and other browser extensions. Logic and data storage are not on the client machine (there may be limited exceptions due to this factor), rather one or more servers take those architectural roles. The UI capabilities on the client machine are limited to what the web browser (including plugins) supports and the programmer generally has no ability to implement arbitrary functionality on the client, but rather must work within the capabilities supported by the client.

WebGL was developed to allow JavaScript applications running in the web browser to take advantage of OpenGL ES 2.0 (first portable mobile graphics API to expose programmable shaders in the latest generation of graphics hardware) compatible GPUs which had been specifically designed for mobile devices. While WebGL has been available for several years in Chrome and Firefox, WebGL support was only recently added to Microsoft's Internet Explorer and Apple's Safari, including iOS.

In the table 2 below, there's a list of the most reliable and used web applications for protein analysis and visualization referring to their type of image rendering (Imposters or Polygons) and the programming language used.

Table 4. Web browser tools with their respective IBR technique and programming language.

Name	Last Update	Impostors	Polygons	WebGL1	WebGL2	JavaScript	Link
HTMol	2019	●			●	●	https://htmol.tripplab.com/
Molmil	2016		●		●	●	https://pdbj.org/molmil2/
Web3DMol	2017		●		●	●	https://web3dmol.net/
JSmol	2016		●		●	●	http://jmol.sourceforge.net/
Aquaria	2015		●	●			https://aquaria.ws/
Mol* viewer	2019		●	●			https://molstar.org/viewer/
EzMol	2018		●	●		●	http://www.sbg.bio.ic.ac.uk/ezmol/
NGL Viewer	2018	●			●	●	https://nglviewer.org/
ICn3D	2019		●			●	https://www.ncbi.nlm.nih.gov/Structure/icn3d/
LiteMol	2017	●				●	http://webchemdev.ncbr.muni.cz/Litemol/

These web applications have way fewer visual features than the standalone applications and at most they provide some shaders and some management regarding the lights source positioning, as discriminated in the table 5 below.

Table 5. Web browser tools with their respective visualization features.

Name	Shaders	Light Source Positioning
HTMol	●	●
Molmil	●	●
Web3DMol		●
JSmol		●
Aquaria		
Mol* viewer	●	●
EzMol	●	●
NGL Viewer	●	●
ICn3D	●	●
LiteMol	●	●

2.3.1 HTMol

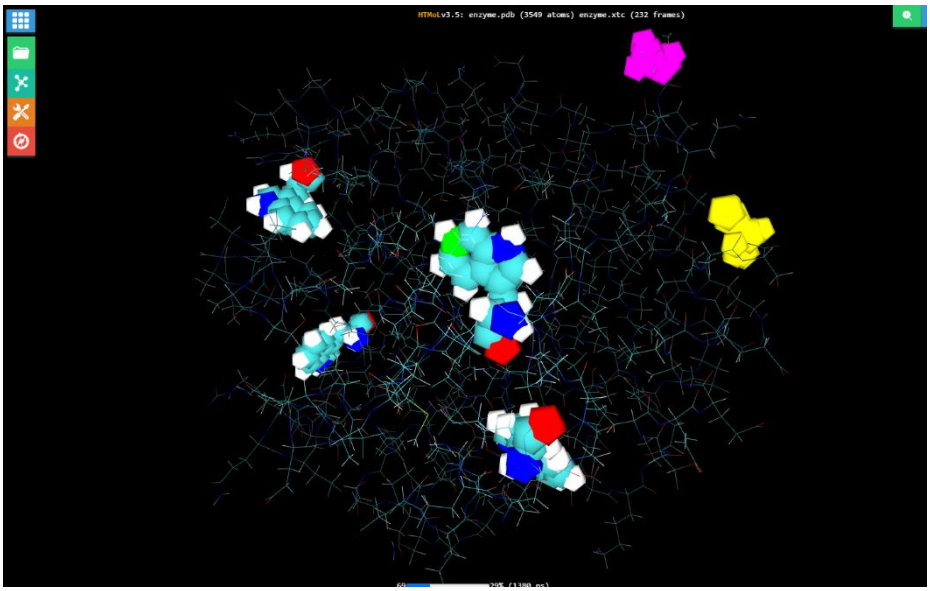

Fig. 4. HTMol Interface.

HTMoL (Carrillo-Tripp, 2018) is a safe GPU-accelerated web application specifically designed to visualize and stream molecular dynamics trajectory data in a web browser.

Focuses on the efficiency of data management in processing and sending 3D information for its visualization.
It integrates the latest web technologies and hardware acceleration in the client-side. Its architecture is a dedicated full-stack tool that integrates the recent web technologies like WebGL, NodeJS, HTML5, CSS3, allowing to execute several remote tasks through calls between the client and a web server. Primarily used by researchers, but is just as applicable to other fields, like learning and education.

Tests were performed to evaluate the effects that the use of third-party 3D libraries (plugins) have on the application's front-end performance, and three.js, a JavaScript library, was developed to create and display animated 3D computer graphics on the web browser, and also minimizes code implementation work, but a detrimental impact was found on the application's performance. Despite these not so good results, HTMoL abandoned the use of such libraries in version 3.0 anyway. Now, the code implemented was made with native primitives built-in. This strategy reduced the total weight of the application and consistently increased the rendering performance one order of magnitude in all browsers and platforms.
Since HTMoL does not depend on a third-party 3D library anymore, it is a light-weight tool reaching high frame rates. Also, the modular design of the core engine ensures that HTMoL will be adaptable to future technological improvements.

Also, a comparison was made through a variety of environments (browsers + Operative Systems) and internet explorer demonstrated to be a little slower due to its lack of support to modern technologies. With this, one weak point of web tools, and particularly in this one, is that the performance of some functionalities from the tool may vary depending on the user's browser.

2.3.2 Molmil

Fig. 5. Molmil Interface.

One of the design goals of Molmil (Bekker, 2016) was to create a molecular viewer which can produce high quality images suitable for publications. It can take advantage of GPU hardware acceleration using WebGL. It runs on a wide range of platforms such as Windows, Linux, Mac OSX, Android and iOS. On the other hand, it should also be able to scale to very large structures available in the Protein Data Bank (PDB) consisting of hundreds of thousands to millions of atoms.

By default, WebGL can only display a limited number of polygons. However, by using an extension to WebGL (OES_element_index_uint) which is available on all modern platforms, it becomes possible to efficiently render very large or highly detailed structures. Molmil uses this extension to build high quality geometry for small and medium sized structures and dynamically scale down the quality as the size of the structure increases to gigantic proportions.

To enable high quality lighting even when using polygon models of low detail for these gigantic structures, Molmil uses tuned Phong shading (reflection model) which can accurately calculate the lighting even for simplistic polygon models.

Molmil is also a light-weight and full featured viewer for the PDB, as such it can load multiple types of file formats, being flexible in that way. Users can also load these files from their local hard drive.

In contrast, it's very dependent on the user's browser, Molmil requires the browser to support WebGL. Also, to be able to load these gigantic structures, an adequate amount of memory is still required, which is often not available on smartphones, tablets and older systems.

Having this, multiple studies were done to different browsers, and what was concluded is that due to limitations of Chrome's JavaScript engine, even Chrome's 64-bits version is incapable of loading 3j3q (Atomic-level structure). Mozilla's Firefox and Apple's Safari however have no such problem with the 64-bit versions. Still, it will still take dozens of seconds to several minutes depending on the web browser, the user's hardware such as the processor and the user's internet connection since 3j3q's compressed data file is still about 40 MB large.

2.3.3 Web3DMol

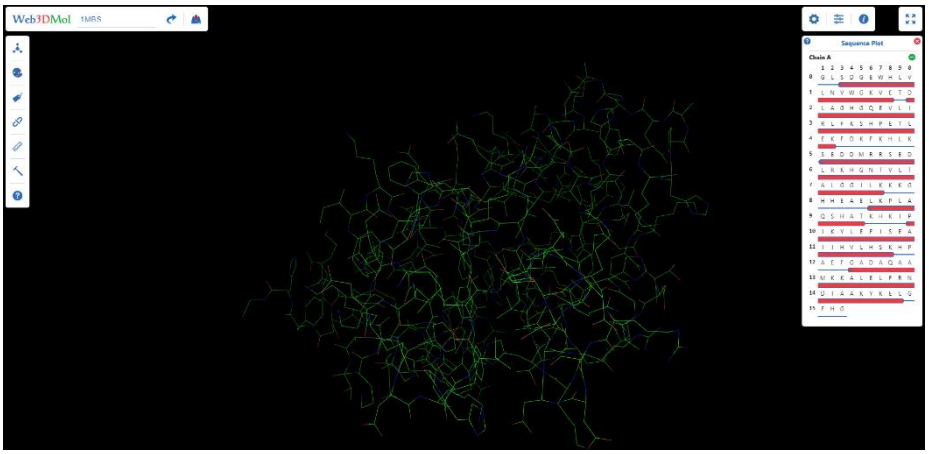

Fig. 6. Web3DMol Interface.

Web3DMol (Shi, 2017) has unique functionalities, including sequence plot, fragment segmentation, measure tool and meta-information display. Is a valuable tool for both

researchers and developers who are interested in protein structure research. It saves users the time and energy spent on installing of software or plugins and gaining familiarity with command-lines. It's helpful for visualizing protein structure data without a need to parse PDB archives (required process for displaying a macromolecular structure) and construct geometries from the beginning.

Since PDB archives record not only the 3D coordinate data but also some related information about the molecules, such as the molecular classification in biology, the structural resolution and the experimental details. With this, Web3DMol supports several types of measurement, including distance, vector angle, dihedral angle and area among atoms, which can be carried out through simple mouse interactions.

As expected, 3D modeling and rendering are both resource intensive calculations, and as a form of interpreted language, JavaScript is not good at high efficiency calculations. Therefore, when the size of a molecule becomes very large, Web3DMol sacrifices some graphical quality to maintain the efficiency. Another restriction is from web browser manufacturers, like for example, V8, the JavaScript engine in Google Chrome, has a threshold for maximum heap memory usage, so that when the number of atoms is too large, Chrome will crash. In fact, for very large molecules, RCSB PDB does not offer common PDB archives to download. JavaScript was originally designed for web page interaction, and its function libraries are not abundant, therefore, it is difficult for Web3DMol to handle PDB archives if they are in zipped format while users are manipulating the 3D structure.
The support for WebGL on mobile devices is uneven, Web3DMol runs well on some of the latest cell phones, but not very smoothly on most mobile devices at present.

2.3.4 JSmol

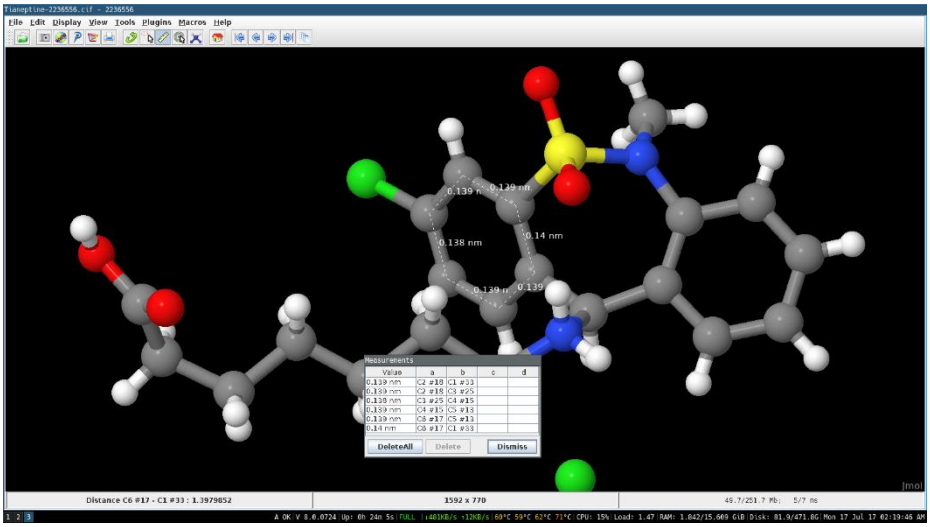

Fig. 7. JSmol Interface.

Jmol (Herraez, 2006) was developed by a community of volunteers and the most relevant feature that make it a promising tool is that it is open source, meaning that anyone can get the source code and modify it, promoting future development. Also, it is available freely on the internet and at no cost. It is written in Java, which makes it compatible with most operating systems and with all major web browsers and provides triple implementation: standalone application, applet for embedding in a web page, and development tool kit for inclusion into other Java software.

The file format is recognized automatically upon reading the file, without giving any instructions. The software has been designed to be modular so is easily expanded into new formats since the file parser is independent of the core functionality. In addition to reading molecules, it can read script files, text files with instructions or commands to be applied to the model (type of rendering, rotation, translation etc.).

Writing WebGL export drivers for Jmol is somewhat more complicated than a direct translation of Java into JavaScript, and Apple decided to not allow Java on their iOS platform and the Java browser plugin coming under fire as a possible security risk. As

a consequence, as technology advances, web applications, such as *Proteopedia*, are at risk of losing their audience.

Although Java does not run in some devices like iPhones and iPads, JavaScript does. Due to this, JSmol (Hanson, 2013) was developed, a JavaScript-only version of Jmol, and its use in *Proteopedia* (3D protein encyclopedia). A key aspect of JSmol is that it includes the full implementation of the entire set of Jmol functionalities, including file reading and writing, scripting, and rendering.

A comparison of Jmol and JSmol was made and Jmol has shown that it is more compact than JSmol. This difference is due to the fact that the Java language library is already present on the client machine, in the form of a Java plug-in, and that the Java code is in a binary format. In contrast, when a page utilizing JSmol is accessed, the required components of the Java language, translated into nonbinary JavaScript, must be downloaded in full. In both cases, however, files are cached, and subsequent page views are far faster than the first viewing. Performance with small molecules is very similar in the two versions, but clearly JavaScript does not scale at the same rate as Java. However, nowadays JavaScript lacks many of the basic features of Java.

2.3.5 LiteMol

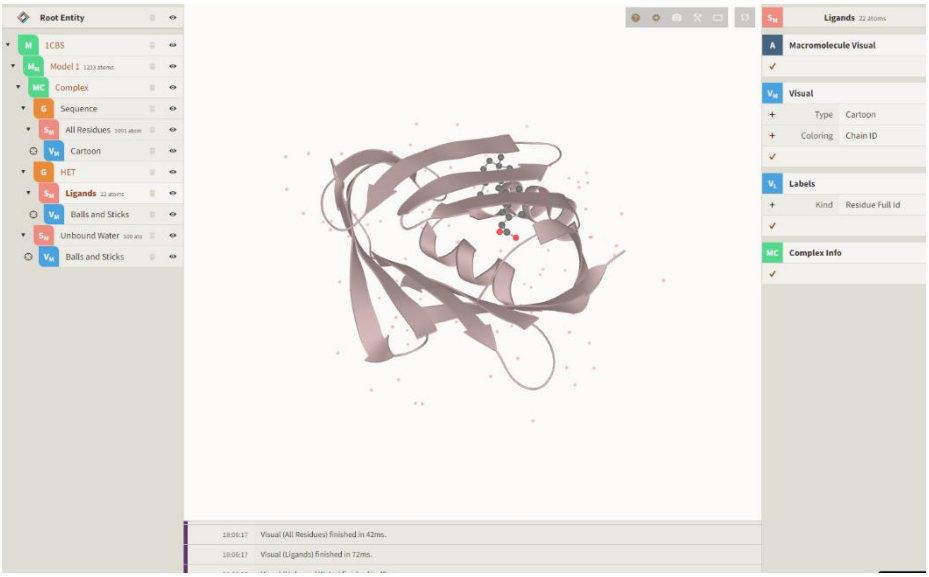

Fig. 8. LiteMol Interface.

LiteMol (Sehnal D. , 2017) focuses on three components: data delivery services, the BinaryCIF compression format (efficient binary encoding), and a new lightweight 3D molecular viewer. Together, these components enable near instant delivery and visualization of large macromolecular data sets. It works in all modern web browsers and mobile devices, and this makes macromolecular structure data available to diverse communities of users with and without structural biology expertise. Provides 3D visualization capabilities to display 3D coordinates data in standard representations and overlay them with additional annotations. Moreover, it readily displays experimental maps and has the ability to visualize low resolution structures obtained using bioimaging techniques and thus lacking atomic coordinate models. LiteMol components can be used either independently or together in a wide variety of applications including the analysis, delivery, and visualization of custom data sets.

The BinaryCIF compression format provides a uniform data storage framework for macromolecular structure data (including experimental maps and annotations), and this

removes the need for handling multiple file formats. Standard PDBx/mmCIF dictionary definitions, provided by the PDB, are used to store macromolecular models, and this facilitates straightforward adaptation of existing software to use BinaryCIF. These features make BinaryCIF an important improvement over the MMTF format, which is limited to storing coordinate information.

On the other hand, a MMTF file is much more oriented to performance and makes things much faster than using BinaryCIF, which can be a negative factor to LiteMol, not being very worried about rendering and parsing speed. NGL Viewer, mentioned in section 2.3.9 is a good example of this, because they have a different file format preference.

2.3.6 Aquaria

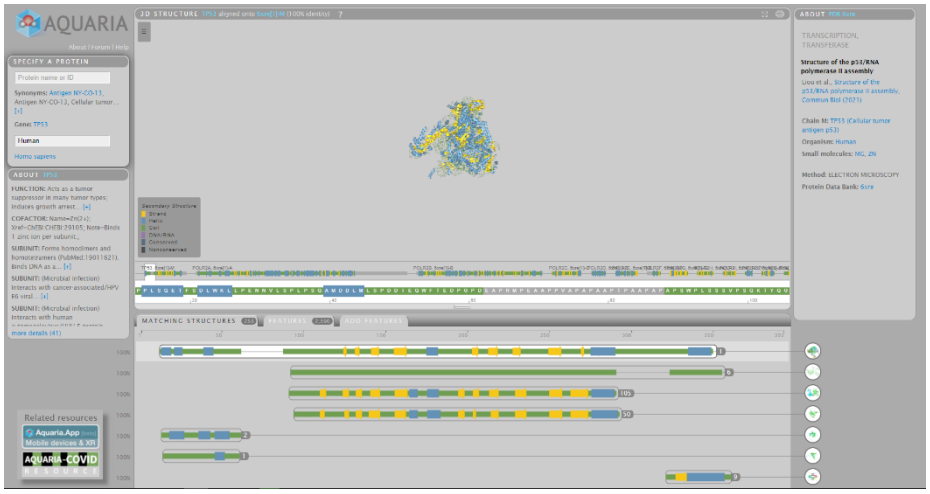

Fig. 9. Aquaria Interface.

Aquaria (O'donoghue, 2015) in contrast to most molecular graphics tools, the user interface is organized primarily by protein sequence, not structure. A user starts by specifying a protein of interest by name and organism, by identifier or by URL, then generates a concise visual summary of all related PDB structures. The related structures are grouped first by alignment to the specified sequence and second by oligomeric state.

Aquaria is designed for biologists, its user interface creates clear and useful default views that shows only the most relevant structural information tightly integrated with sequence, features and text that provide biological context. It uses a minimal set of mouse-based controls that are intuitive yet powerful. For example, its “Autofocus” feature allows exploration of large complexes by focusing on one molecule at a time. Can also be controlled via hand gestures using the Leap Motion (hand tracking software). Currently, Aquaria contains millions of precalculated sequences to structure alignments, providing a depth of sequence-to-structure information currently not available from other resources.
Although much of the Aquaria workflow could be performed using a combination of other resources, it greatly reduces the time and effort required.

Since Aquaria is more focused in organization and guaranteeing a more friendly UI, it doesn’t concern much about how the files are parsed, which type of files are being used, or any kind of high level of rendering techniques, which of course it may impact its performance. Besides, no tests were performed regarding this issue, which proves that they need to work on this particularity.

2.3.7 Mol* Viewer

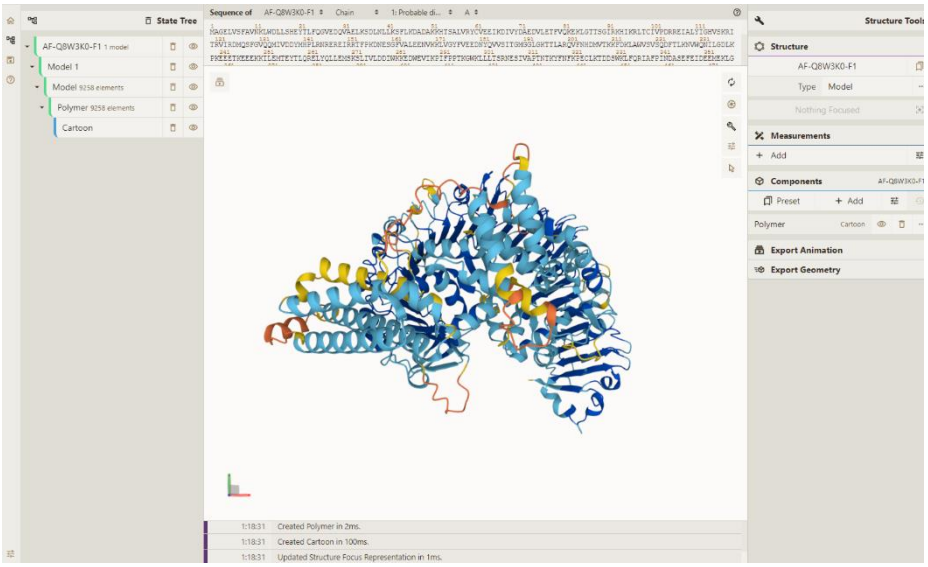

Fig. 10. Mol Star Viewer Interface.

Mol* Viewer (Sehnal D. , 2021) is a powerful web application for the visualization and analysis of molecular data. Its visualization capabilities far exceed other currently available web visualization tools, and its speed and robustness allow the fast and intuitive visualization of molecular data ranging from atomistic models from PDB or MD simulations up to hybrid models with hundreds of thousands of residues. Furthermore, offers advanced selection and a rich set of visualization models and coloring types. Being this way, the primary 3D structure viewer is used by RCSB PDB and can be easily integrated into third party services.

Also enables 3D visualization and streaming of macromolecular coordinate and experimental data, together with capabilities for displaying structure quality, functional, or biological context annotations, it can show one structure, a few and a large set of structures. Inherits many LiteMol and NGL Viewer features (mentioned below in section 2.3.9). High performance graphics and data management that allows users to simultaneously visualize up to hundreds of protein structures, stream molecular dynamics simulation trajectories, and render cell-level models.

One of the main purposes of Mol* Viewer is to enable web based molecular visualization and analyses by providing a common library for the rapid and efficient development of tools and services for the structural biology/bioinformatics enthusiasts and more casual audience.
As such, it is possible to render many types of large systems, including ribosomes, virus capsids, collections of macromolecules or MD simulation systems. It can visualize markedly larger molecular systems than other currently available web visualization tools. Due to built-in BinaryCIF, decompression support, and advanced techniques for model and volume/experimental data streaming, even large structures are interactively renderable over limited bandwidth.

Mol* Viewer offers a diverse visualization and streaming capabilities and displaying options, which is good, however, this many possibilities at the same time is not that very user friendly, at least for casual users who are still learning the application itself and the basics of protein visualization. Furthermore, this factor wasn't even tested or mentioned in the article which leads to a strong indication that they are not worried about how the UI is organized.

2.3.8 EzMol

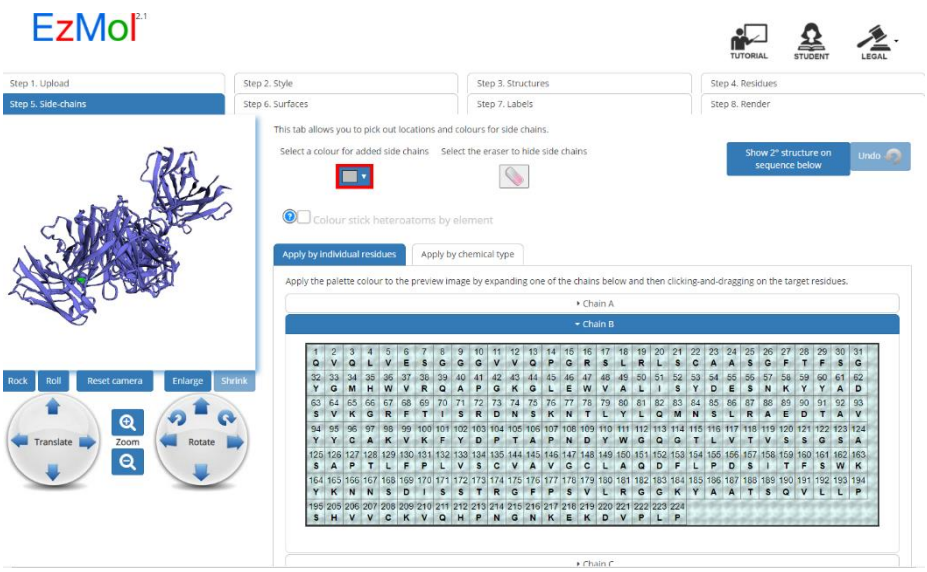

Fig. 11. EzMol Interface.

EzMol (Reynolds, 2018) focuses on guiding the user through a series of steps designed to encompass the most common needs for the visualization of protein molecules. Is intended to allow the quick rendering and coloring of molecular structures. Provides limited functionality that is nevertheless sufficient for many biological purposes and does not require the user to download software. It has applications in situations where the user has a short time to prepare images for an e-mail, presentation or lecture, or even using it dynamically to highlight areas of structures onscreen and visualize where they lie on the protein structure. Has potential for teaching, particularly for younger students who have not yet learned how to use a more complex molecular graphics program, and for the prototyping of macromolecule images.

Allows the upload of molecular structure files in PDB format to generate a customized image manipulation tool for that structure. Provides intuitive options for chain display, adjusting the color/transparency of residues, side chains and protein surfaces, and for adding labels to residues. The final adjusted protein image can then be downloaded as a high-resolution image.

Another main advantage we can get by using EzMol goes around being a cross-browser compatible on multiple platforms, requires no text input and addresses the most common molecular visualization requirements.

EzMol features were put in comparison with those of JSMol, PyMol and UCSF Chimera. As expected, these programs have greater capability, but there is a trade-off between usability and control. To spotlight EzMol, for instance to apply a single application of color to a contiguous set of residues can be done using three clicks in EzMol: two to open the palette and select a color and one to highlight the residues. Also has the benefit of being able to see the color being applied in real time as the corresponding cells are highlighted.

2.3.9 NGL Viewer

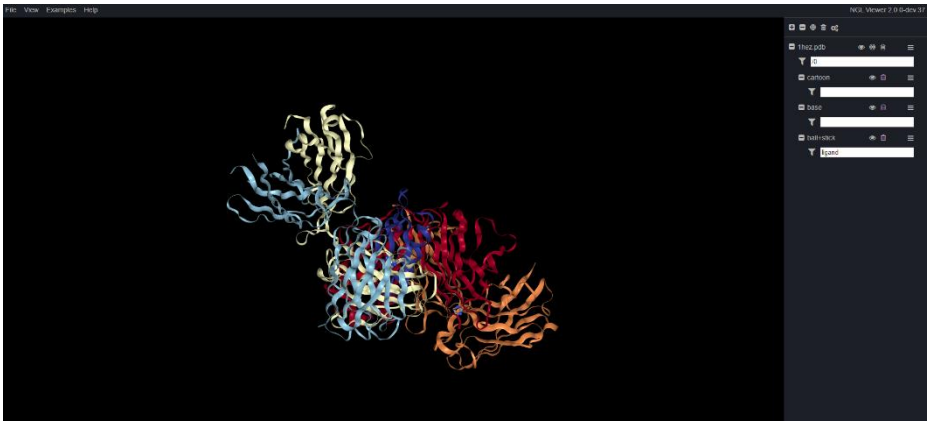

Fig. 12. NGL Viewer Interface.

NGL Viewer (Rose, 2015) is a memory efficient representation for molecular data, as well as a reference implementation for decoding and parsing MMTF files (compressed binary data) in JavaScript, speeding up the download time. More known for the developments in the NGL data model and the use of MMTF that significantly reduce the peak memory consumption. Due to its high performance, NGL Viewer has been selected as the default 3D viewer for the RCSB PDB website, a dedicated viewer for

mobile devices. NGL Viewer downloads MMTF files that contain the asymmetric unit and transformations to create biological assemblies.
It includes all necessary bonding information for rendering. The NGL data model uses a flat, columnar layout with a single JavaScript *TypedArray* for each property. This allows the parsed MMTF data to be reused or copied in blocks to the NGL data model. Several tests and comparisons were done in different file formats like MMTF and mmCIF and the results demonstrated that MMTF files can be parsed about 10 times faster than mmCIF, and one factor that plays a role in that difference is that a MMTF file is one third the size of a zipped mmCIF file. This file format preference, compared with other web tools, may provide less 3D visualization and representation capabilities. Another use case of NGL is the interactive download and rendering of structures using the plugin in Jupyter Notebooks. The fast download and rendering allow the user browse through a set of structures without any noticeable delay in loading and rendering structures.

For rendering, WebGL is efficiently used by preparing the data such that the number of calls to the WebGL API does not grow with the size of the macromolecules. WebGL API calls have a fixed time cost, therefore molecular representations are grouped and rendered together as opposed to rendering each atom individually. By that, the substantial overhead every WebGL API call adds is avoided. Impostors are also used to render cylinders and hyperboloids.

2.3.10 iCn3D (I-see-in-3D)

Fig. 13. iCn3D Interface.

iCn3D (Wang, 2020) can simultaneously show 3D structure, 2D molecular contacts and 1D protein and nucleotide sequences through an integrated sequence/annotation browser. Pre-defined and arbitrary molecular features can be selected in any of the 1D/2D/3D windows as sets of residues and these selections are synchronized dynamically in all displays. Biological annotations such as protein domains, single nucleotide variations can be shown as tracks in the 1D sequence/annotation browser. iCn3D follows the FAIR (Findable, Accessible, Interoperable and Reusable) guiding principles, for this the JavaScript code is oriented to be reusable as annotations could also be retrieved by other tools.

Third party annotations, such as mutations from *ClinVar* (public archive that stores information about genomic variations), functional sites, interaction interfaces, structural or conserved domains, and their simultaneous visualization in 1D/2D/3D can provide useful and compelling evidence relating sequence, structure and function. More detailed genome level annotations could be linked to the structures in the future. For example, protein sequences could be shown together with “chromosome”, “gene”, “intron” and “exon” in a genome browser. In the future, the developers of iCn3Dare thinking of adopting an emerging 3D technology such as virtual reality.

This unique visualization features provided by this tool may bring, of course, some performance issues in its rendering process or at least not that high performance wise compared side to side, for instance, with NGL Viewer.

3 Methodology

A web browser application was created based on billboards and global illumination, programmed in WebGL2 and several techniques were used to reduce the memory usage, load big data structures, increase performance and provide a fairly decent friendly graphics.

3.1 Techniques, Mechanisms and Data Structures

There was a need for special care during the code implementation like trying to reuse objects such as geometries, materials and textures to avoid the creation of unnecessary objects, for instance, in a render loop. A data structure called *BufferGeometry* was mainly used, which is a representation of a mesh, line or a point of geometry from the *three.js* library that is an API used to create and display animated 3D computer graphics in a web browser using WebGL.

JavaScript *TypedArrays* were also used since it is a good option because they are objects that provide a mechanism for reading and writing raw binary data in memory buffers, they grow and shrink dynamically and can have any value, so they are fast. If each property of the data model is a *TypedArray*, it can allow the parsed data to be reused or copied in blocks, which can lead to a reduction of the peak memory consumption. A parser and a loader of PDB files was created in order to obtain the information of the atoms that will be represented as objects in the 3D scenes, allowing as well memory reuse and avoiding duplicating data. Utilization of common text-based 3D data formats was avoided, such as Wavefront OBJ or COLLADA, for asset delivery.

3.2 Rayground

During the development of the web application the framework Rayground (Vitsas N. , 2020) was explored and used for quick prototyping of algorithms based on ray tracing algorithms, it works in any platform with WebGL2. It was developed based on the studies done about ray tracing in the web, called *WebRays* (Vitsas N. , 2021). Its main purpose is to help develop and test modules that showcase a particular method or technique. The graphical UI is designed to have two discrete parts, the preview window and the shader editor. Its visual feedback is interactively provided in the WebGL rendering context of the preview canvas, while the user performs live source code modifications. So, this framework, was really helpful for the ray tracing implementation, which brings, automatically, characteristics like shadows, ambient occlusion and many others.

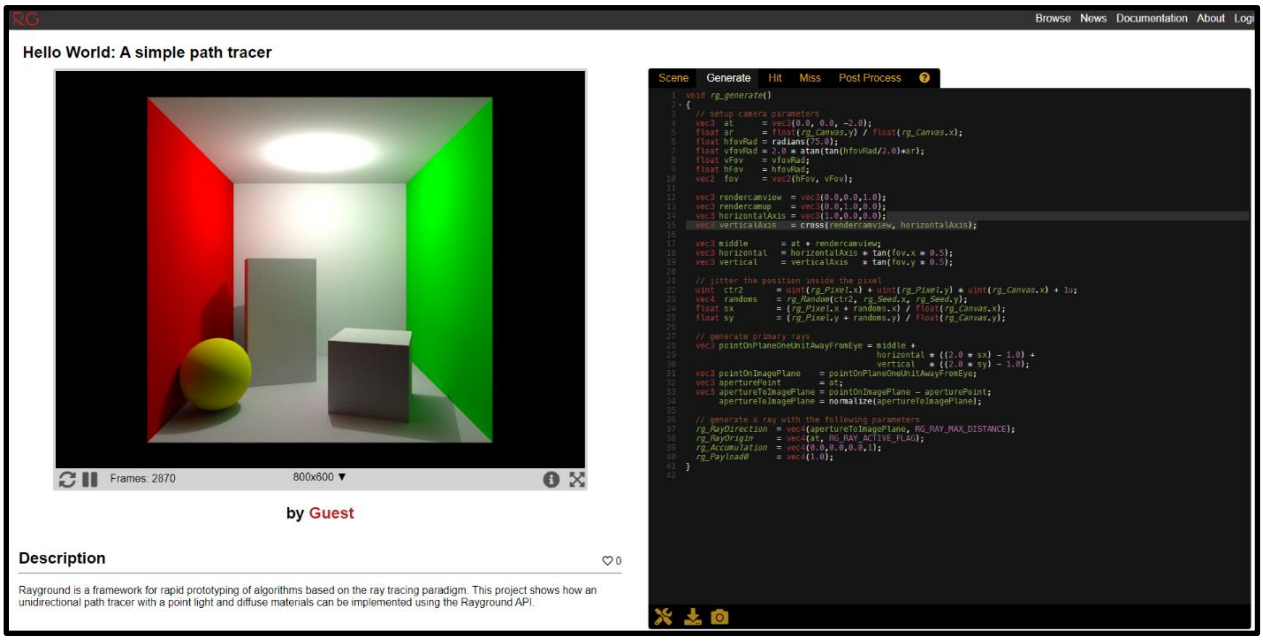

Fig. 14. Screenshot from the web interface of Rayground showing the code and its respective renderization example.

3.3 Implementation

The approach of a hybrid implementation was the main goal, consisting in placing impostors/billboards while the objects, in the scene 3D, are distant and applying the simple GI algorithm (similar ray tracing in terms of visualization) when the objects are near to the camera. This approach was heavily considered, because ray tracing is a slow process and it's not bearable to have it working for all situations in a 3D scene with a lot of information going on simultaneously.

The objects in the 3D scene were only composed by spheres, so it was just needed the coordinates (x,y,z) of the sphere's center, type of element, radius and color. In the end, it was predicted that 7 bytes should be enough for each atom, since 1 byte will be for the element type and 3 x 2 bytes (3 x 16 bits) for the coordinates. A special attention and care were done to the memory allocation, making sure that no memory is wasted and ultimately load the highest possible number of atoms with the least possible memory.

3.3.1 Ambient Occlusion and Simple GI

Before explaining the simple GI algorithm, it is needed to give a little overview of what Ambient Occlusion is, there is a similar demonstration of it in the figure 18 below. So, it gathers the light from everywhere around, but in a very simplified mode, and thus all the geometry in the scene is blocking the light arriving from everywhere to that point. In the case of ambient occlusion, the concept of shadow doesn't exactly exist, because its more occlusion than anything. If a point is completely surrounded by dense geometry, then the point is going to be occluded and no light arrives to it so it will be dark. If a point has no geometry above it, it will be white because all the light arrives on it.

Since Ambient Occlusion is the result of simplifying the rendering equation that describes the light interaction, the idea was to approximate (or fake) global illumination to a very small 3D scene. Mesh vertices become the sampling points of the GI, they are light emitters and receivers. The irradiance emission of a vertex is simulated by its color. First, all vertices are black. To simulate the first bounce of light, each sampling point (vertex) is calculated by how much light arrives from the emitters. This light

gathering process is traditionally done by ray casting the hemisphere around the normal to the surface in each point. However, for this to be fast, the scene is rendered with a camera in the vertex position and oriented in the direction of the normal. It's used a field of view as close to 180 degrees as possible and the color of the pixels is accumulated in the resulting frame buffer to estimate the incoming radiance [a rendering size of 16*16 simulates a total of 256 rays]. This accumulated color will be the irradiance of this vertex for the next pass, repeating the process for each vertex completes the simulation of the first bounce of light. Repeating the process allows to simulate more bounces of light. This method can take as many lights as desired.

This method is applied to every sphere in the scene, individually. To remember, all of this is only applied when the camera is close to the protein.

```
if (bounces === 3){ return;}

let object = sceneSpheres.children[2 + index];
const geometry = object.geometry;
const attributes = geometry.attributes;
const positions = attributes.position.array;
const normals = attributes.normal.array;

if (attributes.color === undefined) { const colors = new Float32Array(positions.length);
  geometry.setAttribute("color",new THREE.BufferAttribute(colors, 3).setUsage(THREE.DynamicDrawUsage));}

const colors = attributes.color.array;
const startVertex = currentVertex;
const totalVertex = positions.length / 3;

for (let i = 0; i < verticeCounter; i++) {
  if (currentVertex >= totalVertex) break;

  position.fromArray(positions, currentVertex * 3);
  position.applyMatrix4(object.matrixWorld);
  normal.fromArray(normals, currentVertex * 3);
  normal.applyMatrix3(normalMatrix.getNormalMatrix(object.matrixWorld)).normalize();

  camera2.position.copy(position);
  camera2.lookAt(position.add(normal));
  renderer.setRenderTarget(rt);
  renderer.render(clone, camera2);
  renderer.readRenderTargetPixels(rt, 0, 0, SIZE, SIZE, buffer);

  color[0] = 0; color[1] = 0; color[2] = 0;

  for (let k = 0, kl = buffer.length; k < kl; k += 4) {
    color[0] += buffer[k + 0];
    color[1] += buffer[k + 1];
    color[2] += buffer[k + 2];}

  colors[currentVertex * 3 + 0] = color[0] / (SIZE2 * 255);
  colors[currentVertex * 3 + 1] = color[1] / (SIZE2 * 255);
  colors[currentVertex * 3 + 2] = color[2] / (SIZE2 * 255);
  currentVertex++;
}
```

Fig. 15. Simple GI Implementation Part 1.

```
  attributes.color.updateRange.offset = startVertex * 3;
  attributes.color.updateRange.count = (currentVertex - startVertex) * 3;
  attributes.color.needsUpdate = true;

  if (currentVertex >= totalVertex) {
    clone = sceneSpheres.clone();
    clone.autoUpdate = false;

    currentVertex = 0;
    index++;
    if (index == end) {
      index = start;
      bounces++;
    }
  }
  requestAnimationFrame(compute);
}
requestAnimationFrame(compute);
```

Fig. 16. Simple GI Implementation part 2.

3.3.2 Impostors/Billboards

If certain spheres from the protein are far enough from the camera, the sphere is automatically replaced by a billboard, a 2D figure that represents a screenshot from the specific atom with its respective color. This in order to improve or maintain performance and have less vertex possible to calculate in real time. This process is done dynamically and is responsive, the exchange is always done whenever the user zooms in, zooms out or rotates the 3D scene.

4 Results and Discussion

The main objective of this implementation is to have an interactive 3D scene while an algorithm that simulates similar visual effects to raytracing is being executed in real time, and some protein customization along with it. While in Rayground, that was mentioned in the section earlier, its raytracing is implemented in the browser, however it does not let the user to interact in the view and is just a static image, at least until the final image is not rendered.

The data sample that was examined, consisted in proteins with different numbers of atoms in order to compare each one of them, more essentially with different magnitude order. Most proteins can have between 1 to 100 thousand atoms.
The application was tested in Chrome, Mozilla Firefox and Microsoft Edge, there was not much big of a difference between the 3, in terms of FPS.

The main metrics that were taken into consideration in order to compare the implementation with the Rayground framework, some algorithm parameters and variances were:

- Percentage of CPU usage,
- Percentage of GPU usage,
- Percentage of memory usage,
- Amount of FPS,
- Response time in Milliseconds (MS),
- Total MBytes (MB) of allocated memory.

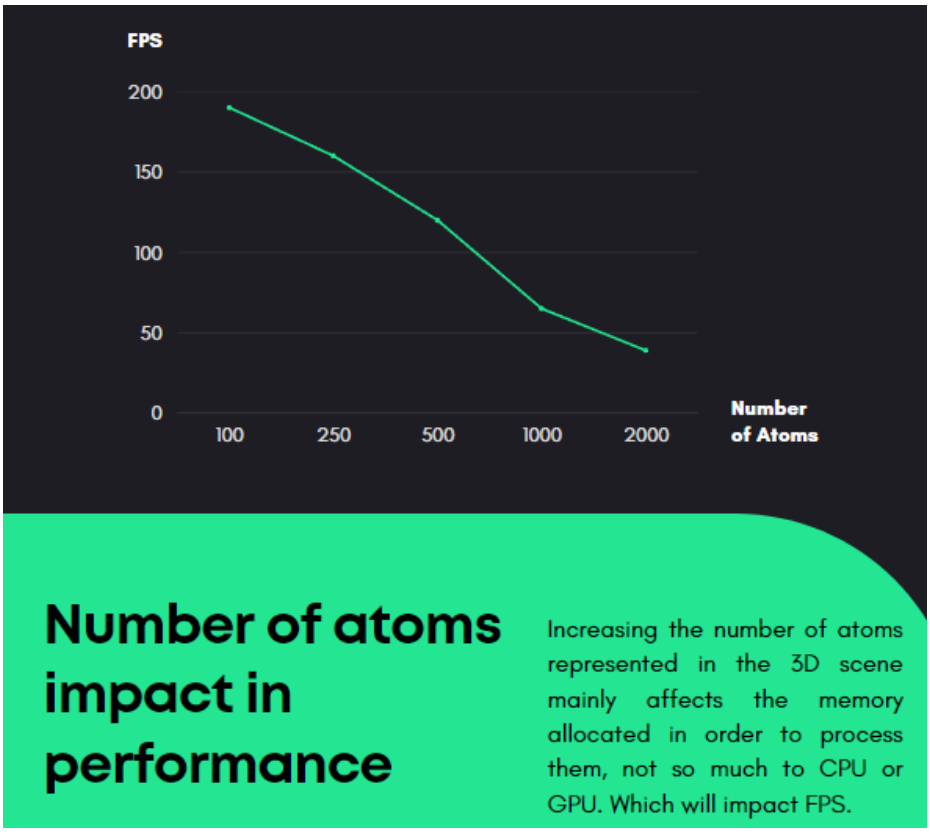

Fig. 17. Number of atoms impact in performance.

Different proteins size were put into testing and as expected when it comes to 2000 plus atoms the fps drop to under 50 as shown in the figure above. And in the table 6 below we will have a more detailed information about the consequences into having a bigger protein size.

Table 6. Number of atoms increase and its impact into the metrics defined before.

Number of atoms	FPS	CPU %	GPU %	Memory (MB)
100	190	15	9	550
250	160	15.1	11	700
500	125	15.3	15	945
1000	65	15.5	19	1170
2000	38	16	24	1420

The Table 6 above shows the variation of FPS, CPU%, GPU% and Memory (MB) accordingly the number of atoms. So, as we can see CPU and GPU are not affected that much, but memory is significantly consumed. Although the FPS are massively impacted by proteins size.

In the figures below we can observe the preview of a protein of 500 atoms and consequently 1000 atoms preview one.

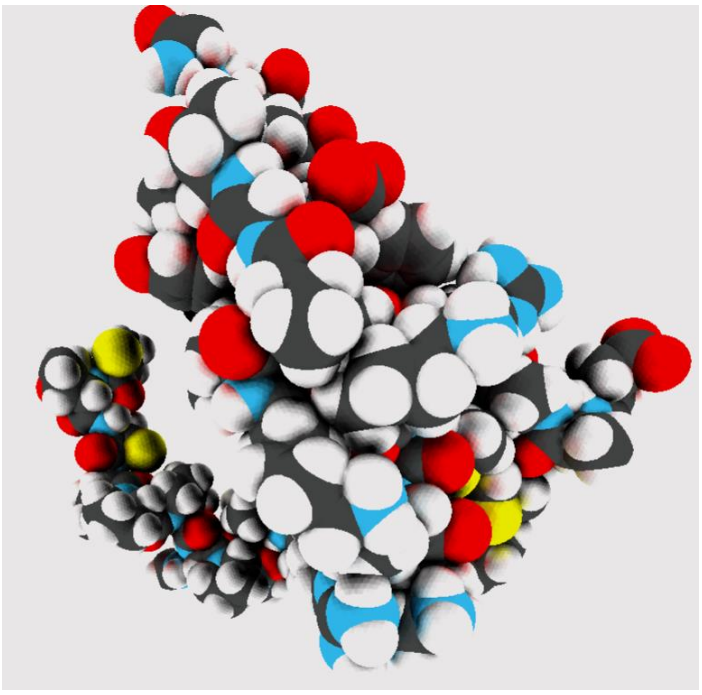

Fig. 18. 1000 atoms preview (Protein: 1d2b).

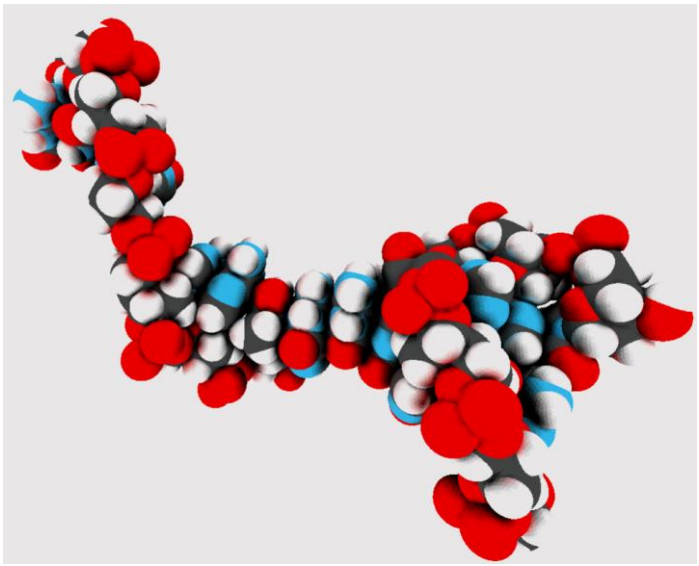

Fig. 19. 500 atoms preview (Protein: 103d).

The most relevant comparison parameters were the FOV (field of view of the camera), SIZE (size of the auxiliar rendering camera target, that is crucial for the GI algorithm) and Detail of each atom.

- SIZE impacted directly the velocity of the Simple GI algorithm.
- FOV did have an impact in performance, but very little, since it only determines how close is the reflection of the color of the vertex that corresponds to the medium color calculated from what's is in the surroundings. However, the more it can be seen, the more the computer has to do to render in those objects.
- The more atoms are detailed, more vertex it will have so it increases greatly the algorithm processing time.

In the figure 17 we can see an overview of the 3 parameters, described above, compared to each other.

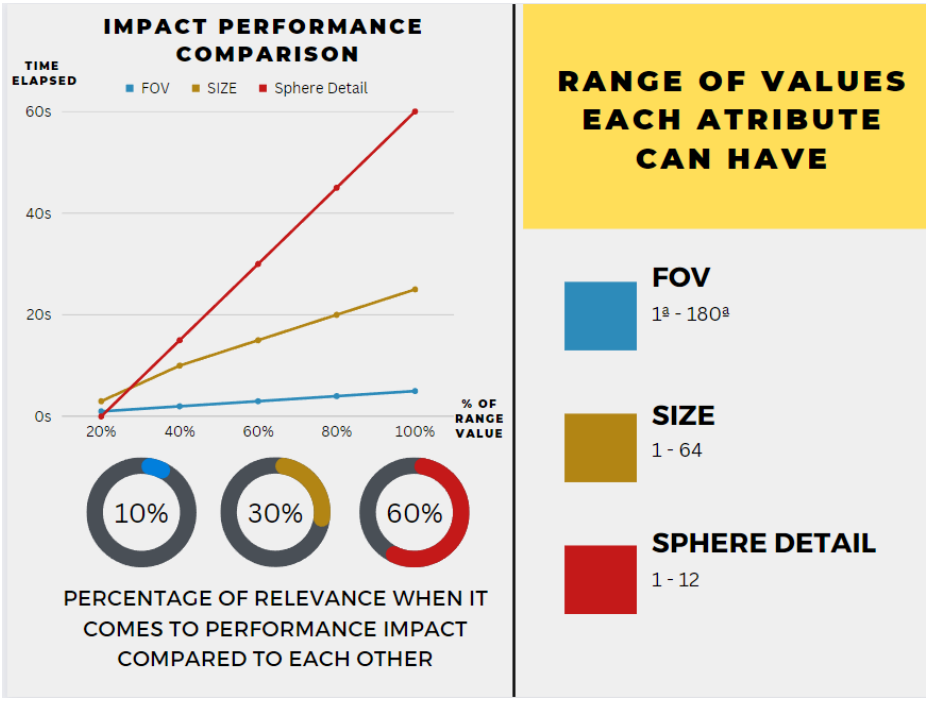

Fig. 20. Impact Performance Comparison.

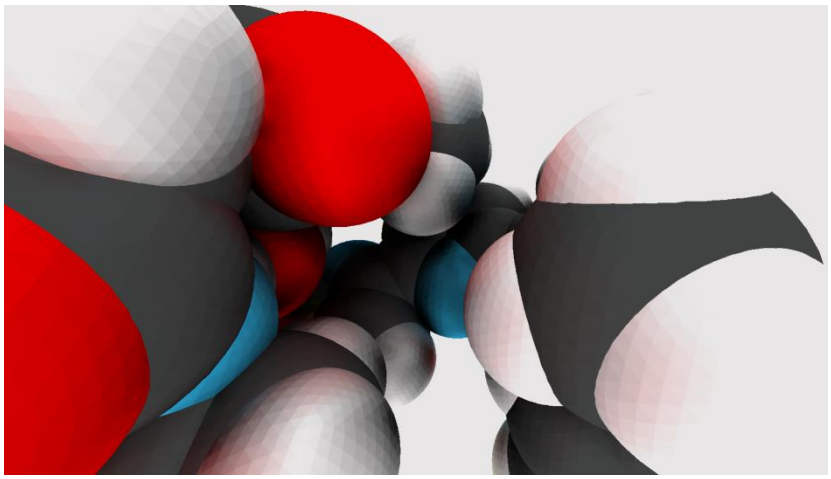

Fig. 21. Shading Similar to Ambient Occlusion.

Also, different proteins were placed into test that have different compositions between each other and of course with a great different in terms of quantity of atoms that composed them.

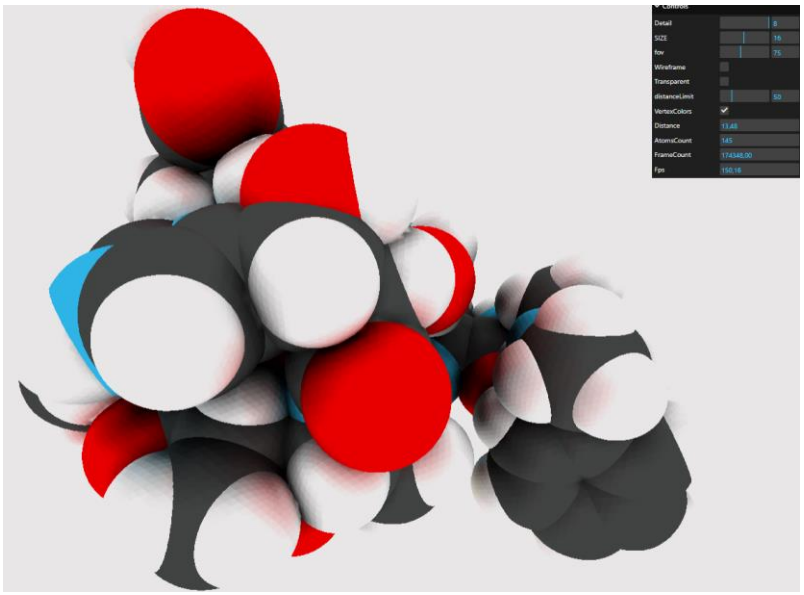

Fig. 22. Ideal Parameters - FOV:75ª, Size:16, Detail:8.

After some tests the parameters that looked closer to raytracing was the atom level of detail placed around between 8-11, SIZE to 32 or 64(preferable to 32 because of performance) and FOV around 65ª - 70ª, giving us fairly decent color reflection as we can observe at figure 19 and 20.

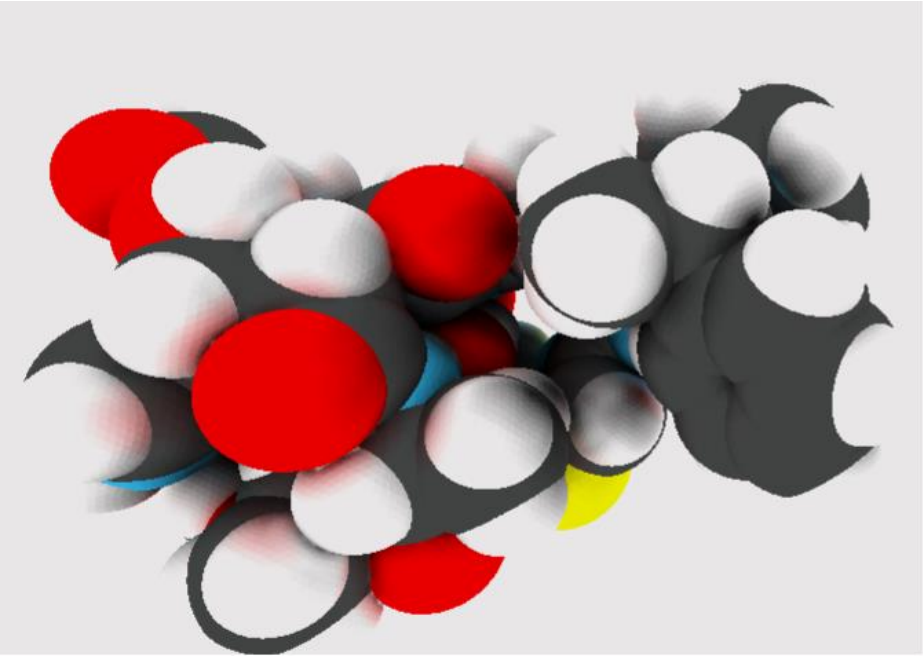

Fig. 23. Ideal Parameters another perspective.

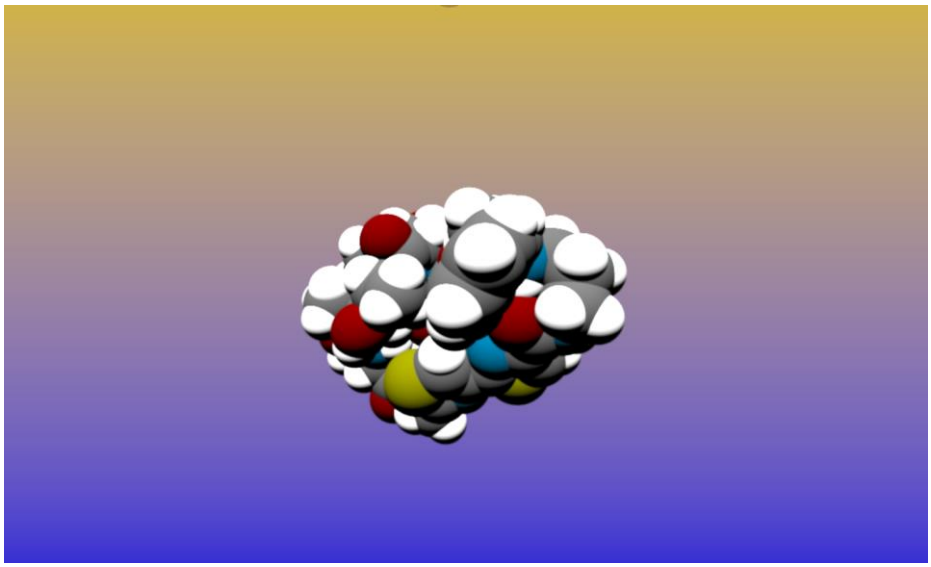

Fig. 24. Rayground raytracing example.

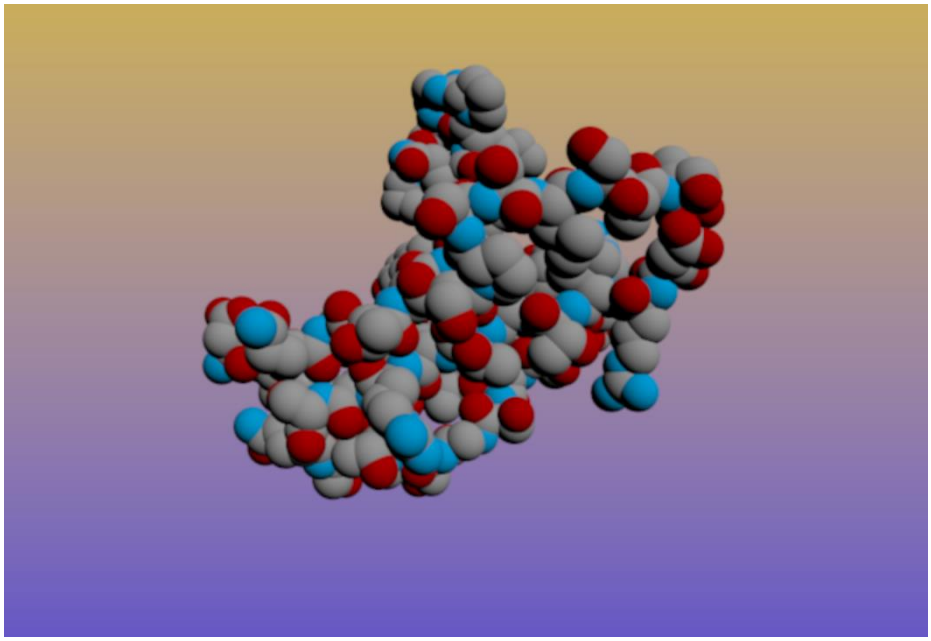

Fig. 25. Another Protein presented in Rayground Framework.

To remember, in Rayground the image is static, just the first frame is rendered, there is no interaction or GUI that the user can use to manipulate the camera of the scene. Also, the visual effect is not that great as we can see in figure 22 and 21 above.

Fig. 26. Low Atom Detail Demonstration – Detail:2, FOV: 70ª, SIZE:32.

In the figure 23 situation its notable how visual impact, the atom detail has when it goes to very low values.

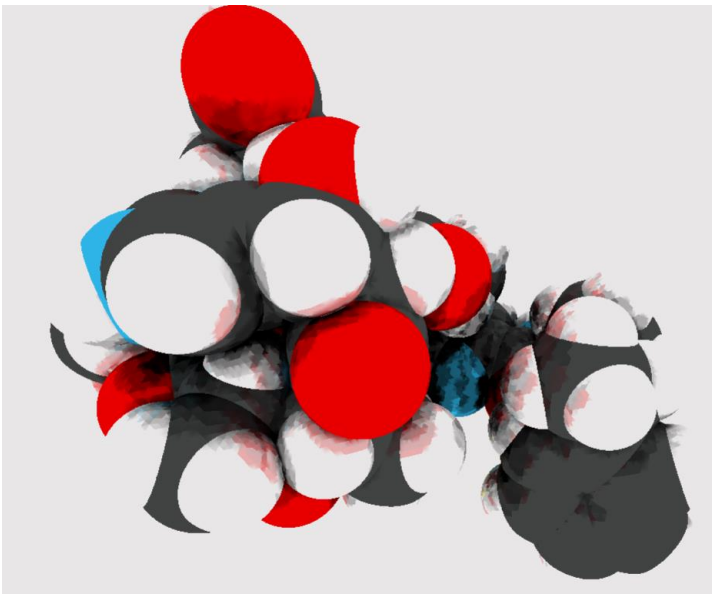

Fig. 27. Low Size Demonstration – Detail:8, FOV: 70ª, SIZE:2.

Here, with very low size we can start to see some kind of stains in the points where should be color reflections as demonstrated at figure 24.

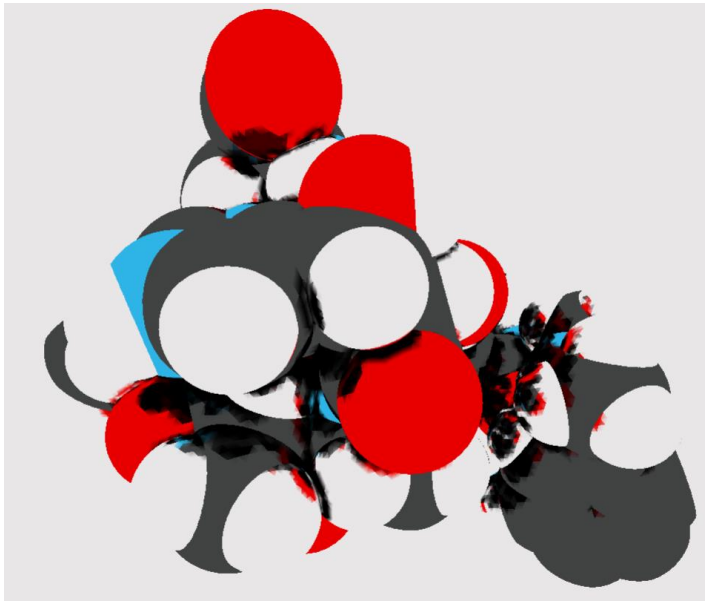

Fig. 28. Low FOV Demonstration – Detail:8, FOV: 5ª, SIZE:32.

With very low FOV, it means the reflection point has a very tiny gap resulting into those effects as shown in figure 25.

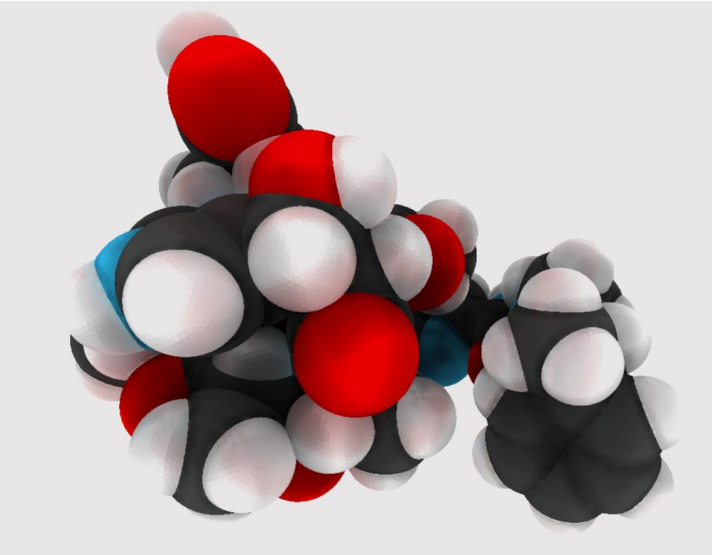

Fig. 29. High FOV Demonstration – Detail:8, FOV: 160ª, SIZE:32.

Instead, very high FOV, it means the reflection point has wide spread spread view resulting into those a more blur visual effect as ilustrated in figure 26.

5 Conclusion

Considering this is a browser application with certain characteristics that currently are still not very refined, there were difficulties and challenges making performance and efficiency going right. In the middle of the thesis, it was verified that ray tracing really takes a massive fraction from the computers resources and an alternative had to be chosen in order to make a balance between good graphics/visualization and performance, so the simple GI algorithm was picked and made a perfect combination, because now it was possible to load hundreds and thousands of atoms, since that was nearly impossible with raytracing even in close range to the camera. This technique has very similar lighting visual effects and results to ambient occlusion without resorting to full global illumination techniques.

Three.js was really handful and made a big part of the implementation, this third-party supportive library and its mechanisms offers support to massive different scenarios and situations, it was game changing.

The results and tests are a little bit incomplete since I did not have the time for it, like using profiling tools to test in multiple browsers etc. Plus, some optimization tools from WebGL2 were not implements like: *Uniform Buffer Objects*, *Texture arrays* and *samplers*. Would be interesting to see those features working into this project.

This method brought some more realism to the 3D scene and a more detailed overview of the atoms which could be beneficial for analysis and studies led by students and scientists. As a future work this method could be improved by a lot, for instance, calculate how many milliseconds the algorithm can spend in the lifetime of a frame, use all those milliseconds optimally and at the same time make the frame reach 30 FPS (1000ms/30ms) and for each group of 32 vertex, check how many times it has elapsed since the beginning of the frame and continue to update even more vertex and more sphere until we get to that gap.

Another alternative is to implement something similar to masking as presented in this article (Zadvornykh, 2016). It will bring a very innovative and interesting perspective visually.

6 References

Bekker, G. (2016). Molmil: a molecular viewer for the PDB and beyond. *Journal of cheminformatics*, 8(1), 1-5.

Berman, H. (2003). Announcing the worldwide protein data bank. *Nature Structural & Molecular Biology*, 10(12), 980-980.

Botzki, A. (2021). *Pdb file format*. Retrieved from VIB: https://elearning.bits.vib.be/courses/protein-structure-analysis/lessons/introduction/topic/pdb-file-format/

Carrillo-Tripp, M. (2018). HTMoL: full-stack solution for remote access, visualization, and analysis of molecular dynamics trajectory data. *Journal of computer-aided molecular design*, 32(8), 869-876.

Christiansen, K. (2005). The use of Imposters in Interactive 3D Graphics Systems. *Department of Mathematicsand Computing Science Rijksuniversiteit Groningen Blauwborgje*, 3.

DeLano, W. (2002). Pymol: An open-source molecular graphics tool. *CCP4 Newsletter on protein crystallography*, 40(1), 82-92.

Goddard, T. D. (2018). UCSF ChimeraX: Meeting modern challenges in visualization and analysis. *Protein Science*, 27(1), 14-25.

Hanson, R. (2013). JSmol and the next-generation web-based representation of 3D molecular structure as applied to proteopedia. *Israel Journal of Chemistry*, 53(3-4), 207-216.

Herraez, A. (2006). Biomolecules in the computer: Jmol to the rescue. *Biochemistry and Molecular Biology Education*, 34(4), 255-261.

Humphrey, W. (1996). VMD: visual molecular dynamics. *Journal of molecular graphics*, 14(1), 33-38.

Kocincová, L. (2017). Comparative visualization of protein secondary structures. *BMC bioinformatics*, 18(2), 1-12.

Krieger, E. (2014). YASARA View—molecular graphics for all devices—from smartphones to workstations. *Bioinformatics*, 30(20), 2981-2982.

Marcella Martos, M. T. (2021). *Protein Art... and What Proteins Really Look Like*. Retrieved from ASU - Ask A Biologist: https://askabiologist.asu.edu/venom/protein-art

Mary, M. (2004). Space-Filling Model. *Encyclopedia of Biological Chemistry*.

Methionine Essential amino. (2021). Retrieved from Png Egg: https://www.pngegg.com/en/png-nbuvx

O'donoghue, S. (2015). Aquaria: simplifying discovery and insight from protein structures. *Nature methods*, 12(2), 98-99.

Pettersen, E. (2004). UCSF Chimera—a visualization system for exploratory research and analysis. *Journal of computational chemistry*, 25(13), 1605-1612.

Pienaar, J. (2013). JSWhiz: Static analysis for JavaScript memory leaks. *Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, pp. 1-11.

Quilez, I. (2005). Per vertex ambient occlusion. *Simple Global Illumination*, p. 4.

Reynolds, C. (2018). EzMol: a web server wizard for the rapid visualization and image production of protein and nucleic acid structures. *Journal of molecular biology*, 430(15), 2244-2248.

Rose, A. (2015). NGL Viewer: a web application for molecular visualization. *Nucleic acids research, 43(W1)*, W576-W579.

Sehnal, D. (2017). LiteMol suite: interactive web-based visualization of large-scale macromolecular structure data. *Nature methods*, 14(12), 1121-1122.

Sehnal, D. (2021). Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures. *Nucleic Acids Research.*

Shi, M. (2017). Web3DMol: interactive protein structure visualization based on WebGL. *Nucleic acids research*, 45(W1), W523-W527.

Tarini, M. (2006). Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE transactions on visualization and computer graphics*, 12(5), 1237-1244.

Vitsas, N. (2020). Rayground: An Online Educational Tool for Ray Tracing. *Eurographics*, pp. 1-8.

Vitsas, N. (2021). WebRays: Ray tracing on the web. *Ray Tracing Gems II*, pp. 281-299.

Wang, J. (2020). iCn3D, a web-based 3D viewer for sharing 1D/2D/3D representations of biomolecular structures. *Bioinformatics*, 36(1), 131-135.

Zadvornykh, S. (2016, 11 9). *WebGL Masking & Composition*. Retrieved from Medium: https://medium.com/@Zadvorsky/webgl-masking-composition-75b82dd4cdfd