Fast Photorealistic Rendering of Protein Representations

Nelson Ferreira
Departamento de Engenharia
Informática
Instituto Superior Técnico
Lisbon
nelson.95.oliveira@gmail.com

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

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.

Figure 1: Example of PDB File format.

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).

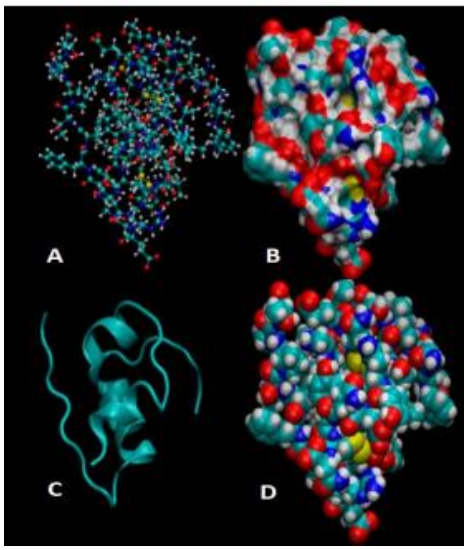

Figure 2: Protein Representations.

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 hardwood 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.

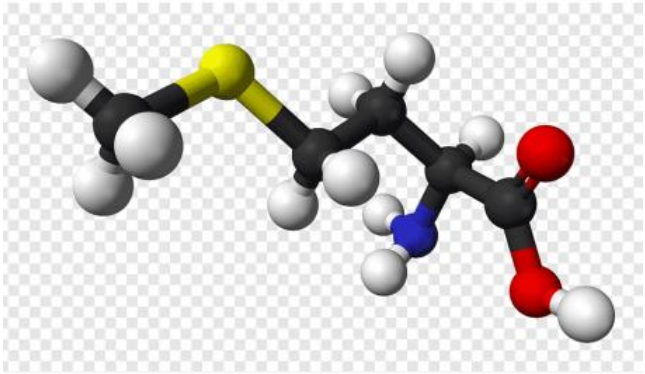

Figure 3: Proteins Elements.

Table 1. Elements that build a protein.

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.

2.2 Standalone 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.
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.

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.jslibrary 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, wasreally helpful for the ray tracing implementation, which brings, automatically, characteristics like shadows, ambient occlusion and many others.

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.

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.

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.

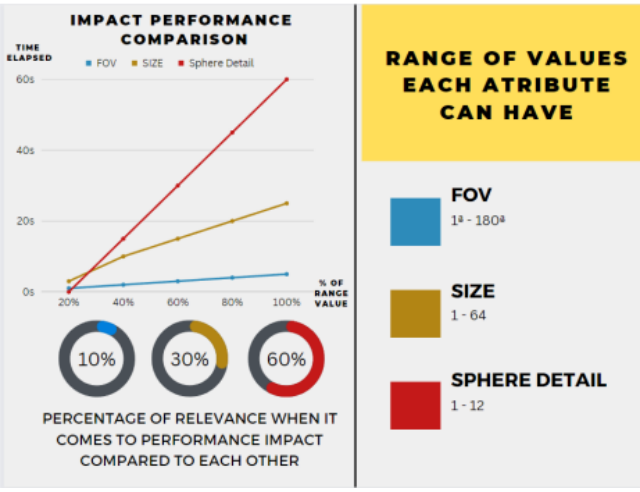

Figure 4: Impact Performance Comparison.

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.

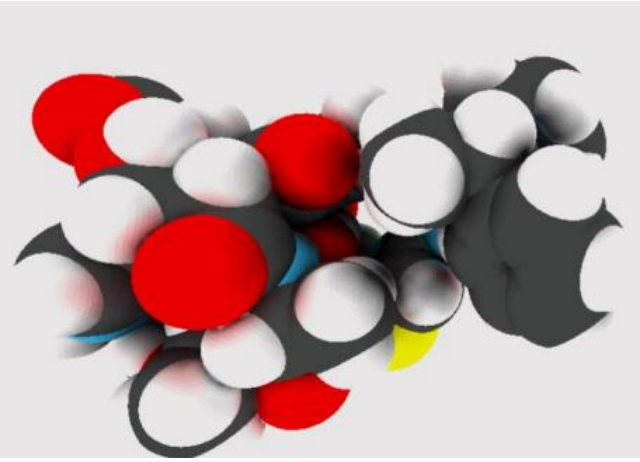

Figure 5: Ideal Parameters Result.

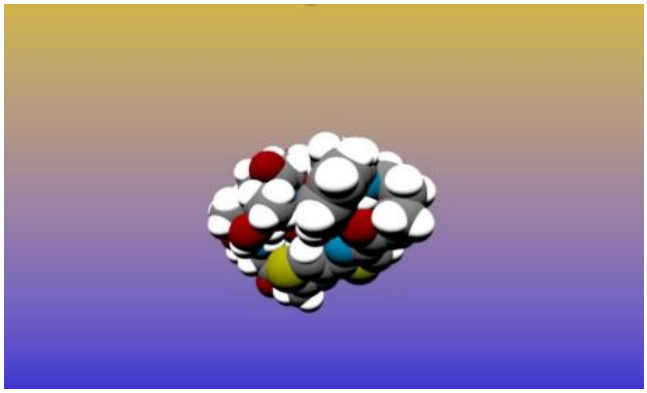

Figure 6: Rayground Raytracing example.

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 the figure above.

Figure 7: Low atom detail.

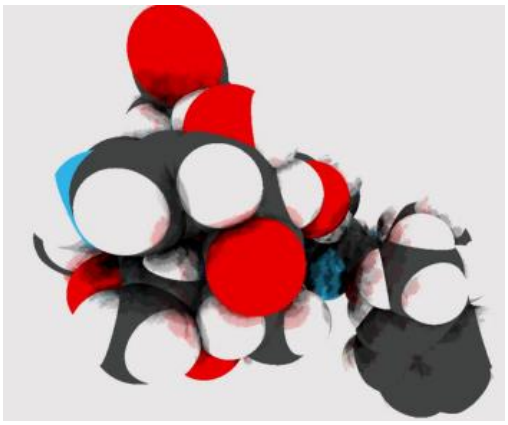

Figure 8: Low size.

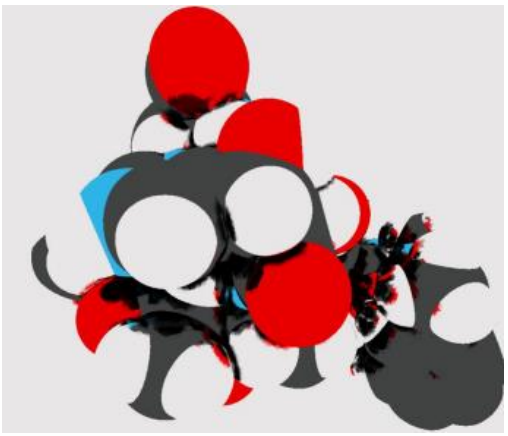

Figure 9: Low FOV.

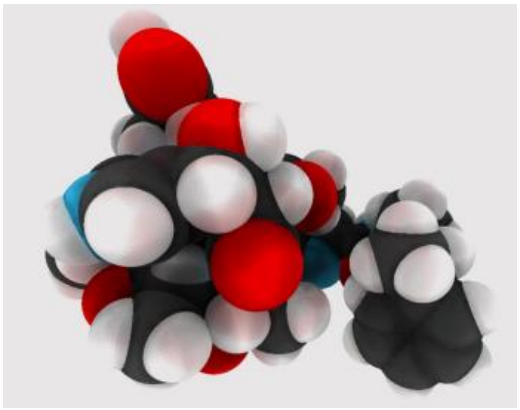

Figure 10: High FOV.

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.

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.

REFERENCES

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

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

[3] Botzki, A. (2021). Pdb file format. Retrieved from VIB: https://elearning.bits.vib.be/courses/protein-structureanalysis/lessons/introduction/topic/pdb-file-format/

[4] 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.

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

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

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

[8] 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.

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

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

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

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

[13] 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

[14] 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

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

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

[17] 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.

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

[19] 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.
[20] Rose, A. (2015). NGL Viewer: a web application for molecular visualization. Nucleic acids research, 43(W1), W576-W579.
[21]Sehnal, D. (2017). LiteMol suite: interactive web-based visualization of large-scale macromolecular structure data. Nature methods, 14(12), 1121-1122.
[22] Sehnal, D. (2021). Mol* Viewer: modern web app for 3D visualization and analysis of large biomolecular structures. Nucleic Acids Research.
[23] Shi, M. (2017). Web3DMol: interactive protein structure visualization based on WebGL. Nucleic acids research, 45(W1), W523-W527.
[24] 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.
[25] Vitsas, N. (2020). Rayground: An Online Educational Tool for Ray Tracing. Eurographics, pp. 1-8.
[26] Vitsas, N. (2021). WebRays: Ray tracing on the web. Ray Tracing Gems II, pp. 281- 299.
[27] Wang, J. (2020). iCn3D, a web-based 3D viewer for sharing 1D/2D/3D representations of biomolecular structures. Bioinformatics, 36(1), 131-135.
[28] Zadvornykh, S. (2016, 11 9). WebGL Masking & Composition. Retrieved from Medium: https://medium.com/@Zadvorsky/webgl-masking-composition75b82dd4cdfd