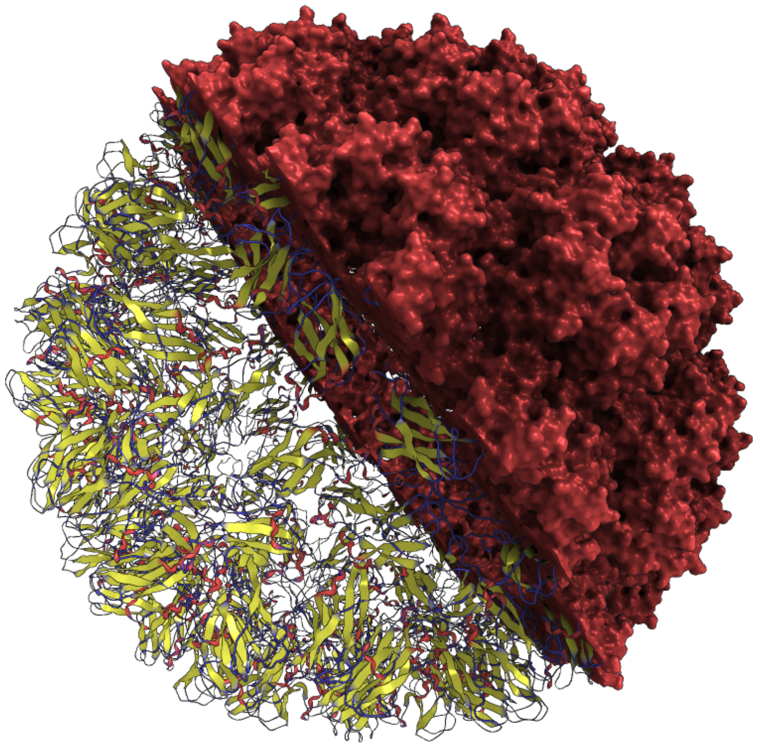


ADVANCED INSPECTION TECHNIQUES FOR MOLECULAR SIMULATIONS

Pedro Hermosilla Casajús

Supervised by

Pere-Pau Vázquez

Àlvar Vinacua

Advanced inspection techniques for molecular simulations

PhD Thesis

Pedro Hermosilla Casajús

ViRVIG research group
Universitat Politècnica de Catalunya

Electronic and Atomic Protein Modeling group
Barcelona Supercomputing Center

Supervised by
Pere-Pau Vázquez
Àlvar Vinacua

Pedro Hermosilla Casajus
June 2017

Acknowledgements

The work presented in this thesis is not only the result of three years of work, but also the result of the knowledge and support received from many people since I first enrolled in the university. Therefore, I would like to dedicate some lines to all the people that make this thesis possible.

First, I would like to thank my advisors, Pere-Pau Vázquez and Alvar Vinacua. I will always be eternally grateful to Pere-Pau for introducing me to the world of research when I was only a student trying to find my way almost ten years ago, for his guidance during all these years, and for his constant motivation and support (even when I was miles away). I am also forever thankful to Alvar for his guidance and for showing me the values a scientist should have. I will really miss his scientific anecdotes. I cannot imagine a better team to help a student take his/her first steps into science.

I would like to thank my number one fan, my partner Gloria. I cannot begin to describe how grateful I am for her daily support, her continuous encouraging words, and her permanent smile which lighted even the darkest days after hours of writing. I could not have gone through this last stage of my thesis without her support.

I am infinitely grateful to my siblings Susana and Carlos, who have morally and emotionally supported me in my life. To my mother, Concepcion, for raising me and providing me with everything I needed, even if it was not always easy, and to my father, Jose Antonio, for all his proud words after each publication.

A very special gratitude goes out to everyone at Moving research group, with a special mention to Isabel Navazo, who always took every student in the group under her wing, and to everyone who worked in the CRV in the past six years. After all the fun we shared, inside and outside the office, you became more than colleagues, you became my friends.

I am also grateful to Victor Guallar and Jorge Estrada for helping and providing the funding for the project, and for all their suggestions and feedback, without which this work could not have been done.

And finally, last but by no means least, to the people who have crossed my path and have contributed directly or indirectly to this thesis.

Thanks you all for your encouragement!

Contents

1 Introduction

Visualization is a discipline whose objective is to communicate information through images. One could argue that visualization has been used even before written language. Cave drawings, cartographic maps and scientific illustrations are some examples of how images were used to transfer and store knowledge. In the past decades, visualization evolved rapidly thanks to computers, as they provided the tools and techniques to generate more complex and accurate images, and the necessity to understand and analyze bigger amounts of data. This made visualization a field of growing importance and interest with a vast number of application areas.

One of these areas is scientific visualization (SciVis). SciVis is focused on the visualization and inspection of scientific data in order to help scientists with understanding, knowledge discovery and hypothesis testing processes. Depending on the nature of the data, the objectives of the visualization and the techniques used to achieve them are different. In this thesis, we focused on molecular visualization, a subarea of SciVis that aims to visually represent molecules and their properties.

Molecular visualization has been a power tool in the development of modern chemistry, having, thus, a direct impact in other areas like biology, medicine, physics or geology. Molecular visualization has its roots at the beginning of the 19^{th} century, when Dalton [Dal10] used hand drawings to illustrate his atomic theory. Over the succeeding decades, molecular visualization helped in communicating new breakthrough discoveries, such as the illustration of the double-helix model of DNA structure proposed by Watson and Crick [WC53] in 1953. In addition, it helped lead to actual discoveries, such as the 3D model used by Kekulé to solve the structure of benzene [Kek66] in 1866. However, molecular visualization did not become a hot research field until a few decades ago. The improvements of methods to visualize the structure of molecules, such as X-Ray crystallography, increased the available known structures of biomolecules (the Protein Data Bank [BWF^{+}00] currently has almost 130k known structures of biomolecules). This vast amount of available data together with the increasing computational power of hardware and the development of methods to simulate the behavior of such structures, as molecular dynamics or Monte Carlo techniques, drove the attention of researchers into this field. In last decades many advances in the field were carried out, improving the quality of molecular visualization itself, providing insight about certain properties of molecules, or even developing new techniques to understand their behavior over time. This thesis aims to contribute to this field with novel visualization techniques which improve the pharmacology drug design processes and enzymatic catalysis studies.

1.1 Motivation

Molecular dynamics simulations are computer simulations of the physical movements of atoms and molecules, and the interactions between them. These simulations are used, among other places, in chemical physics, materials science, and the modeling of biomolecules. In the particular cases we focus on (pharmacology drug design and enzymatic catalysis), molecular dynamics simulations predict the binding mode and binding affinity of a small molecule (the drug) with a biomolecule (figure 1.1). Usually, the objective of the drug is to activate or inhibit the function of the biomolecule, which, in turn, results in a therapeutic benefit to the patient. Therefore, determining the pathway the drug takes to dock onto the biomolecule and how strong the binding is, determines the potency of the drug.

Computing molecular dynamic simulations is a complex and time-consuming process, which usually requires a specific hardware such as the Anton machine [DDG^{+}12], which allows the parallelization of the calculation. Even so, the calculation may last for days or weeks. New technological advances, based on protein structure prediction algorithms and Monte Carlo sampling, have been made [MSG13, BVAG05] to reduce this calculation time to the order of hours or minutes and, with hardware advances, this time could be reduced even more (to the order of seconds) in the near future.

The results generated by these techniques are usually hundreds of trajectories composed of thousands of single steps, leading easily to several gigabytes of data for each simulation. In spite of the speed of these methods, analyzing such results can take days or weeks, since most of the available software focuses on the analysis of single conformations. Moreover, it is not until the analysis

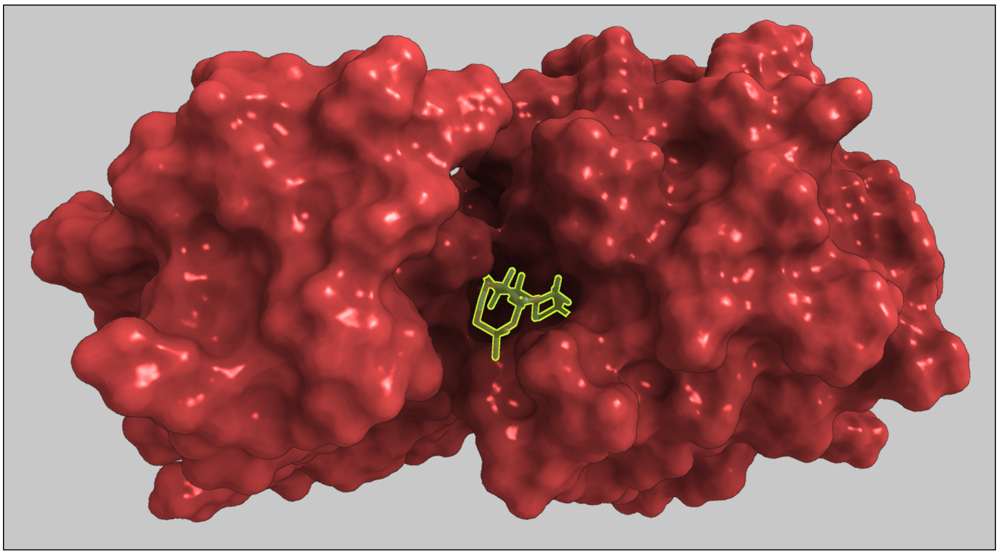


Figure 1.1: *Visual representation of the docking process of a drug into a protein. The drug, highlighted with a yellow silhouette, is trying to enter into an opening of the protein. When the drug achieves the active site, it will create chemical bonds with the residues of the area, activating or inhibiting the function of the biomolecule.*

of these results when the researcher realizes if the parameters chosen to run the simulation were appropriate to achieve the desired results, having to rerun the simulation again if they were not.

Providing new tools and techniques to visualize and to interact with these simulations is crucial to understanding them. Moreover, proper tools could allow the analysis and modification of the simulations in real-time, guiding them to better results (which would directly translate to a reduction of the calculation time).

1.2 Biomolecular background

This section aims to provide a brief introduction into the field of biomolecules, providing the reader with the basic knowledge to understand the techniques developed during this thesis. To obtain a complete introduction into the subject, the reader can refer to [VV10].

Proteins are large biomolecules crucial for life, as they carry out a large number of functions in living organisms. Proteins replicate the DNA, catalyze

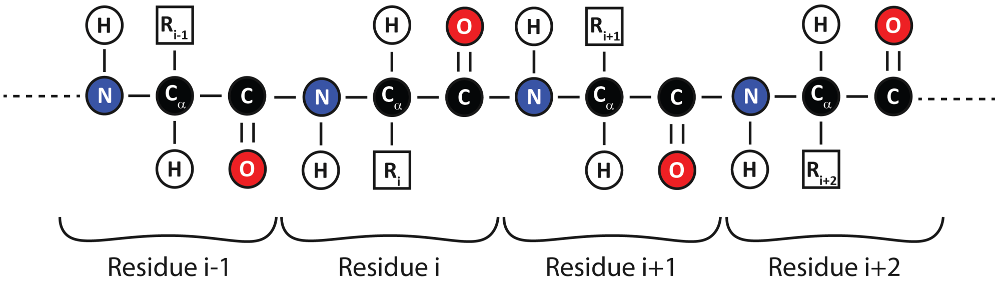


Figure 1.2: *Proteins are sequences of connected amino acids. All amino acids share a common structure (H, N, C_α, C and O atoms in the image) and a side-chain specific to each amino acid type (R_i blocks). The common structure allows the creation of peptide bonds between them, gathering together sequentially and creating a chain known as the protein backbone.*

metabolic reactions, transport molecules from one location to another and so on. They are composed of one or more chains or sequences of amino acids. These amino acids are connected between them by peptide bonds and the order, type, and length of the residue chains define the characteristics of the protein, as they are unique.

There are 20 amino acids which make up proteins [GWCD15, RUC+13]. All of them have a common structure (H, N, C_α, C and O in Figure 1.2) and a residue that is different for each type of amino acid (R in Figure 1.2). The common atoms of the amino acids are known as the protein backbone, as they connect one amino acid to the next in the chain.

The peptide bonds between amino acids are not rigid and they can fold, allowing the movement of the atoms. Each protein has a specific conformation (disposition of the atoms in the space) which makes it operative and functional, also known as its *native conformation*. To achieve this native state, most of the proteins fold by themselves, but, in some special cases, proteins are assisted by other molecules (molecular chaperones). Correct folding is key for the right function of a protein, whereas incorrect folding can lead to a malfunction of the system. Indeed, many diseases are related to incorrect folding of proteins, e. g. Alzheimer's disease or Parkinson's disease [VV10].

The structure of folded proteins can be analyzed at different scales, from atom level to protein complex level. Linderstrøm-Lang, in his third Lane Lecture in 1952 [K.52], introduced a classification to describe the structures of a protein at different levels:

- **Primary structure**: This is the lowest level at which proteins are analyzed, namely the atomic level. The description of the primary structure

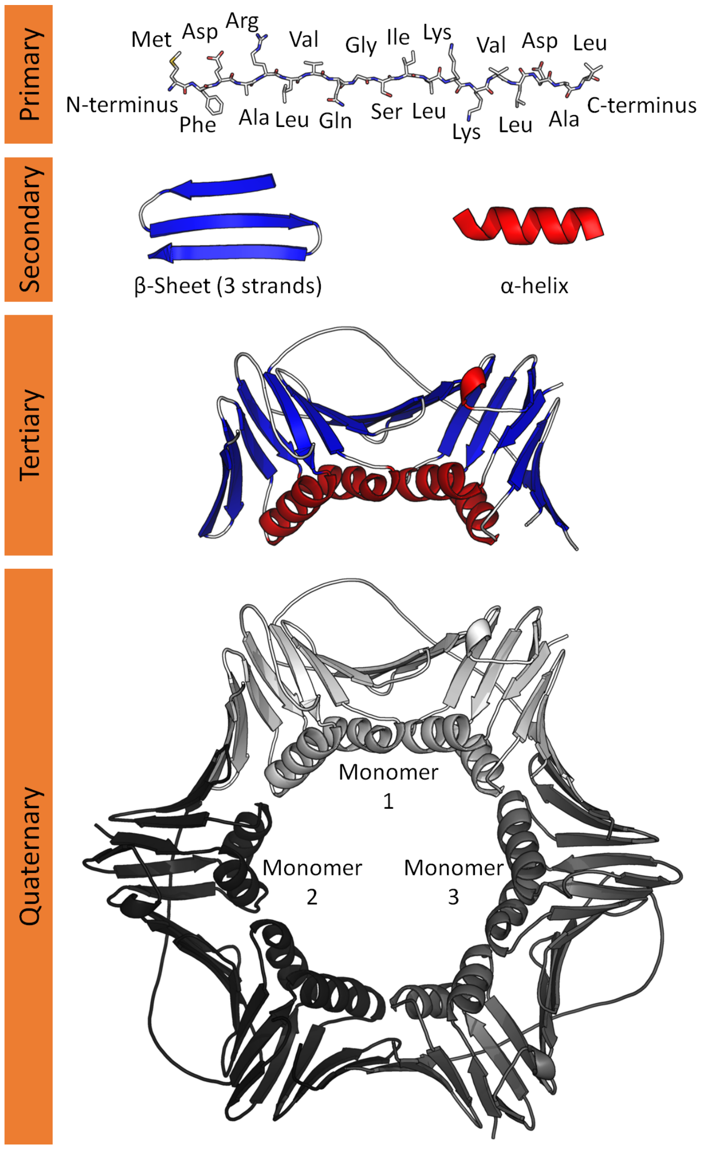


Figure 1.3: *Different classification of the protein structures at different levels. The primary structure refers to the sequence of amino acids of the protein. The secondary structures describe the patterns formed along the backbone due to hydrogen bonds. The tertiary structure is given by the final conformation of the protein. And, lastly, quaternary structure describe the interactions between different single proteins to form a protein complex.* © By Thomas Shafee (Own work) [CC BY 4.0 (http://creativecommons.org/licenses/by/4.0)], via Wikimedia Commons.

is given by the sequence of amino acids which form the backbone of the protein.

- **Secondary structure**: The three-dimensional structure of the protein may lead to the formation of hydrogen bonds between two non-consecutive amino acids in the backbone, creating characteristic patterns, most commonly α-helices and β-sheets. In the α-helix pattern, a part of the backbone acquires the form of a helix due to hydrogen bonds created between amino acids separated by approximately 4 positions in the backbone. On the other hand, in the β-sheet pattern, two distant parts of the backbone (or even two different backbone chains) are interconnected in parallel by hydrogen bonds between at least two or three pairs of amino acids.

- **Tertiary structure**: The term of protein tertiary structure refers to the final structure of a single protein after folding. The residues of the amino acids interact and bond with each other, which makes the protein fold into its native conformation. The tertiary structure of a protein is given by the 3D coordinates of its atoms in this native state.

- **Quaternary structure**: The quaternary structure describes how different *protein subunits* (single proteins) gather together forming a *protein complex* to perform a certain task.

The different levels of protein structure analysis are illustrated in Figure 1.3.

1.3 Addressed problems and contributions

The work presented in this thesis aims to address the following problems related to molecular visualization within the scenario described in section 1.1:

- **Better communication of the molecular shape:** Rendering molecules using basic lighting could make the recognition of their shapes difficult. Ambient occlusion is a well-known technique used to increase the shape perception of objects. It is commonly used in molecular visualization due to the particular shape of molecules (usually possessing several cavities and pockets). However, most of the current techniques do not have an acceptable quality, cannot be computed in real-time and/or do not scale well with the size of the molecule. This thesis presents a new technique to compute the ambient occlusion factor, which overcomes the aforementioned problems [HGVV16].

- **Enhance the understanding of the ligand position during the simulation:** Being able to identify the ligand position in a single time step and during the whole simulation is mandatory to validate a simulation. This thesis introduces a technique to render halos around certain molecule and to generate temporal halos, which indicate the areas where the molecule was in previous steps [HGVV16].

- **Fast generation of complex representations:** The ribbons representation model illustrates the underlying structure of a protein, making the identification of certain properties easy. The rendering techniques commonly used to generate this representation have several limitations. This thesis presents a new technique to generate and render large molecules using the ribbons representation model in real-time [HGVV15b, HGVV15a].

- **Real-time generation of solvent excluded surface (SES):** The solvent excluded surface is a representation method which emphasizes the surface of the molecules that can be reached by another molecule. The computation of these representations is a time-consuming process which the current state-of-the-art is only able to compute in real-time for molecules with a limited number of atoms. In this thesis, we introduce a new method to compute an approximation of the solvent excluded surface for large molecules progressively, providing a quick approximation and allowing interaction with the intermediate results during the refinement [HKG$^+$17].

- **Effective visualization of molecular interaction forces:** Molecular dynamics simulations are driven by the interaction forces between all the atoms of the system. The understanding of these interactions is crucial to understand the simulation. Despite their importance, the visualization of these forces has not received much attention. In this thesis, we introduce a novel system to compute, visualize, and filter these forces for complete simulations in real-time [HEG$^+$17].

1.4 About this document

The remainder of this document is organized as follows: Chapter 2 reviews the state-of-the-art methods related to the research topics addressed in this thesis. Chapter 3 covers the contributions related to efficient rendering of molecules. Thus, Section 3.1 describes the method developed to compute ambient occlusion factors in molecular scenes. After that, Section 3.2 presents the algorithm used to generate halos around the ligand while Section 3.3 shows the techniques used to generate and render the secondary structures of a biomolecule. Lastly, the algorithm to compute solvent excluded surfaces is introduced in Section 3.4. The last contribution of the thesis, the visualization of the interacting forces of a simulation, is presented in Chapter 4. In the last chapter, Chapter 5, the conclusions, the possible future lines of research and the publications are addressed.

2
State of the art

In this chapter, we will review the most significant work in the different fields we have addressed in this PhD thesis. First, we will talk about the models used to represent molecules in molecular science, then, we will talk about the shading technique ambient occlusion, and at the end, we will talk about the visualization of the interaction forces involved in the molecular dynamics simulations.

2.1 Representation methods

In this section, the most popular representation methods used in molecular visualization are presented. Moreover, the use of the different models in molecular software is described, indicating its major shortcomings.

2.1.1 Space-filling

The Space-filling model is one of the most commonly used methods to visualize molecules, as it gives a good approximation of the overall shape of the molecule [Per05]. This model represents each atom of the molecule using a sphere with its size determined by the element's van der Waals radius[1], which illustrates the volume occupied by the molecule [KKL$^+$16].

This model was first used by Pauling, L. and his colleague Corey, R. in 1953 [CP53]. They created a physical model of various molecules (figure 2.1) using hardwood spheres to represent the atoms, with their size proportional

[1]The van der Waals radius describes how close two unbonded atoms can get before they begin to repel one another. Introduced by Pauling in 1945 [Pau45], it is named after Johannes Diderik van der Waals, winner of the 1910 Nobel Prize in Physics.

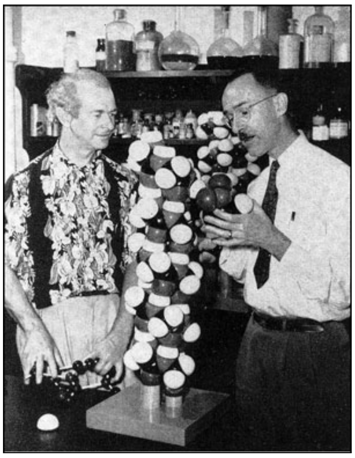


Figure 2.1: *Pauling and Corey with their Space-filling model, 1951.* © Copyright California Institute of Technology.

to the element's van der Waals radius (1 inch = 1 Å). These spheres were cut to create matching flat faces, allowing intersections between them. Moreover, they colored the spheres using a color palette which assigned each atom type a specific color. The initial color palette had the colors listed below:

- White: Hydrogen
- Black: Carbon
- Blue: Nitrogen
- Red: Oxygen

In 1965 L. W. Koltun patented a simplified version of the model [Kol65] where the spheres were made of plastic and joined with snap connectors. The Space-filling model is also known as the CPK model, named after their inventors, **C**orey, **P**auling, and **K**oltun.

Later, with the appearance of molecular visualization software [HDS96, KBW96, GS01, PVV02, PGH$^+$04, MGB$^+$05, Sch16], the Space-filling model was used in computer graphics to represent molecules. The first approach taken by these packages to draw the atom spheres was use triangle meshes [HDS96, KBW96, GS01, PVV02], and the standard rasterization algorithm to project them on screen. Although the graphics pipeline was optimized to render triangles, they are not the best choice to render the spheres of the Space-filling model since they could generate visual artifacts and performance penalties. The process of triangulating a sphere consists in using a set of planes to approximate a curved surface, so that the approximation can produce sampling

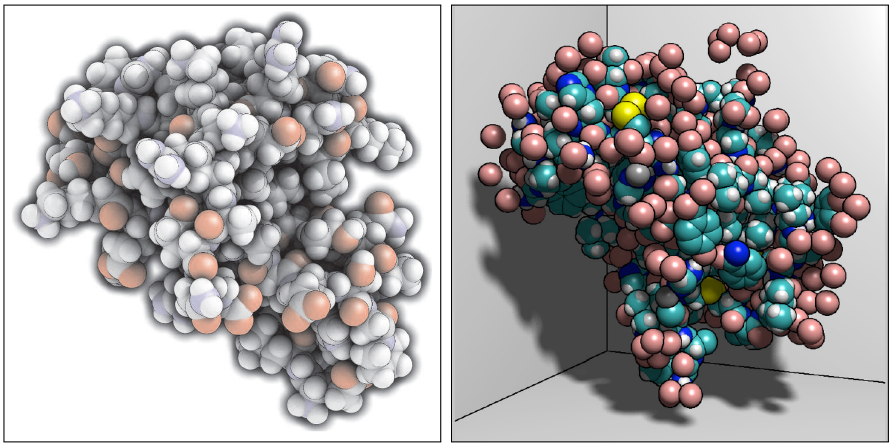


Figure 2.2: *Images of Sigg's implementation [SWBG06] of the Space-filling model on the left and Tarini's [TCM06] on the right.*

artifacts. If the number of triangles used is big enough, we can assure that the projection area in pixels of these triangles is small for a certain distance, so the curved surface of the sphere can be approximated with small error. But if the software allows zooming on the scene, the sampling can always be visible if the camera is close enough. Therefore, the user had to select between performance (determined by the number of triangles of the scene) and image quality (defined by the number of triangles used to approximate the spheres).

To overcome these problems, and with the appearance of programmable GPUs, Randima et Kilgard [RK03] proposed a method to render spheres using impostors. They rendered a quad oriented towards the camera that covered the whole sphere, and, for each pixel, they determined the real shape of the sphere using a set of textures. Their implementation improved both performance and image quality, but the algorithm was only designed for a camera with orthographic projection. In 2004, Bajaj et al. [BDST04], used Randima's technique to render the spheres of the Space-filling model under orthographic and perspective projection, although, in perspective projection, the sphere shape and pixel depth were only approximations, as the technique was designed only for orthographic projection. In 2005, Halm et al. [HOF05] also used textured impostors to visualize molecules, but they improved the rendering under perspective projection by modifying the depth value stored in the texture.

Despite improving the rendering of spheres, textured impostors still had sampling problems (the texture only stored a discrete number of samples, and the performance suffered from texture bandwidth consumption). Procedural impostors, on the other hand, could generate accurate representations without

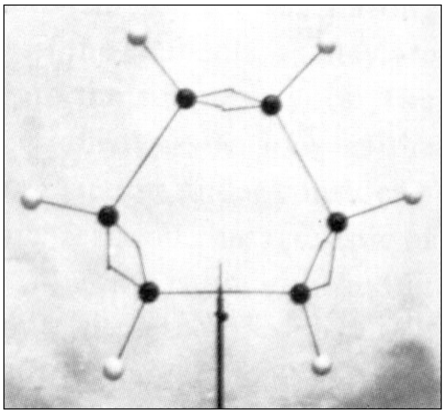


Figure 2.3: *Physical 3D model of Kekulé's Ball and Stick model (1865) [Kek66].* Image obtained from [Per05].

any sampling problem. They were successfully used to represent glyphs by Gumhold in 2003 [Gum03], but it was not until 2006 when Tarini et al. presented their work [TCM06] in which they used them to render molecules. They computed, for an orthographic projection, all the needed information to render the spheres on the fragment shader, including silhouettes and other illustrative effects (Figure 2.2). A more general algorithm to render procedural impostors was developed by Sigg et al. [SWBG06] the same year. They computed the real ray-sphere intersection on the fragment shader for both orthographic and perspective projection (Figure 2.2).

These techniques produced high-quality images for most of the molecules, but they did not scale well when they had to visualize big molecular aggregations, as whole cells. Different approaches were developed to deal with this amount of data, but, as the topic is out of the scope of this thesis, we are only going to mention the most relevant ones. An incremental work carried out by different authors [GRDE10, LBH12, FKE13] led to a system that could handle big molecular systems. This software ray-casted the scene distributed in a grid-based data structure, allowing to find the first atom which intersected with the ray without unnecessary tests. Another approach was developed by Le Muzic et al. in 2014 [LMPSV14], which used the latest features of the GPUs to generate different levels of detail adaptively. A deeper analysis of these techniques can be found in the state-of-the-art report presented by Kozlíková et al. [KKL$^+$16].

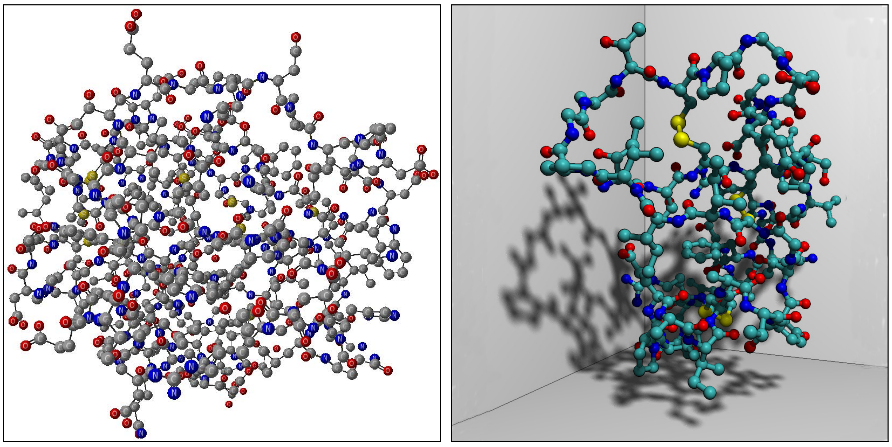


Figure 2.4: *Images of Halm's implementation of the Balls & Stick model [HOF05] on the left and Sigg's implementation [SWBG06] on the rigth.*

2.1.2 Balls & Sticks

The Balls & Sticks model (B&S) and the Space-filling model are similar as both use spheres to represent the atoms. But, nevertheless, the B&S model, uses a smaller sphere radius, being able to represent the bonds between atoms by a set of cylinders. In 1865, during the "Friday Evening Discourse" at London's Royal Institute, this model was first presented by A. Hoffman [Hof65]. Hoffman used colored croquet balls connected by sticks to demonstrate his work, "The Combining Power of Atoms". The next year, Kekulé, created a similar model in 3D to solve the structure of benzene [Kek66] (Figure 2.3). The B&S model was used later by several Visualization Software Packages [HDS96, KBW96, GS01, PVV02, PGH$^+$04, MGB$^+$05, Sch16] as one of the most commonly used methods to communicate the structure of molecules.

The rendering techniques used to visualize molecules with the Space-filling and the Balls and Sticks models evolved together as both of them needed to render spheres to represent the atoms. However, the B&S model also had to render cylinders to represent the bonds between atoms. In the rest of this subsection, we will discuss only the most popular techniques to render cylinders, as the techniques used to render spheres have already been described.

The first approach used to visualize molecules with the B&S model approximated their geometry by triangle meshes and used the standard pipeline to render them. As we discussed in Subsection 2.1.1, this can lead to sampling problems and a low rendering performance. To overcome these problems, as

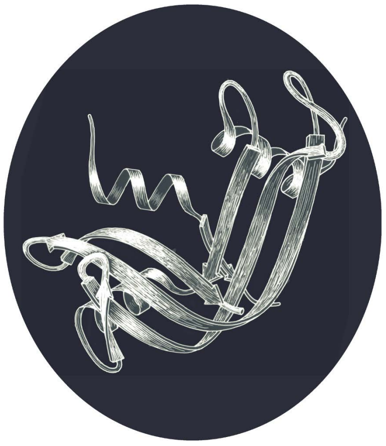


Figure 2.5: *Hand drawing of the structure of a molecule using ribbons (1981) [Ric81].*

for the Space-filling model, Bajaj et al. in 2004 [BDST04] and Halm et al. in 2005 [HOF05] used textured impostors to represent the cylinders (see Figure 2.4). However, as Halm et al. commented on their paper, for perspective projections the pixel depth values obtained with these techniques were only approximations.

A year later, in 2006, two different methods improved cylinder rendering. The first one, the method proposed by Tarini et al. [TCM06], used procedural impostors. Although Tarini's impostors were fast to render, they worked only with orthographic projection cameras (they were developed under that assumption). A more general algorithm was developed by Sigg et al. [SWBG06], where the impostors performed a real ray-surface intersection, allowing both orthographic and perspective projections (see Figure 2.4).

2.1.3 Ribbons

Another relevant representation method is the ribbons mode. This method provides a higher abstraction of the underlying structure of biomolecules by rendering their backbone chain using a set of sheets and tubes. This representation helps experts to focus on particular properties of the biomolecule by removing from the visualization not relevant information.

This method was popularized by Richardson [Ric81]. He used this method on his hand drawings to illustrate the common secondary structures found in biomolecules (see Figure 2.5). Later, the ribbons representation method was used in molecular visualization for its simplicity.

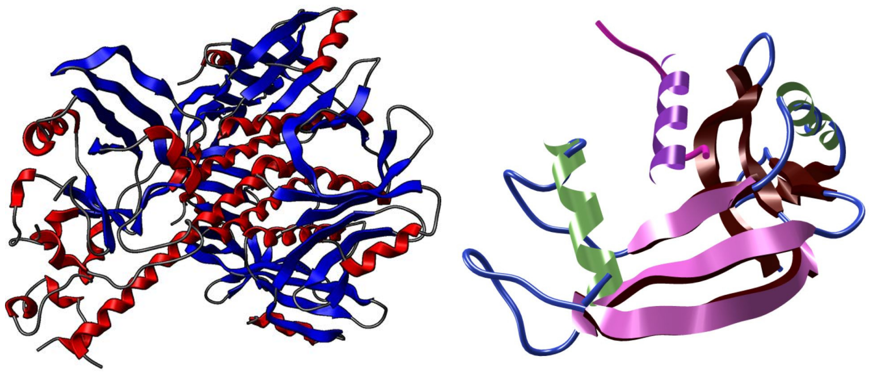


Figure 2.6: *Images of Krone's implementation of the ribbons model [KBE08] on the left and Bagur's implementation [BSN12] on the right.*

The method most commonly used to extract the 3D geometry from the molecule is the one proposed by Carson [Car91], as it is fast to compute and produces a smooth representation. He used a B-Spline to interpolate between the positions of the carbon-α atoms of the amino acids and then, generate the tubes and sheets over them. Later in 2004, this work was extended by Halm et al. [HOF04] to generate an adaptive number of triangles depending on the distance from the camera and their orientation. The reduced number of triangles increased the rendering speed, but the adaptive triangle creation added overhead on the CPU side and an extra transmission load between CPU and GPU (they had to create the triangles every time the camera changed its position and then sent them to GPU memory). Zamborsky et al. [ZSK09], in 2009, proposed a method with the objective of reducing the amount of data stored by an animation. Their method interpolated the backbone and used a predefined representation for each segment type. In 2008, Krone et al. [KBE08], compared three different methods to generate the geometry of the secondary structures. One method created the geometry completely on the CPU, another method was a hybrid CPU-GPU algorithm, and the last one generated all the geometry on the GPU using geometry shaders (see Figure 2.6). The comparison pointed out that the faster method to render molecules on ribbons mode was the precomputation of the geometry on the CPU, but it required too much memory to store the data, making it unsuitable for large simulations. On the other hand, the GPU implementation did not require precomputation or additional stored data, but the rendering was too slow for large molecules. The only method that had acceptable results for all the simulations tested was

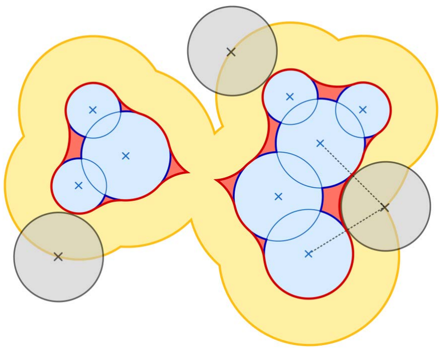


Figure 2.7: *Solvent accessible surface (yellow) and solvent excluded surface (red).* © Image from the state-of-the-art report published by Kozlíková et al. [KKL+16].

the hybrid method, as it provided a good trade-off between rendering speed and stored/transmitted data. These results were obtained using the hardware available in 2008, but, as it was pointed out by Kozlíková et al. in their State of the Art report [KKL+16], these results should be different using the latest GPUs. In 2011, Wahle et al. [WB11], also introduced a new method to generate the ribbons geometry on the GPU. Despite using the GPU, they only exploited basic GPU features, such as vertex and pixel shaders, being able to execute their algorithm in older hardware.

A different approach to visualizing molecules in ribbons mode was taken by Bajaj et al. [BDST04]. They used procedural impostors to represent the α-helix and cylinders to represent the β-sheets. Despite using impostors did not require precomputation and it improved the performance, the visual quality was not as good as the one provided by the 3D geometry. As similar technique was used by Bagur et al. [BSN12] (see Figure 2.6). They also used procedural impostors to represented the secondary structures, but, although the visual quality of the method was better than Bajaj et al.'s, they still had visual artifacts in the joints of the secondary structures with the rest of the protein backbone.

2.1.4 Molecular surfaces

Molecular surfaces are popular representation methods focused on visualizing the area around molecules which match certain criteria.

One of the first surfaces being defined was the *solvent accessible surface* (SAS). This molecular surface was defined by Lee and Richards in 1971 [LR71]. It was designed to illustrate the regions of a molecule that can be accessed by a solvent molecule, approximated by a sphere or probe (water is usually represented by a sphere of radius 1.4 Å). It is described by the center of the sphere while rolls over the atoms of the molecule, represented by the Space-filling model (see Figure 2.7). Although it shows the accessible regions, this molecular surface does not show the real volume of the molecule, and it can intersect with other molecules during a simulation. The SAS is commonly visualized with the same rendering techniques used to represent the Space-filling model, as it can be interpreted as a Space-filling model with the atom radius increased by a user defined probe radius.

In 1977, Richards [Ric77] defined a new molecular surface that showed the regions of a molecule accessible by a solvent molecule but without the disadvantages of the SAS, the *smooth surface*. He defined his surface by a rolling probe as the SAS but, instead of using the center of this probe, he used its surface (red surface on Figure 2.7). Greer and Bush in 1978 [GB78] also defined the same surface, but they called it *solvent excluded surface* (SES), a term that is usually used to refer this surface.

The algorithms used to calculate the SES can be divided into two types, the ones computing the surface discretizing the 3D space surrounding the molecule, and the ones that compute an analytical representation of the surface by determining the implicit surface equations of all patches.

The first type of algorithms discretize the 3D space surrounding the molecule using a regular grid and classify the points on the grid as inside or outside the SES. The scalar field represented by the 3D grid is then triangulated using the *marching cubes* algorithm [LC87] or rendered directly using ray-marching. Although these methods are easy to implement and they are fast for small grids, the memory requirements and the computation time increase with the grid resolution. Can et al. [CCW06] developed an algorithm that falls into this category based on level sets, Yu [Yu09] presented an efficient algorithm which used lists to speed up the computations, and the EDTSurf algorithm, by Xu and Zhang [XZ09], extracted high-quality SES meshes based on Euclidean distance transformations.

The second algorithm type has its origins in Conolly's work [Con83], where he presented the equations to define the SES analytically and an algorithm to compute it. These equations were used later by different algorithms to compute the SES analytically: In 1994, Varshney et al. [VBJ$^+$94], developed a parallelizable algorithm to compute SES, and Edelsbrunner and Mücke [EM94] presented the *alpha-shapes*, which can be used to compute the SES. Later in 1996, Sanner et al. [SOS96] presented an auxiliary data structure called *reduced surface* and an algorithm to compute it from where the SES could be extracted. This algorithm was later improved by Krone et al. [KBE09], being able to update only the parts of the data structure that changed from one frame to another (see Figure 2.8). In 2010, Krone et al. [KDE10] also

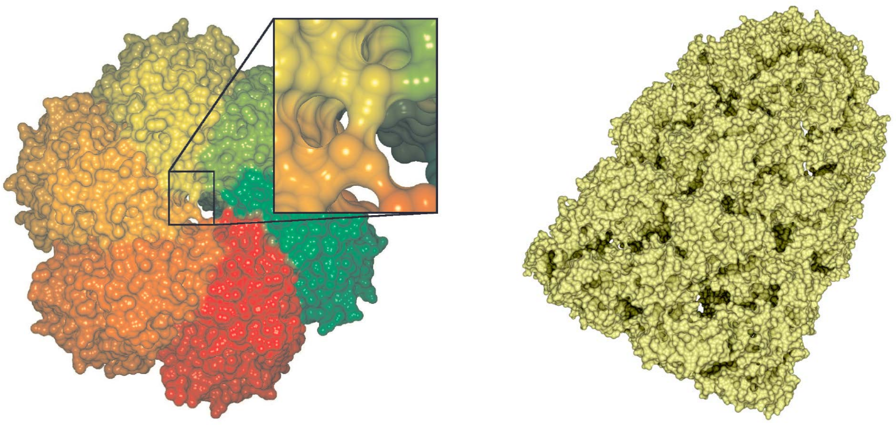


Figure 2.8: *Ray-casting of a solvent excluded surface (SES) ([KBE09] on the left hand side of the figure and [LBPcH10] in the right hand side).*

developed a parallel version of the algorithm that computed the reduced surface on the GPU. The same year Sanner presented his reduced surface data structure, Totrov and Abagyan [TA96] developed a parallelizable algorithm to compute the SES analytically, the *contour-buildup*. This algorithm was implemented on the CPU by Lindow et al. [LBPcH10] obtaining interactive frame rates for molecules up to 10^4 atoms (see Figure 2.8). A year later, Krone et al. [KGE11], adapted the same algorithm for GPUs obtaining interactive frame rates for molecules up to 10^5 atoms, being the fast algorithm to compute the SES analytically.

A couple of decades ago, the visualization method most commonly used to render surfaces was triangular meshes [SOS96, TA96, VBJ$^+$94], but recently, ray-casting was successfully applied to represent them (see Figure 2.8). Krone et al. [KBE09, KDE10] and Lindow et al. [LBPcH10] used ray-casting to render surfaces, reducing thus the rendering time and the memory consumption. In 2012, Parulek and Viola [PV12] presented a modified ray-marching algorithm that computed the implicit surface during the ray traversal.

Other molecular surfaces have been defined aiming to improve the SES and SAS or present extra information. In 1999, Edelsbrunner [Ede99] presented the *molecular skin surface*, a molecular surface that was C^1-continuous. Later, in 2012, Krone et al. introduced the *gaussian density surface*. This surface provided an approximation of the SES with the advantage that it could be computed faster than the SES. Another surface related with the SES is the *ligand excluded surface*, defined by Lindow et al. [LBH14] in 2014. This surface, instead of approximating the solvent molecule by a sphere, the whole molecule was used to test the areas of the protein that could be reached by it.

A more detailed overview of the molecular surface types is given in the state-of-the-art report presented by Kozlíková et al. [KKL+16].

2.2 Illustrative visualization

2.2.1 Introduction

Illustrative visualization encompasses a vast number of techniques which aim to highlight certain features of the visualized model. These techniques were used for centuries to enhance medical and scientific illustrations, focusing on the communication of information more than in mimicking the real world. A clear example can be found in Leonardo Da Vinci and his anatomical illustrations, or in the drawings used by Goodsell [GO92, Goo03] to illustrate molecules and cells.

Later, with the popularization of computer generated images, these techniques were also applied to Scientific Visualization, since they are the perfect tool to communicate scientific results. Silhouette rendering, for example, is able to highlight important parts of the model and it can also increase the perception of its shape [ST90, MH04, HV09, HV10]. A similar technique is the rendering of halos, which, instead of drawing a line in the contour, uses a halo with the same purpose [BG07, JD08]. Both techniques are a common resource in communicating visual information of different data types. In the particular case we are focusing on, molecular visualization, these techniques were successfully applied by Tarini et al. [TCM06]. Other commonly used illustrative visualization technique is hatching [PHWF01], where the color and illumination is simplified and only represented by a set of lines, emphasizing thus the shape of the model. Some examples of this technique applied to molecular visualization can be found in [LKEP14, Web09]. The techniques described in this section are deeply covered in the book [GG01] or in the tutorial [VGB+05], where other illustrative visualization techniques are presented.

2.2.2 Ambient occlusion

Among all illustrative visualization techniques, one that gained a lot of attention, both in molecular visualization and in other areas, is ambient occlusion. It was first described by Zhukov et al. [ZIK98] as an approximation of the indirect illumination in the scene. Since then, it has been widely used in different visualization and computer graphics fields. Many algorithms have been developed to compute or approximate ambient occlusion, using different rendering techniques to achieve the desired trade-off between image quality and computation time. Based on this trade-off we could classify the existing algorithms in two main groups, the algorithms that compute ambient occlusion in real time and the ones that compute it offline.

Offline computations of ambient occlusion usually achieve better quality than the ones computed in real time, since they are not restricted by time, being able to compute it more accurately. Most of these algorithms use ray-tracing and Monte Carlo integration to solve the ambient occlusion equation (the ambient occlusion equation is described in Section 3.1.1). An example of these algorithms is the one presented by Landis in his SIGGRAPH course. [Lan02]. He pre-computed ambient occlusion factors for each point of the model and then stored them in textures. Later, he used these precomputed values during rendering to approximate ambient light. This technique was extended by Pharr and Green [PG04] to use these precomputed values in real-time applications. They also used the same preprocessing step, but the precomputed values were then used to calculate the object's ambient light during rendering in real-time. The main drawback of this technique was that the computed ambient occlusion factors were only valid for static scenes, as soon as one object of the scene moved the values had to be computed again. The same year, Sattler et al. [SSZK04] developed an algorithm to compute the ambient occlusion terms also in a preprocessing step, but, instead of tracing rays through the scene, they used the graphics hardware to create multiple shadow maps from a set of predefined views to approximate the visibility of the scene points.

Real-time applications raised the necessity of having more realistic illumination, thus driving research into creating new algorithms which approximate ambient occlusion in real-time. These algorithms, focusing on the speed instead of on the quality, compute rougher approximations of the ambient occlusion factors, but with the advantage of doing it in real-time. The algorithms of this group can be again classified into subgroups based on the information used to compute the visibility of each point: screen-space ambient occlusion, geometry-based ambient occlusion, and volume-based ambient occlusion.

The algorithms of the first group, screen-space algorithms, are those algorithms which only use 2,5D information to compute the visibility of a point. These algorithms sample the camera depth buffer to roughly estimate the visibility of a point. These algorithms are commonly used in games since they are very fast and can be easily integrated into their rendering pipelines. However, the poor sampling and the lack of 3D information cause important visual artifacts. The first image-space algorithm was introduced by Mittring [Mit07] in 2007, which was included in the game Crysis. This method was later improved by different authors [BSD08, FM08, MOBH11], but all of them based their algorithms on comparing depth information and normals of the neighboring pixels.

Geometry-based ambient occlusion algorithms, on the other hand, compute visibility using the 3D information of the scene, achieving better quality than screen-space methods. However, these techniques are more expensive to compute. Their cost depends on the number of objects in the scene, making them not practical for big dynamic scenes. Inside this group, there is a wide variety of algorithms. The method proposed by Bunnell in 2005 [Bun05] is a clear example of a geometry-based algorithm. This method uses proxy elements to approximate scene objects during visibility computation, simplifying thus the intersection tests. The idea of using proxy elements during visibility tests was also used by Shanmugam and Arikan [SA07]. They used proxy elements to compute low-frequency occlusions, while high-frequency occlusions were calculated using a screen-space method. A more recent algorithm, presented by McGuire in 2010 [McG10], extruded the object's triangles in the geometry shader to generate a volume for each triangle. Then, in the fragment shader, he accumulated the individual contributions of the volumes to approximate the occlusion of the pixel.

The algorithms of the last group, volume-based ambient occlusion algorithms, compute the visibility of each point sampling a volume which approximates the geometry of the scene. These techniques use 3D information of the scene to compute the visibility of each point, but, on the contrary to geometry-based approaches, their cost is decoupled from the complexity of the scene. The approach used by Papaioannou et al. [PMP10] falls into this category. A more interesting method was proposed by Crassin et al. [CNS+11]. They created a hybrid method to approximate the ambient occlusion factor and the indirect illumination in dynamic scenes with real-time frame rates. This method creates a hierarchical approximation of the scene, a *Sparse Octree*, which is updated with the dynamic objects every frame, and then, in image-space, the ambient occlusion factor is calculated for each pixel sampling this data structure.

Ambient occlusion is a commonly used technique in molecular visualization since Tarini used it in his software QtMol [TCM06]. Ambient occlusion is the perfect tool to easily identify the shape and cavities of the molecules, something of key importance in molecular simulation analysis. Tarini et al. [TCM06] adapted the method proposed by Sattler et al. [SSZK04] to visualize molecules with the Space-filling and Balls & Sticks models. They calculated multiple shadows maps from different views and saved the accumulated shadow factor in a texture. Then, during rendering, the pixel shader accessed the correct position of the texture to retrieve the ambient occlusion factor. Despite obtaining

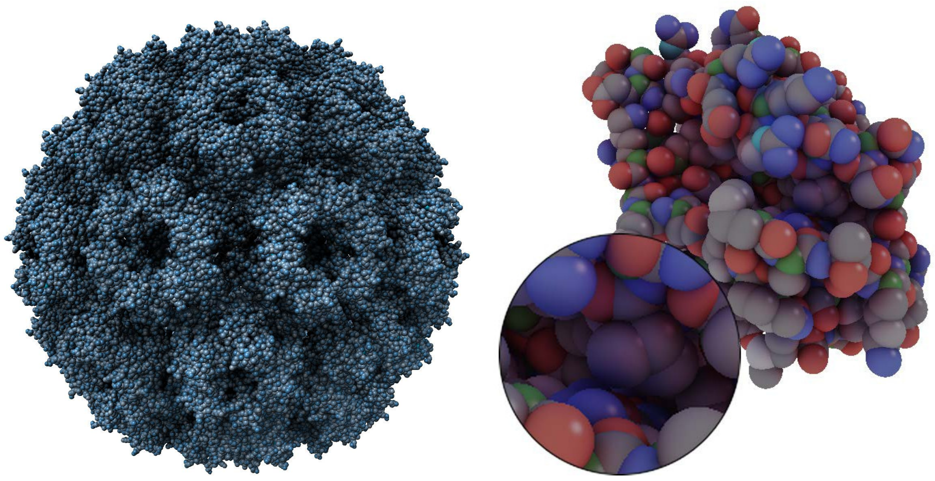


Figure 2.9: *Image obtained with Grottel et al.'s method [GKSE12] on the left, and an image of the method proposed by Skånberg et al. [SVGR16] on the right (this method also includes diffuse interreflections).*

high-quality images, this method is not suitable for dynamic scenes, since the ambient occlusion is precomputed. Moreover, the size of the texture used to store the occlusion factors increases with the number of atoms, making it not very scalable. Later, in 2012, Grottel et al. [GKSE12] proposed a volume-based method to calculate ambient occlusion factors for molecular scenes (see left Figure 2.9). This method is similar to the method developed by Crassin et al. [CNS$^+$11]. They created a coarse approximation of the molecule using a 3D occupancy grid and, in image-space, they calculated the ambient occlusion factor for each pixel sampling this data structure. They achieved high frame rates, but the approximation of the scene did not have enough resolution to capture occlusions between nearby objects. This method was later improved by Staib et al. [SGG15] by using a hierarchical 3D grid to approximate the scene and, in screen-space, they computed the ambient occlusion factor using voxel cone tracing over the data structure. Their implementation also allowed them to use transparencies and global illumination effects. A complete different (geometry-based) approach was presented by Skånberg et al. [SVGR16] (see right Figure 2.9). They performed a search in the local neighborhood of an atom and computed the occlusion contribution of each neighbor. They also computed the diffuse interreflection between the atoms using a mathematical formula obtained with a symbolic regression algorithm.

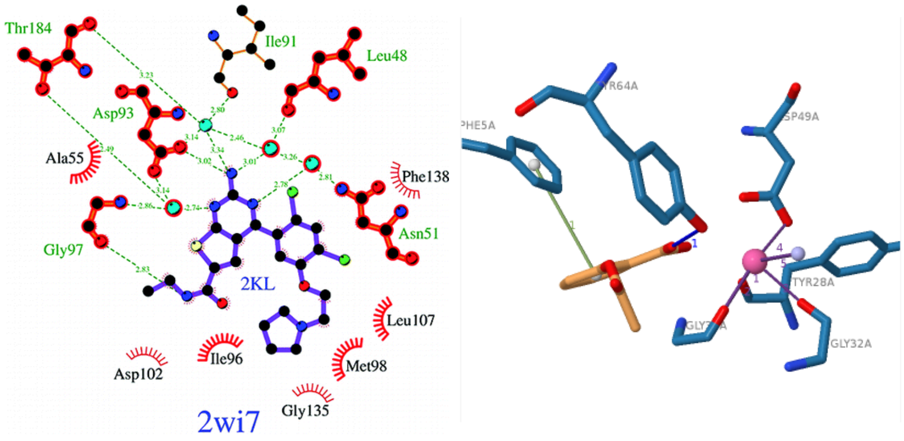


Figure 2.10: *Image generated by LigPlot+ [LS11] of the closest interactions between a molecule and a protein on the left, and window of the PLIP software [SSH$^+$15] on the rigth.*

2.3 Visualization of interaction forces

Nowadays molecular visualization packages have a vast number of techniques to visualize the conformation of the atoms during a simulation, but the techniques available to visualize the interaction forces that drive these simulations are limited. Most of the software packages use a 2D visualization of the molecule of interest (ligand) with the nearest interacting atoms (or groups of atoms), making difficult to understand the real 3D arrangement of these atoms. Moreover, most of them have limited (or null) interaction and only allow the analysis of a single step of the simulation. This is the case of LigPlot+ [LS11], where a 2D view of the ligand is shown with some motifs that give information about the interaction of this molecule with the neighboring residues (Figure 2.10). A similar result can be obtained by using LeView [Cab13], Maestro [Sch16] or PoseView [SMR06].

Visualizing these interactions in 3D has had the same attention than in 2D, and they also focus on showing only the closest interactions. Furthermore, their analysis is limited to single frames of the simulation with insufficient interaction. However, visualizing the underlying physics is common in related fields, such as material science, where Grottel et al. [GBM$^+$12] represent scalar fields through color overlaid on the molecular surfaces and represent electrostatic dipoles with arrows. Similar representations using isosurfaces are common to other crystallography applications such as Vesta [MI08, MI11], where the overlaid color represents electrostatic potential or hydrophobicity.

Hyde [SHL+12] uses a similar representation to code the total affinity energy contribution, whereby the process takes several seconds. Cipriano and Gleicher [CG07] illustrate charges over the molecular surface by stylizing both the surface shape and the charge values. Günter et al. also focus on the atom level, where the signed electron density and reduced gradient fields are computed and then simplified to illustrate van der Waals and steric repulsion forces between atoms [GBCG+14]. By showing colored dots at the contact surface between atoms, representing the van der Waals and hydrogen-bonding interactions, Word et al. allow the evaluation of atom packing in biomolecular structures [WLL+99]. Another tool, LigandScout, exploits several views to support drug designers when screening chemical databases [WL05]. It allows for the interactive creation of so-called pharmacophores (which are based on known ligands) that act as templates for finding new ligands. In the creation process, LigandScout highlights the ligands key features that interact with the protein and supports surface coloring based on lipophilicity, hydrogen bonding or charge, using predefined scoring functions. The PLIP system (Figure 2.10) is a web service that generates 3D views focused at the atom level and showing several interaction types [SSH+15]. However, as discussed later, no interaction or filtering is supported, and it is not designed to deal with a sequence of frames. More recently, Skånberg et al. have proposed to visualize energy interactions between atoms through diffuse interreflections computed for the surfaces of the atoms [SVGR16]. Falk et al. visualize molecule reactions by means of arrows augmenting paths representing molecule trajectories [FKRE09]. Khazanov and Carlson exploit tables to communicate molecule interactions [KC13]. They also communicate the interaction between ligand and binding site through modification of color and van der Waals radii on an atom scale and indirectly address the residue scale by performing this depiction individually for each amino acid. Sarikaya et al. also take into account the residue scale, by visualizing classifier performance with respect to protein chains on which a classifier has operated [SAMG14]. Finally, to communicate the differences of surface projected parameters, Scharnowski et al. propose to use deformable models [SKR+14].

In the following chapters, we present the methods designed throughout the development of this thesis to overcome the limitations of the techniques described in this chapter.

3 Efficient rendering of large molecular models

This chapter introduces the contributions of the thesis related to efficient rendering of large molecules. Techniques to accelerate the rendering of common representation methods and illustrative techniques to enhance the visualization are presented.

3.1 Real-time volume-based ambient occlusion

Illustrative visualization techniques are commonly used in helping to highlight specific features of the model or in favoring the recognition of the shape of the scene. Many of these techniques are also used in molecular visualization [Goo03], as the understanding of the scene is crucial in this kind of visualization. In particular, ambient occlusion is a rendering technique which approximates the global illumination at every point in the scene based on their visibility. As a result, this technique darkens the concave areas of an object and brightens the convex ones, making it suitable for emphasizing the shape of the molecules.

Ambient occlusion facilitates the recognition of the molecular conformation and the identification of features like pockets or tunnels. However, this technique is expensive to compute. Many researchers have approximate this effect in real-time by computing it on screen-space [Mit07, FM08, Kaj09]. Despite handling dynamics scenes at higher frame rates, these techniques suffer from artifacts, as they do not use all the information of the scene to compute the ambient occlusion factors. Other techniques were developed for the specific case of molecules [TCM06, GKSE12] in which the computation takes into account the whole scene (or a simplification of it). Nevertheless, they cannot be computed in real-time or the resulting quality is not good enough. Later

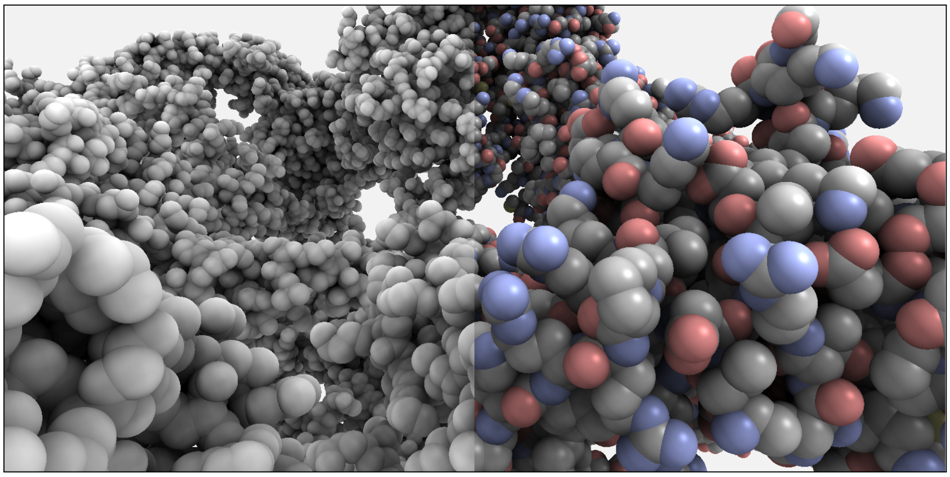


Figure 3.1: *The left part of the figure is generated using only the ambient occlusion factors computed by the algorithm presented in this section, and the right part of the image is the final composition of the standard Phong shading with our ambient occlusion factors.*

on, while the work presented here was under revision, a similar technique was presented by Staib et al. [SGG15].

The previous chapter analyzed the most important techniques to compute this effect, for both molecular and generic scenes. This section, on the other hand, introduces a new method to compute the ambient occlusion factor for the specific case of molecules represented by the Space-filling or the Balls & Sticks models. This method takes advantage of the simplicity of the elements that compose these scenes, to approximate the ambient occlusion factor in real-time for molecules up to 1356K atoms. The resulting high-quality shading we obtain is illustrated in Figure 3.1.

The rest of the section is organized as follows: Section 3.1.1 introduces the reader to ambient occlusion, Section 3.1.2 gives an overview of the method, in Section 3.1.3 the algorithm is described, and, in Sections 3.1.4 and 3.1.5, the results and the conclusions are presented.

3.1.1 Ambient occlusion

Ambient occlusion is a rendering technique which aims to improve the shape perception of a scene. This technique shades each point of the scene as a function of its visibility. Points occluded by near geometry appear darker than unoccluded objects, which are shaded with a brighter tone. Figure 3.2 presents

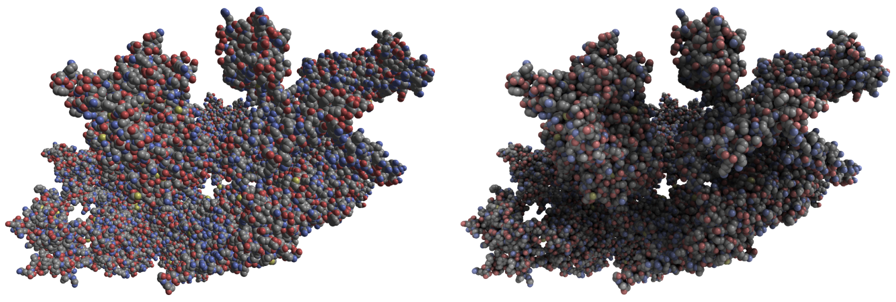


Figure 3.2: *Comparison of the same molecule rendered using a constant term to approximate the ambient light (left) and rendered using the ambient occlusion algorithm (right). Ambient occlusion clearly improves the shape perception of the molecule, making easily noticeable features like tunnels or cavities.*

a comparison of a molecule rendered using a constant ambient term and the same molecule rendered using ambient occlusion. This figure clearly illustrates how in the image rendered using ambient occlusion the shape perception is increased, making their cavities and tunnels easily recognizable. This section introduces ambient occlusion briefly, for completeness, following the description gave by Sunet in his Master Thesis [Sun16] (where the interested reader may find a complete description).

Ambient occlusion can be understood as an approximation of the rendering equation, from where its mathematical equation can be derived through some simplifications. The rendering equation is used by rendering algorithms to shade the objects of a scene since it calculates the amount of reflected light by a point in a specific direction. This equation defines the outgoing light as a function of the incoming light from all possible directions:

$$L_o(p, w_o) = \int_\Omega f(p, \omega_o, \omega_i) L_i(p, \omega_i) \cos\theta_i \mathrm{d}\omega_i \tag{3.1}$$

In this equation, $L_o(p, \omega_o)$ refers to the amount of reflected light by point p in the direction w_o and is calculated as the sum of the incident light $(L_i(p, \omega_i))$ from all possible incoming directions (ω_i) multiplied by the surface reflectance properties $(f(p, \omega_o, \omega_i))$. The incoming light directions are defined as all the directions within the hemisphere centered at the surface normal, so an integral over this hemisphere has to be evaluated, being ω_i a differential solid angle in this hemisphere (an infinitesimally thin cone of incident directions). The amount of incident light in each direction is modulated then by the surface reflectance properties. $f(p, \omega_o, \omega_i)$, also known as *bidirectional reflectance dis-*

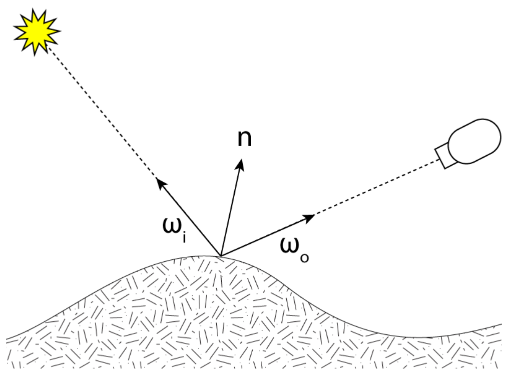


Figure 3.3: *Parameters involved in the evaluation of the BRDF. The BRDF determines the amount of reflected light in the outgoing direction ω_o, by a point with normal n, for an incoming light direction ω_i.*

tribution function or BRDF, is a function that encodes the material behavior, modulating the amount of reflected light by a surface point (p) in a particular outgoing light direction (ω_o) for a given incoming light direction (w_i), i. e., it modulates the reflected light for a given input and output directions. Figure 3.3 illustrates this process. The final light contribution is attenuated by cosθ, the cosine of the angle between the normal and the incoming light direction ω_i.

This equation does not have an analytical solution, so it can only be approximated. One of the most commonly used methods to approximate the rendering equation is ray-tracing. Ray-tracing simulates the light in the scene with a set of rays, and the integral of the equation is approximated using Monte Carlo integration. Ray-tracing, like all the other methods to approximate this equation, is computationally expensive due to the recursive nature of the rendering equation. The incoming light in a given direction ω_i may arrive not only from a light source but also reflected from another surface, where the rendering equation has to be evaluated too. This recursion increases the computational cost exponentially, depending on the degree of accuracy we want to achieve (determined by the number of light bounces simulated).

One of the simplest approximations of the rendering equation is to take into account only the direct light coming from the light sources and ignoring the light bounced on other surfaces. Despite being very easy to evaluate, this approximation does not generate realistic results, since all the points the light cannot reach are rendered black. A better approximation that keeps this simplicity is assuming that bounced light reaches every point in every direction with the same intensity, transforming the indirect light contribution into a constant. The images generated by this method appear to be more realistic because all parts of the scene are shaded. However, in those parts where only the ambient term contributes to the lighting, the depth and shape perception is reduced.

Ambient occlusion is an algorithm which tries to approximate indirect light by computing an ambient term for each point of the scene instead of using a constant. This ambient term is computed based on the visibility of the point, giving a measure of how much light could arrive at the point without being occluded by other surfaces. In order to derive the ambient occlusion equation from the rendering equation, this method makes two assumptions. The first assumption made by this method is that surfaces are perfectly diffuse, i. e. they reflect light equally in every direction, which transforms our rendering equation in:

$$L_o(p, w_o) = k \int_\Omega L_i(p, \omega_i)\cos\theta_i \mathrm{d}\omega_i \tag{3.2}$$

substituting the BRDF of the surface, $f(p, \omega_o, \omega_i)$, by a constant k. The other assumption made by this method is that light intensity is independent of the direction in which light reaches a point, being the same for all directions. Here is where ambient occlusion differs from the constant ambient term. The constant ambient term method assumes that light reaches the point equally in all directions, but, ambient occlusion instead, assumes that light reaches the point equally only in the directions where the light is not occluded by other surfaces. If the light coming from a specific direction reaches another surface first, its intensity should not be added to the outcoming light intensity of the point. This method can be expressed by the following modification of the rendering equation:

$$L_o(p, w_o) = k \int_\Omega V(p, \omega_i)\cos\theta_i \mathrm{d}\omega_i \tag{3.3}$$

where

$$V(p, \omega_i) = \begin{cases} 0 & p \text{ occluded in direction } \omega_i \\ 1 & \text{otherwise} \end{cases} \tag{3.4}$$

In this new equation the incoming light intensity, $L_i(p, \omega_i)$, is substituted by the visibility function $V(p, \omega_i)$, which has a value of 0 when the light is occluded by another surface in the direction ω_i and one otherwise. For convenience, we define the ambient occlusion factors in the range $[0, 1]$, which also helps us to derive the value of the constant k. For a completely unoccluded point, the ambient occlusion factor should be 1. Moreover, in this case, every individual evaluation of $V(p, \omega_i)$ is also evaluated to 1, since no light direction is occluded. Putting these two definitions together we can easily derive the value of k:

$$\begin{aligned} L_o(p_{\text{unoccluded}}, w_o) &= k \int_\Omega V(p_{\text{unoccluded}}, \omega_i) \text{cos}\theta_i \text{d}\omega_i \\ &= k \int_\Omega \text{cos}\theta_i \text{d}\omega_i \\ &= k\pi \end{aligned} \tag{3.5}$$

$$k = \frac{1}{\pi} \tag{3.6}$$

which gives us the last element to define our ambient occlusion equation:

$$L_o(p, w_o) = \frac{1}{\pi} \int_\Omega V(p, \omega_i) \text{cos}\theta_i \text{d}\omega_i \tag{3.7}$$

This equation computes the ambient occlusion factors for points in a scene, but without any further modification, it can lead to completely pitch black scenes in some cases. When we compute this value for closed scenes, like rooms or buildings, this equation will generate ambient occlusion factors equal to 0 for all the points. The visibility function $V(p, \omega_i)$ will always be evaluated to 0 for all light directions (because the light will hit the walls first). To avoid this problem, a modification of the ambient occlusion equation was designed where the visibility function $V(p, \omega_i)$ is substituted by a *fall-off* function ρ.

$$L_o(p, w_o) = \frac{1}{\pi} \int_\Omega \rho(d(p, \omega_i)) \text{cos}\theta_i \text{d}\omega_i \tag{3.8}$$

ρ is a function that takes as a parameter the distance between the point p and the point which occludes p in the direction ω_i. The value returned by ρ is 0 for small distances and increases to 1 when the occluder is farther than a given threshold.

$$p(d) = \begin{cases} f(d) \in [0, 1] & d < \text{threshold} \\ 1 & \text{otherwise} \end{cases} \tag{3.9}$$

This modification of the ambient occlusion equation is known as *ambient obscurance*, and not only makes ambient occlusion work in closed scenes but also reduces the algorithm to a local problem, since the visibility tests are performed only with the surrounding geometry. In practice, this equation is the one used in literature when referring to ambient occlusion.

3.1.2 Overview

For molecules, we propose an algorithm that aims to approximate the ambient occlusion equation by using a volume-based simplification of the scene during the point visibility tests instead of the real geometry. This method is integrated into a deferred rendering pipeline (an overview of this pipeline is illustrated in Figure 3.4), and has the following steps:

1. In the first step, the atoms and bonds of the molecule are rendered. As discussed in Chapter 2, the fastest method to render molecules represented by Space-filling or Balls & Sticks models is using procedural impostors. Similar to [SWBG06], our method renders a quad for each element (an atom or a bond), ensuring that the projection of the quad on the screen covers the projection of the element. Then, for each pixel, the ray-element intersection test is performed, and the normal and depth values are computed. These values (normal and depth), together with the atom color, are stored for each pixel in a G-Buffer composed by two floating-point buffers.

2. While the rendering of the molecules is taking place, two compute shaders process each atom and bond and fill our data structure.

3. When the two previous passes are finished, a third pass evaluates the ambient occlusion for each pixel on the screen using the data structure computed in step 2.

4. In the last pass, the color, normal, depth and ambient occlusion values computed in the previous steps are used to evaluate the lighting for each pixel.

The steps 2 and 3 are discussed in more detail in the next section, but steps 1 and 4, since they implement previously known techniques, are not covered.

3.1.3 Algorithm

The proposed method, as mentioned in the previous section, is composed mainly of two different steps: the simplification of the scene and the ambient occlusion computation. The first one uses a 3D hierarchical data structure to store the simplified representation of the scene, which we call *Occupancy pyramid*. This pyramid is filled by the first step and then used by the second one to compute the final ambient occlusion values. This algorithm is based on the work presented by Crassin et al. [CNS+11] in 2011, but, our method

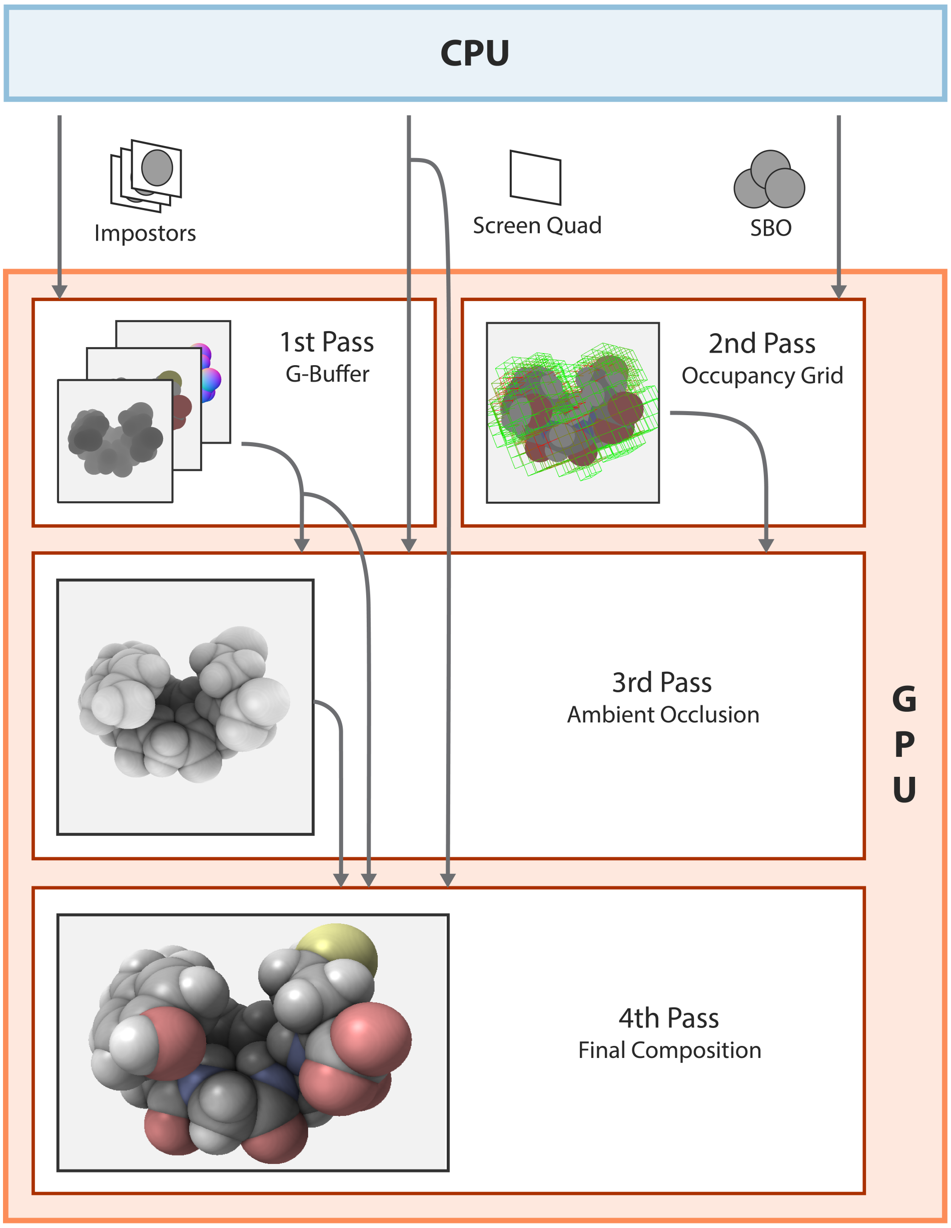


Figure 3.4: *The pipeline of the algorithm. In the first step, the G-Buffer is filled out with the scene information. In parallel, the second step computes the occupancy pyramid. In the third step, the occlusion factors are computed for each pixel. And, in the last one, the values obtained int the previous steps are combined to generate the final image.*

does not require a preprocess step, being able to compute our data structure at every frame.

3.1.3.1 Occupancy pyramid

The Occupancy pyramid is a hierarchical 3D voxelization of the space delimited by the bounding box of the molecule, where each voxel contains an occupancy value. These values approximate the volume of the voxel occupied by the atoms of a molecule. Therefore, this data structure allows us to estimate, with only one texture fetch, the occlusion generated by the part of the molecule contained inside the voxel. Due to the hierarchical nature of the data structure, different volume sizes can be consulted by selecting different levels in the hierarchy.

In our implementation, we used a maximum and minimum resolution of 128^3 and 8^3 respectively, having five different resolutions of our occupancy approximation. These numbers were selected empirically and were found to generate good quality ambient occlusion for all the models we tried.

Moreover, the fact that we use a low-resolution data structure allowed us to use a floating-point 3D texture to represent it. Despite not being optimized to remove empty space areas, 3D textures provide a hardware implementation of trilinear interpolation and texture cache access. These two hardware features improve the performance of the reading and writing operations compared to Crassin's approach, which used a less GPU-friendly data structure, the *Sparse Octree.*

3.1.3.2 Occupancy computation

In order to compute the occupancy values of each voxel, we execute two compute shaders that calculate the occupancy contribution of each element of the scene (spheres and cylinders). One of the shaders computes the occupancy contribution of the spheres (atoms), and the other shader computes the occupancy contribution of the cylinders (bonds).

Computing the exact intersecting volume between an element and a voxel would make our algorithm not suitable for real-time applications since it is computationally expensive. We chose, instead, an algorithm that, although it overestimates the volume, is cheap to compute and produces acceptable results (see Figure 3.11 for a comparison between our method and the AO computed using path tracing).

The list of atoms is stored in GPU memory using one *Shader Storage Buffer*, where each atom stores its radius and the 3D coordinates of its center. The bond list is also stored in another Shader Storage Buffer, where each bond stores the 3D coordinates of the two atoms connected by the bond. The

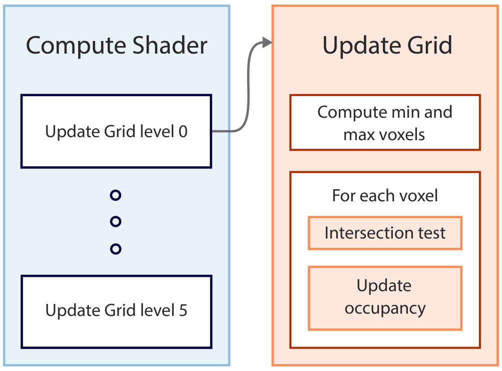


Figure 3.5: *Overview of the algorithm executed by the compute shaders to fill out the Occupancy pyramid. First, the shader determines the voxels that possibly intersect with the element, and for each of them, an intersection test is performed. If there is an intersection, the shader increments the voxel value with an approximation of the incident volume of the element into the voxel.*

two compute shaders described before are then executed, with their respective Shader Storage Buffer as input (Figure 3.5). The threads of these compute shaders first determine the voxels (of the different levels of the occupancy pyramid) which the element could intersect. These voxels are defined as the ones that intersect with the volume of the element's bounding box. The selection of these voxels can be carried out easily by determining the voxels where the minimum and maximum points of the bounding box lay, and then select the voxels between the two. This simple algorithm is able to skip most of the voxels easily at a low computational cost. Then, for each selected voxel, a more accurate intersection test is computed to determine whether the element really intersects the voxel or not. Where the intersection test succeeds, the compute shader increments the occupancy value with an approximation of the incident volume.

In order to approximate the overlap of an atom (given by a center and a radius) with a voxel, we find the point in the voxel that is closest to the center of the atom and compute the distance between them (r). If that distance r is less than the radius r_a of the atom, we increase the occupancy of the voxel by the equation:

$$\frac{r_a - r}{D} \tag{3.10}$$

where D is the length of the diagonal of a voxel (see Figure 3.6-right). In

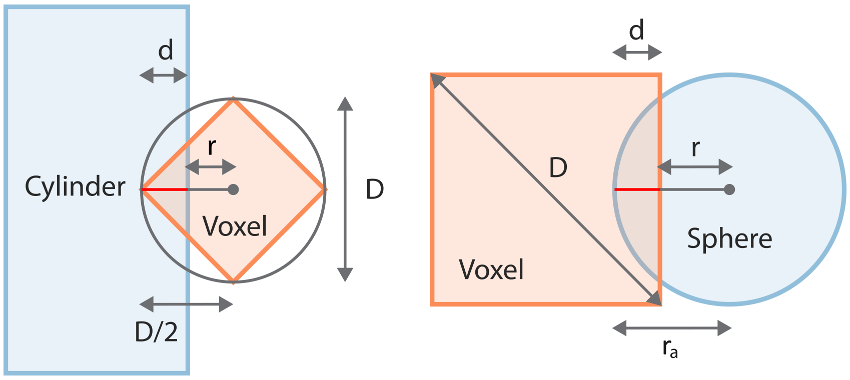


Figure 3.6: *Evaluation of the occupancy of a voxel. The portion of a voxel that is occupied by a bond (left) or atom (right) is approximated as a ratio of the penetration distance d and the diagonal of the voxel D.*

the case of bonds, we compute the point on the cylinder that is closest to the center of the voxel and compare the signed distance between them with D/2 (see Figure 3.6-left). If the distance is smaller than D/2, we increment the occupancy of the voxel by:

$$\frac{\frac{D}{2} - r}{D} \tag{3.11}$$

Having two or more threads trying to add its occupancy contribution to one voxel at the same time is a common scenario during the update of the data structure. These collisions could lead to different occupancy values for a voxel on consecutive frames, generating a flicker effect on the computed occlusion factors. Since our factors are floating-point values, we cannot use the integer atomic operations provided by OpenGL to overcome this problem. Instead, and like in [CG12], we access our floating-point texture as an integer texture using *imageAtomicCompSwap()*, converting values from float to int and vice versa during read and write operations via *floatBitsToUint()* and *uintBitsToFloat()*.

Despite removing the synchronization problem between threads, this mechanism adds extra computational cost because some threads have to wait until they are able to write into memory. As the number of collisions increases, the system slows down. In order to reduce the number collisions, we shuffle the elements in the Shader Storage Buffer, ensuring that elements that are close in the scene are far away in the buffer. Since the atoms of the residues are read following the order of the backbone, the atoms of the same residue are packed together. Shifting the atoms a fixed number of positions ensures that two neighbor atoms in the resulting buffer will not be close in the backbone.

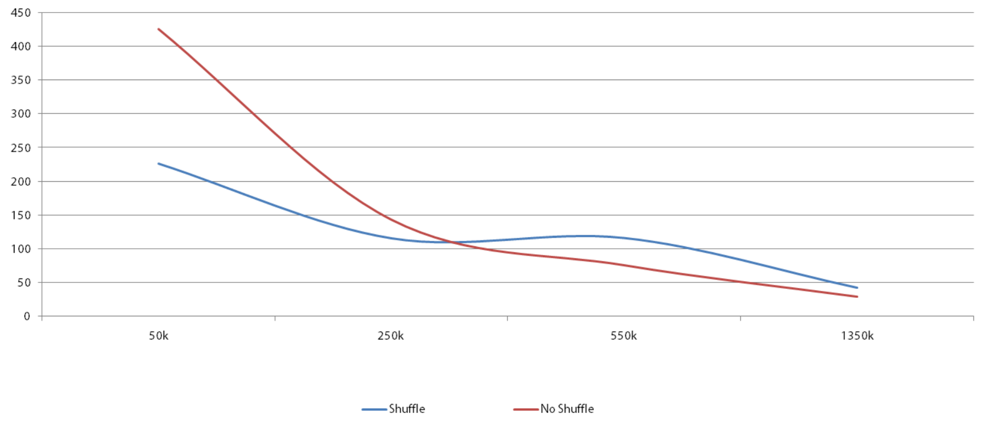


Figure 3.7: *Frame rates obtained with different molecules when the elements of the Shader Storage Buffers are shuffled and when they are not. Note that when the number of atoms is greater than 300K, the shuffled buffers obtain better performance due to the reduction of collisions in the voxel access.*

However, this could not be enough in some cases: Being far in the backbone does not guarantee the atoms are far in the scene too. Two residues far away in the backbone can be close in the space due to the protein folding. A more sophisticated method could be used to solve this issue in which the spatial relationship between atoms determines the final ordering. Nevertheless, we found our solution enough to reduce the impact of collisions in the performance. The plot in Figure 3.7 illustrates the frames per second obtained for molecules with a different number of atoms.

The plot shows that when the number of atoms is greater than 300K, the shuffled buffers obtain a better performance than the regular ones. When the number of atoms is high, not all the atoms can be processed in parallel in the GPU, so the buffer is processed in chunks. Shuffling the buffer reduces the collisions inside these chunks. On the contrary, when the number of atoms is low, fewer chunks have to be processed which leads to a high number of collisions. Moreover, this method has another potential drawback: the coherence in the texture access for each compute shader workgroup is reduced, increasing thus the texture cache misses. Based on the results of our experiments, we shuffle the buffers for molecules with more than 300K atoms.

This algorithm allows us to fill the Occupancy pyramid every frame for molecules up to 1350K atoms while maintaining a real-time frame rate.

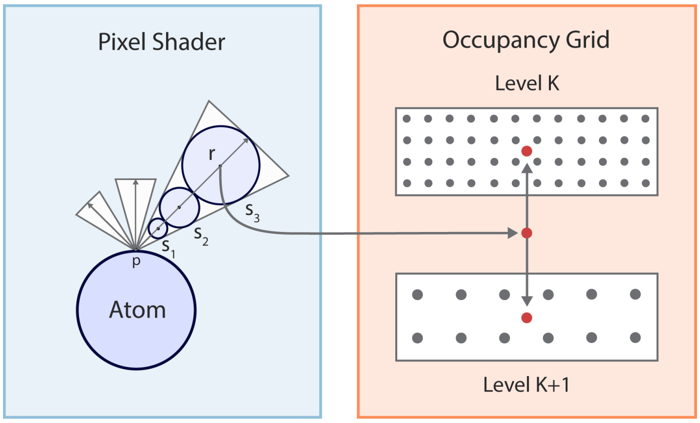


Figure 3.8: *In order to evaluate AO, we trace rays along the main direction of a set of cones that cover the space above the shading point. For each ray, we sample a set of points $s_1, s_2, ...s_n$ and the ambient occlusion factor is obtained by querying the occupancy pyramid at these points. The radius of the inscribed spheres determines the LOD to use when querying the occupancy pyramid.*

3.1.3.3 Volume-based ambient occlusion

As discussed in the section 3.1.1, ambient occlusion is an algorithm that approximates the global illumination by measuring the amount of light that may reach each surface point. The algorithms used to compute it can be divided into several categories. The most commonly used algorithms in real-time applications compute the visibility of each point based only on additional information stored for each pixel, dismissing thus 3D information of the scene. Despite having good performance, these algorithms do not take into account all the scene to compute the visibility, which generates artifacts and incorrect shadow estimations. Other techniques compute the occlusion based on the real geometry of the scene achieving a high-quality shading, although they have a high computational cost. Our algorithm, instead, uses a volume-based simplification of the scene to approximate the visibility of each point, obtaining a good trade-off between performance and image quality. Our algorithm is based on the work proposed by Crassin et al. [CNS+11] since both techniques use a volume-based simplification of the scene that, later, is sampled using *Voxel Cone Tracing*.

After the Occupancy pyramid is computed, the algorithm renders a quad covering the screen. Then, for each pixel, the depth and the normal of the pixel are recovered from the G-buffer filled in the first step (Figure 3.4). From the depth value the 3D world position is reconstructed, and, together with

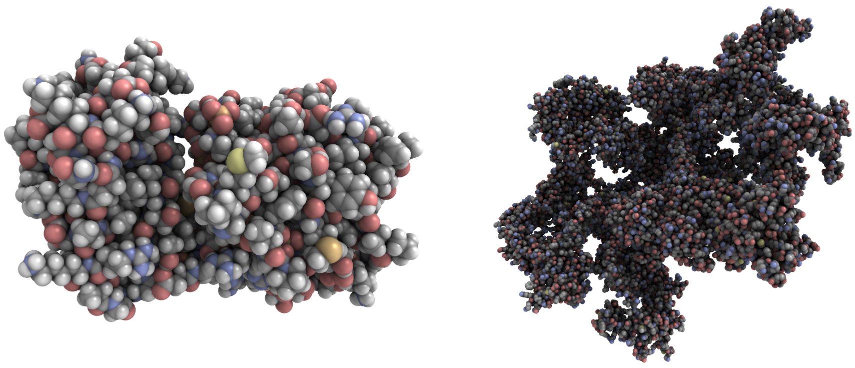


Figure 3.9: *Molecule A with 3.9K atoms (left) and molecule 3J3A with 46K rendered with the ambient occlusion presented in this section.*

the normal, the ambient occlusion factor is approximated using Voxel Cone Tracing. Voxel Cone Tracing approximates the visibility of the hemisphere centered at the point of interest by using a set of cones, as it is illustrated in Figure 3.8. In our implementation, we used three, five or nine cones to cover the hemisphere, depending on the desired quality/performance. For each of these cones, we compute the center and radius of a set of tangent spheres that fit inside the cone. We tested different configurations, and four spheres cover enough distance to have a high shadowing quality. We sample the occupancy pyramid at the center of each sphere, accessing the level of the hierarchy with a voxel size equal to the sphere radius. Since the sphere radius usually does not coincide with any voxel sizes in the hierarchy, a trilinear interpolation between the two closest levels is performed to have a good approximation of the occupied volume of the sphere. Then, we accumulate the occupancy along the ray using the following equation to determine the percentage of indirect light rays that arrive at the point from this set of directions:

$$ray_o += f * sph_o * (1.0 - ray_o) \tag{3.12}$$

where ray_o is the occlusion accumulated along the ray (which is initialized to 0), sph_o is the occlusion of the sphere and f is a scaling factor used to adjust the resulting occlusion. As discussed before, this algorithm overestimates the incident volume of the elements into the voxels. Furthermore, it does not take into account the overlap between atoms in the Space-filling model. In order to parametrize the introduced error, we scale the occupancy of the tangent spheres by the factor f, which decreases with the sphere radius. As more volume is approximated, more error could have been introduced. This parameter was

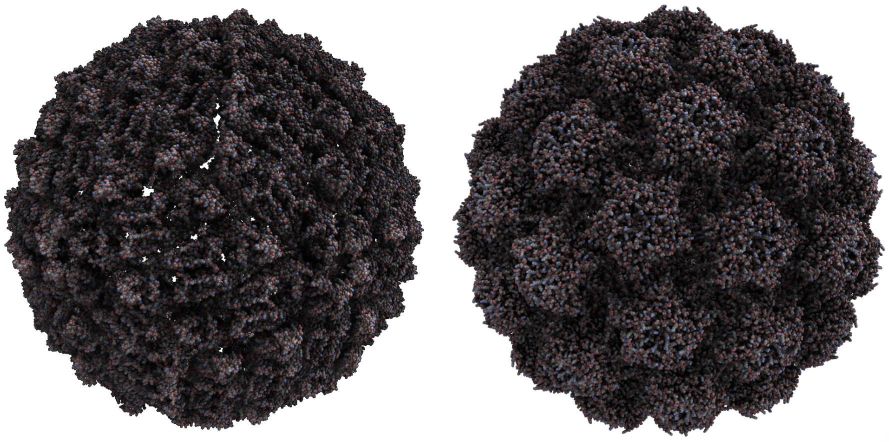


Figure 3.10: *More examples of molecules rendered with our ambient occlusion algorithm: molecule 1K4R with 545K atoms on the left and molecule 3IYJ with 1356K on the right.*

initialized to [1.0, 0.9, 0.85, 0.8] for the four spheres of each ray, but they can be adjusted to obtain different shading effects. The final occlusion factor is then averaged between all the cones.

The final composition step combines the color with the ambient occlusion and calculates the lighting using the information stored in the G-buffer generated by the first step. Note the high-quality ambient occlusion obtained, and how close to the ground truth our algorithm is when it is compared to an image generated using path tracing (Figure 3.11).

3.1.4 Results

In this section, we present the results obtained by the presented algorithm. It was implemented and tested on an Intel Core i7 PC, running at 3.5 GHz, with 16 Gb of RAM, and a GeForce 770 GTX. The viewport size used for all these tests was 1280 × 720.

The algorithm presented in this chapter was used to render molecules with a different number of atoms using both, Space-filling and Balls & Sticks representation models. These molecules were obtained from the *Protein Data Bank* and simulations carried out by our collaborators at the *Barcelona Supercomputing Center*. As Table 3.1 shows, we obtained more than 30 fps in all the tests, even for the largest tested molecule which has more than 1.3M atoms (computing the occupancy pyramid at every frame). Since our algorithm performs expensive tasks in the pixel shader, we decided to evaluate the performance

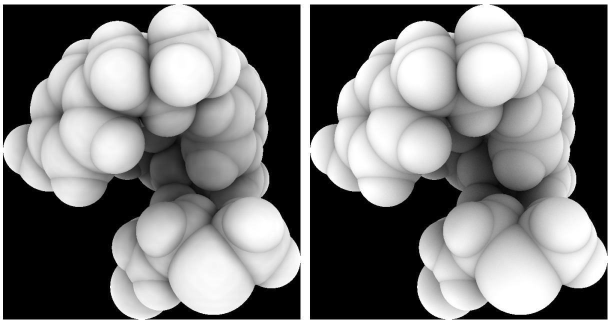


Figure 3.11: *Comparison of our ambient occlusion algorithm (left) with the ground truth, computed using path tracing [Kaj86] (right).*

using three different configurations: In the first one the molecule fills the whole viewport (column headed by the letter N in Table 3.1), in the second one the molecule is at a medium distance from the camera (column headed by the letter M in Table 3.1), and, in the last one, the molecule is far away from the camera (column headed by the letter F in Table 3.1). Performing these three tests, not only we were able to determine the performance of our algorithm in the worst case scenario, but also we were able to identify when the bottleneck of the algorithm was the pixel shader and when it was the compute shader. If the frames per second obtained for the three configurations are substantially different, it indicates that the main cost of the algorithm is in the pixel shader. On the contrary, when the performance is similar for all of them, it indicates that the number of pixels rendered does not determine the cost of the algorithm. This is the case of the molecules A and 3IYJ in Table 3.1. Molecule 3IYJ has a large number of atoms, which makes the compute shader execution expensive. Molecule A, on the other hand, has only a few atoms, but every atom covers a high number of voxels, which makes the compute shader execution also expensive.

Besides measuring the performance of the algorithm, Table 3.1 also evaluates the impact of shuffling the elements in the Shader Storage Buffers. This comparison is presented in the rows headed by S (shuffled buffer) and NS (Non-shuffled buffer). As discussed in Section 3.1.3, shuffling the elements of the buffers is only useful when the GPU is not able to process in parallel all the elements of the scene (they are processed in chunks). Table 3.1 shows that when the number of elements in the scene increases, the shuffled buffers obtain better performance than the regular ones (Molecules 1K4R and 3IYJ in Table 3.1).

Images of the molecules used to get the measurements can be found in the

Table 3.1: *Performance measured in frames per second for different molecules (N)ear the camera (occupying the whole viewport), at (M)id-distance, and (F)ar from the molecule. Ambient occlusion has been computed at the highest quality (nine cones per point). "S-F" indicates usage of the Space-filling render mode, while "B&S" indicates usage of the Balls & Sticks render mode. Frame rates have been measured without shuffling the buffer of elements (NS) and with the shuffle applied (S).*

	Molecule (#atoms - [Figure])	AO	Vertex Sorting	N	M	F
S-F	A (3.9 K - Fig. 3.9)	No AO	-	816.52	1,953.11	3,444.91
		AO	S	188.19	223.36	236.02
			NS	**201.22**	**241.46**	**257.51**
	3J3A (46 K - Fig. 3.9)	No AO	-	547.88	1,233.95	1,491.38
		AO	S	171.53	216.79	226.86
			NS	**265.45**	**393.55**	**425.65**
	1CWP (227 K - Fig. 3.12)	No AO	-	169.74	382.59	499.96
		AO	S	76.98	120.51	115.98
			NS	**89.16**	**130.62**	**143.48**
	1K4R (545 K - Fig. 3.10)	No AO	-	136.66	192.29	456.49
		AO	S	**70.83**	**85.38**	**116.42**
			NS	53.58	61.57	76.01
	3IYJ (1,356 K - Fig. 3.10)	No AO	-	64.72	88.68	96.95
		AO	S	**34.09**	**38.95**	**42.51**
			NS	25.44	28.03	29.46
B&S	A (3.9 K - Fig. 3.9)	No AO	-	2,255.40	3,104.45	3,705.44
		AO	S	671.81	779.95	852.78
			NS	**706.51**	**840.02**	**907.55**
	3J3A (46 K - Fig. 3.9)	No AO	-	1,156.75	1,504.23	1,858.25
		AO	S	**345.12**	**402.41**	**434.37**
			NS	342.05	398.52	431.88
	1CWP (227 K - Fig. 3.12)	No AO	-	351.47	527.38	600.48
		AO	S	**123.88**	**148.24**	**157.04**
			NS	92.53	105.86	109.98
	1K4R (545 K - Fig. 3.10)	No AO	-	209.63	275.68	276.19
		AO	S	**64.6**	**71.63**	**72.14**
			NS	34.36	36.21	36.41
	3IYJ (1,356 K - Fig. 3.10)	No AO	-	88.23	107.19	108.08
		AO	S	**30.63**	**33.08**	**33.27**
			NS	14.83	15.35	15.42

Table 3.2: *Impact of the number of cones used to compute the ambient occlusion on the rendering speed in fps. The same molecule A from Figure 3.9 was rendered in Space-filling and Balls & Sticks modes using 3, 5 and 9 cones per fragment.*

		9 cones	5 cones	3 cones
Space-filling	near	201.22	204.45	206.28
	mid.	241.46	243.16	243.65
	far	257.51	258.19	258.28
Balls & Sticks	near	706.51	772.49	783.12
	mid.	840.02	882.29	886.01
	far	907.55	939.53	941.86

Figures 3.9, 3.12 and 3.10. In Figure 3.13 only the ambient occlusion factors are illustrated, for both the Space-filling and Balls & Sticks models.

All the frame rates measurements in Table 3.1 are taken using nine cones to approximate the hemisphere visibility at each point. Table 3.2 shows the results obtained varying the number of cones (3-5-9) for the same molecule. From the table, we can conclude that the variation of the number of cones has a little impact on the final cost of the algorithm. Although it may seem that extra texture accesses would highly increase the cost of the algorithm, the locality of our accesses and the hardware cache make it almost negligible.

In order to evaluate the correctness of the ambient occlusion factors computed with our method, we compared the results to that obtained using a path tracing algorithm. The results of the comparison are shown in Figure 3.11. Note that the differences are hardly visible. While our method yields slightly higher occlusions in marked creases, the shadings are smooth and coherent with the shape, offering correct depth cues and adequate high-frequency detail.

Besides comparing the results obtained by our algorithm with the ground truth, we also compared our method with a state-of-the-art method. We compared our method with the one proposed by Grottel et al. [GKSE12] since they also use a voxelization of the scene to approximate the occlusion factors. As it is illustrated in Figure 3.12, our method is capable of handling high and low-frequency occlusions at the same time due to the hierarchical nature of the voxelization. On the other hand, Grottel et al.'s method only takes into account low-frequency occlusions, which is translated into a wrong shape perception. Furthermore, Grottel et al.'s method was designed to generate occlusions in scenes composed of billions of particles, in which each atom can be approximated by a single point without introducing a significant error. In

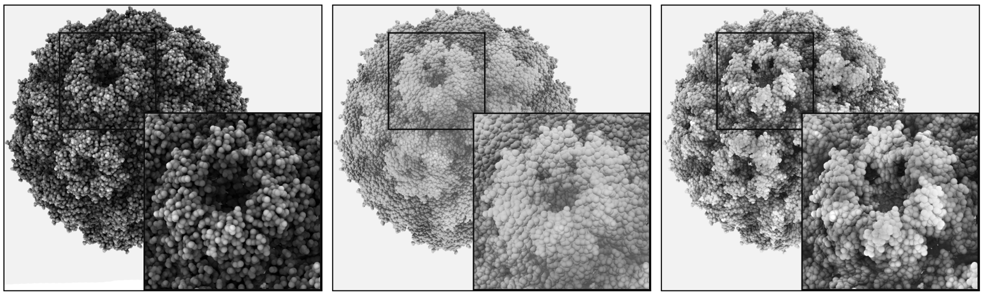


Figure 3.12: *Comparison of our ambient occlusion algorithm (left) with the algorithm proposed by Grottel et al. [GKSE12] with a resolution of* 8^3 *(middle) and* 16^3 *(right).*

our case, on the other hand, where molecules have fewer atoms (the molecules commonly used in our pharmacology simulations rarely exceed 60-80K atoms), this assumption leads to wrong occlusion estimation.

3.1.5 Conclusions

This section has presented a novel approach to enhance the depth perception in molecular visualization by using ambient occlusion. The presented algorithm is capable of approximating the ambient occlusion effect in real-time even for complex scenes composed of hundreds of thousands of atoms. All the computations are carried out on the GPU using a GPU-friendly data structure, which allows us to use hardware capabilities as the texture cache. The method was tested on different molecules obtained from the *Protein Data Bank* and on real molecular dynamics simulations, obtaining high-quality images in all cases. Moreover, our method was compared with the ground truth and with a state-of-the-art method, which demonstrated that our method provides a high-quality ambient occlusion closer to the ground truth than previous approaches in real-time.

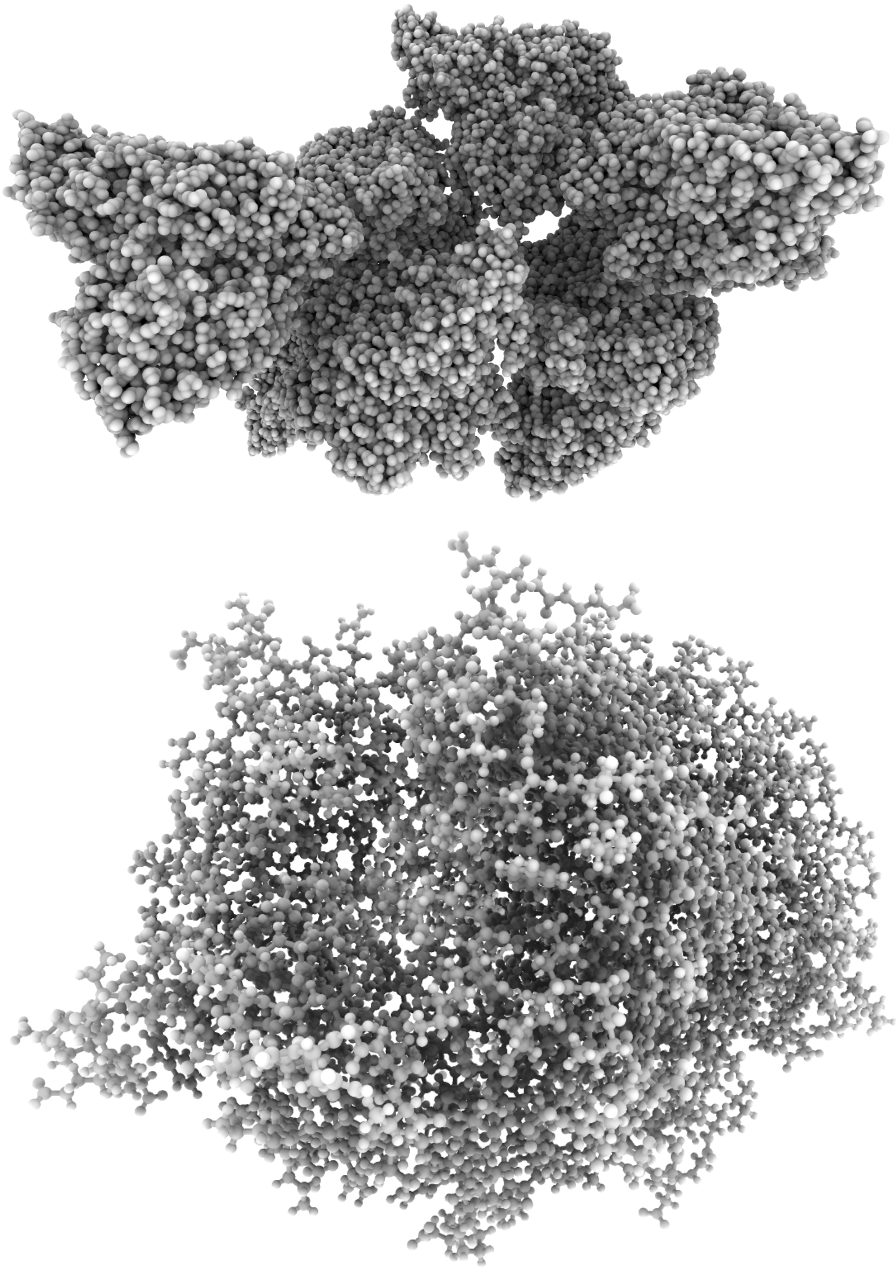


Figure 3.13: *Images rendered with only the ambient occlusion factors, using the Balls & Sticks model (bottom) and the Space-filling model (top).*

3.2 Realtime halos

Pharmacology drug design simulates the docking of a small molecule (the ligand) into a protein. The simulation software generates hundreds of trajectories where the ligand is trying to find the docking position moving around the protein. Experts, then, inspect those trajectories that achieve a better docking (trajectories where the system achieves the lowest energy) using a molecular visualization package. These tools allow them to explore the frames of each trajectory using different representation methods. However, selecting the same representation methods for the ligand and the protein, or selecting similar ones with the same color coding, could make difficult to identify the ligand in the scene. Moreover, experts are interested in visualizing the areas where the ligand tried to dock or the pathway the ligand followed to reach the docking position. One possible solution could be visualizing the ligand on all the frame positions, but in most of the cases, it will generate a cluttered image. Bidmon et al. [BGB+08] and Furmanová et al. [FJB+17] have presented different methods to visualize the ligand position during long trajectories. In both cases, the number of frames was simplified and they used the center of the ligand on the remaining ones to generate a set of lines. Hence, despite giving a good overview of the areas where the ligand was trying to dock, those approaches only use a subset of the ligand positions.

Illustrative visualization techniques are focused on highlighting specific fea-

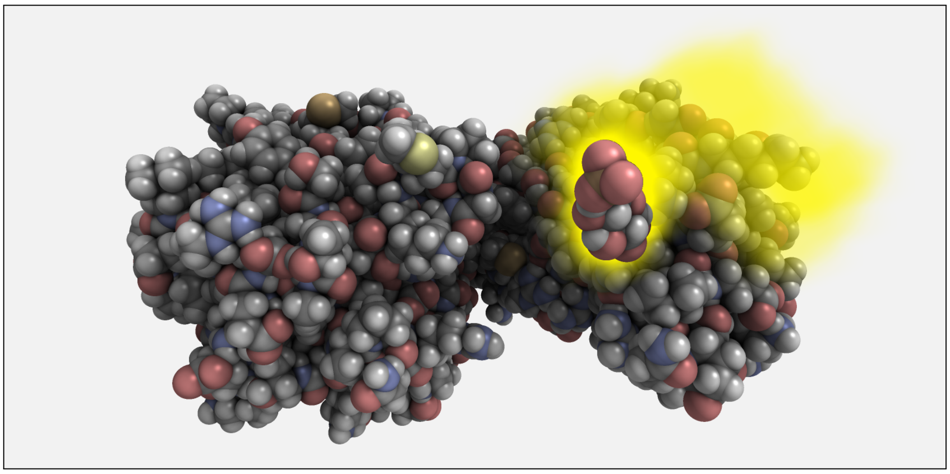


Figure 3.14: *Image of the trajectory of a ligand highlighted with our technique to render temporal halos. In this image, the attenuated halo shows the regions recently visited by the ligand in the simulation.*

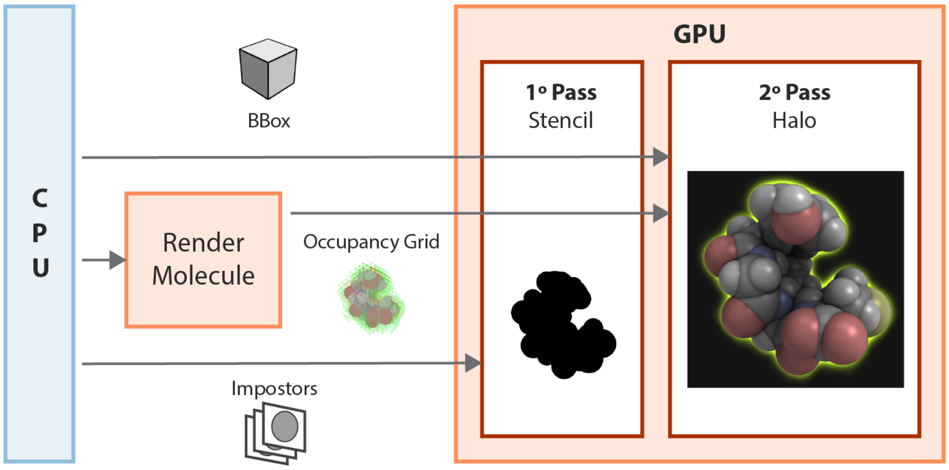


Figure 3.15: *Pipeline of the halo algorithm. Once the scene is rendered using the pipeline described in Section 3.1 the molecule which generates the halo is rendered to the stencil buffer. In a second pass, the halo is computed in the non-modified pixels of the stencil buffer using the occupancy pyramid computed by the ambient occlusion algorithm.*

tures of the model (on Section 3.1 we used ambient occlusion to improve the understanding of the shape of the molecule), which makes them the perfect candidates to improve the inspection of the trajectories during the drug design process. Halos were successfully used by Tarini et al. [TCM06] to highlight the shape of molecules. However, they were limited to orthographic projection. This chapter introduces an algorithm to generate halos around a group of atoms represented using Space-filling or Balls & Sticks models, which help the experts to identify the ligand within the 3D view. Moreover, it also presents an algorithm to generate temporal halos, which highlights the areas recently visited by the ligand. These techniques are efficiently integrated into the graphics pipeline described in Section 3.1. As an example of the results obtained, Figure 3.14 illustrates both techniques applied to a trajectory.

3.2.1 Overview

The algorithm presented in this section is integrated into the pipeline described in Section 3.1. The same algorithm used to compute the Occupancy Pyramid in the ambient occlusion algorithm is used here to compute an extra occupancy pyramid, which is filled out only with the volume occupied by the atoms we are interested in highlighting. This new occupancy pyramid is used then to generate the halo in two extra passes (see Figure 3.15):

- First, the molecule we want to highlight is rendered in the stencil buffer.
- Then, for the pixels not marked in the stencil buffer, the halo is computed using a ray-marching algorithm.

In the following section, we describe this process in detail.

3.2.2 Halos

Halos can be easily rendered for the whole molecule using the same data structure defined in Section 3.1. However, their utility becomes prominent when rendered around the ligand, since this helps researchers to clearly and quickly identify the ligand and distinguish it from the protein. Therefore, we add a second copy of our data structure that stores the occupancy pyramid of the drug only. Since the drug is smaller than the protein, this data structure can be of lesser resolution. However, for simplicity, we keep it with the same dimensions as the original occupancy pyramid used in AO computations for the whole molecule. This new data structure is filled out together with the previous one during the ambient occlusion computation by the same compute shaders, but, because the number of image bindings to a shader is limited, we can only update the first three levels of this new occupancy pyramid (128^3, 64^3 and 32^3). Notice that the computational cost added by the new occupancy pyramid consists only in some extra writing accesses when the atoms tested belong to the ligand.

Once the image is composed by the regular pipeline, the halo is computed and added to the final image by two extra rendering passes. The first one renders the procedural impostors of the molecule on focus (the drug) into the stencil buffer, which prevents us from generating halos over it. These impostors are rendered using the existing depth buffer from the previous steps, so, in order to avoid discarding pixels by the depth test, we render the procedural impostors using a depth offset. Moreover, since we are only interested in identifying the pixels where the molecule is projected and not in computing the lighting, we can skip the computation of the normals in the pixel shader.

The second step computes the final halo intensity of each pixel and merges the result with the final image. In order to do so, the bounding box of the ligand is sent through the pipeline where only the back faces are rasterized. For each pixel not marked in the stencil buffer, the algorithm traces a ray from the back face towards the camera. If the depth value stored at the G-Buffer (generated at the beginning of the pipeline) is closer than the rasterized back face, this

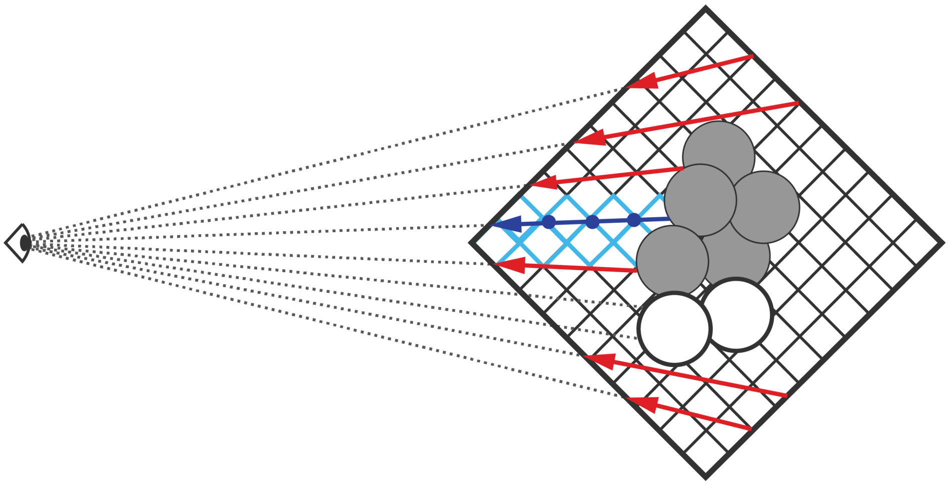


Figure 3.16: *Computation of the halo intensity. For each pixel, a ray is traced towards the camera and the occupancy grid is sampled uniformly while its occupancy values are accumulated. The resulting accumulated occupancy is then used as the halo intensity value.*

depth is selected as the ray starting point. Each ray traverses the occupancy pyramid until it reaches the front faces of the bounding box. The occupancy values stored in the voxels of the occupancy pyramid are accumulated along the ray, sampling the data structure using trilinear interpolation at equidistant sampling points. The result, then, is used as the alpha channel of the halo color which determines its transparency. This process is illustrated in Figure 3.16. The halo can be adapted by the user to each scene by selecting its intensity and size. The intensity of the halo can be adapted by modifying a user defined variable, which scales the final alpha component of the halo color. The halo size, instead, is determined by the LOD level selected during the access to the data structure. As a higher level in the hierarchy is selected, a bigger halo is generated. Notice that with this algorithm the halo is calculated using 3D information and it can be rendered even if the molecule to highlight is hidden in the scene.

3.2.3 Temporal halos

As discussed before, the pathway a ligand followed is crucial to the analysis of a simulation. This section introduces a new technique to visualize this pathway using a continuous halo around the ligand.

The algorithm to generate halos described in the previous section uses an occupancy pyramid to generate halos around the ligand. This data structure is cleared at the beginning of every frame, and it is computed again for the new configuration of the scene. Instead of clearing it, we accumulate the occupancy values along different frames, keeping the areas visited by the ligand during the simulation in our data structure. By tracing rays through the data structure, as we did to generate the halo around the ligand in the previous section, we are able to visualize all the places visited by the ligand as a continuous halo. Instead of using the bounding box of the ligand to trace the rays, we use the bounding box of the whole scene, generating rays over the whole data structure.

This technique illustrates the ligand pathway effectively, but it can generate cluttered images for long trajectories. In order to reduce the amount of information visualized, we slowly vanish the halo along a time frame T. At the beginning of every frame, a compute shader reduces the occupancy of each voxel (for the three levels of the occupancy pyramid) according to the time elapsed between frames. The new occupancy is computed using the following equation:

$$O_f = O_i - max\left(\frac{t - t_o}{T}, 1\right) \tag{3.13}$$

where O_i is the initial occupancy of the voxel, t is the current time, t_o is the time of the last frame and T is the desired duration of the halo persistence. Figure 3.17 shows this technique applied to a simulation.

3.2.4 Results

This section describes the results obtained by applying the proposed halo technique to different scenes. All the tests were performed on an Intel Core i7 PC, running at 3.5 GHz, with 16 Gb of RAM, and a GeForce 770 GTX. The viewport used in these tests had a size of 1280 × 720.

We used a temporal halo to highlight the pathway of the ligand along a real simulation generated by our colleagues (Figures 3.14 and 3.17). Table 3.3 presents the frames per second obtained by using a halo and a temporal halo in this simulation. Both algorithms were tested modifying the projection size of the molecule on screen. Despite always having real-time performance, our algorithm has a relatively high impact on the rendering speed, as our ray-marching algorithm does not use any optimizations (e. g. proxy geometry could be used to reduce the number of rays).

Moreover, we used halos within our application not only to emphasize the position of the ligand but also to highlight the selection of any group of atoms. In Figure 3.18 a chain of a molecule is accentuated rendering a halo around it.

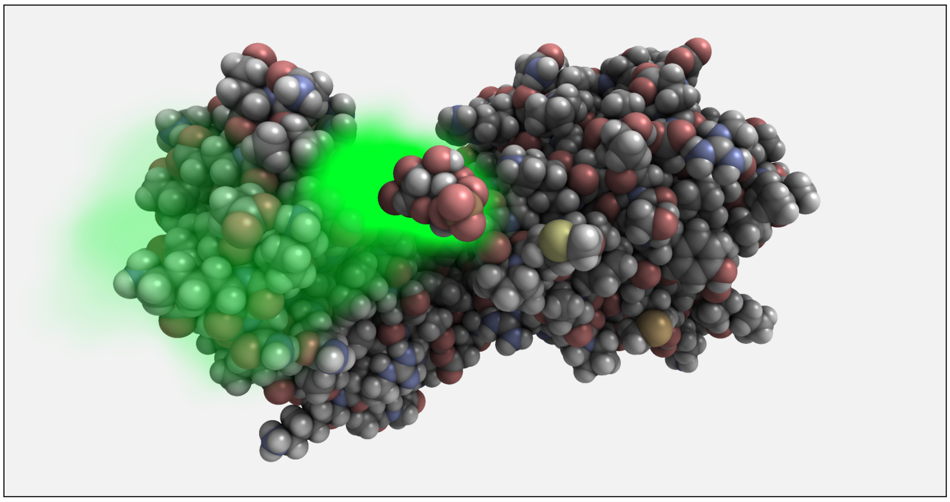


Figure 3.17: *Ligand path highlighted using temporal halos. The image illustrates how the halo intensity fades along the time, leaving a low-intensity halo around the first visited areas.*

Table 3.3: *Impact of halos on rendering speed. The same molecule (see Figure 3.14) is rendered with no halos, a halo around the ligand, and a halo around the larger molecule. Frame rates are reported both for the Space-filling and the Balls & Sticks render modes, with a* 1280 × 720 *viewport.*

		no halos	ligand	mol.
Space-filling	near	201.22	113.49	101.75
	mid.	241.46	126.68	122.91
	far	257.51	131.28	129.52
Balls & Sticks	near	706.51	484.77	328.78
	mid.	840.02	575.9	489.98
	far	907.55	621.83	604.08

Besides the performance tests, we also carried out a small user study to assess the utility of the halos and the ambient occlusion. We prepared a questionnaire that was answered anonymously by 11 domain experts, all of them working with drug simulations daily. We asked them to rank in a 1...7 Likert scale (where one is either "totally agree" or "totally preferred" and seven is "totally disagree") their preference with respect to five questions, summarized in the following listing. The brackets indicate the contents of the pictures we showed to the users when asking the questions.

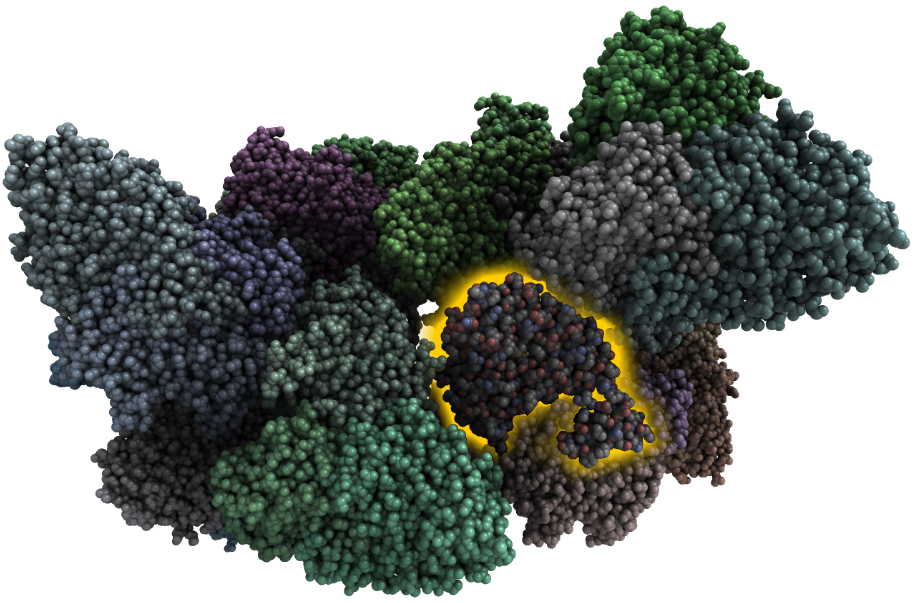


Figure 3.18: *Halo used to highlight a chain of a protein complex. The attention of the user is driven to the selected chain thanks to the bright halo color surrounding the atoms.*

1. [AO image] is better to understand the rendering [than an image without AO].
2. If you could always use [AO] in your visualization, you would do it.
3. [Image with a halo around a drug] helps better understand the drug position [than the image without a halo].
4. [Halos] are helpful to visualize simulations.
5. [Image with a halo path] helps to understand the path covered by the molecule.
6. The resulting image [image with a halo path] is useful.

The plot in Figure 3.19 shows the results obtained in the user study. For all the answers we obtained a mean value below 3 and the 3rd quartile below 3,5. Despite having some answers greater than 5, the results of the test show that the experts appreciate the quality of our rendering techniques and they

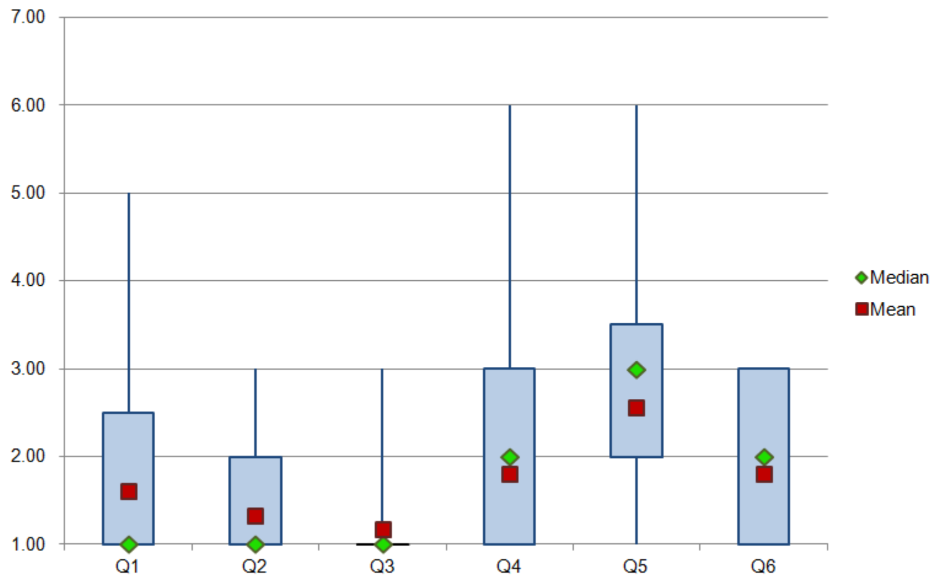


Figure 3.19: *Results of the user study. The user answered 6 questions (where 1 is "totally" agree and 7 is "totally disagree") quantifying the quality and utility of the images generated by our ambient occlusion and halo techniques. The results show that experts think these techniques can improve their task on drug design analysis (all the answers have a mean and median below 3).*

think these techniques could improve the analysis of drug design simulations as the understanding of the conformation of a molecule.

3.2.5 Conclusions

We have introduced a new technique to highlight the ligand, or any group of atoms in a molecule, through rendering a halo around them. This technique uses the same data structure used to compute the ambient occlusion in Section 3.1, which make it easily integrable within our pipeline. Moreover, the halo is computed using 3D information of the scene, making it stable against occlusions or fast movements of the camera.

Furthermore, we have extended this algorithm to support temporal halos. Them, not only help to highlight the ligand but it also illustrates the pathway the ligand has followed to reach the current position.

The utility of this technique, together with the ambient occlusion algorithm, was tested through a small user study, which revealed that domain experts find them helpful to analyze their simulations.

3.3 Realtime secondary structures generation

Ribbons is an abstract visualization technique commonly used by experts to understand the underlying conformation of a protein since only the structural information of the molecule is presented. The backbone chain of the biomolecule is rendered using a set of sheets and tubes, depending on the pattern created by the chain. These patterns are widely repeated among biomolecules, and the most common are α-helix and β-sheet. The first one, the α-helix, appears when two or more close amino acids in the chain create a hydrogen bond between them, twisting the backbone into a helix. β-sheets, on the other hand, are formed when a set of consecutive amino acids in the backbone create hydrogen bonds with another set of consecutive amino acids far away in the chain. The formation of these secondary structures is briefly described in Section 1.2.

Most systems generate a 3D mesh to represent the secondary structures in a preprocess [HDS96], which is later used during rendering. This method is able to render molecules in ribbons mode at high frame rates, even for large scenes, but nevertheless, in some scenarios, the preprocess makes this method prohibitive. Inspecting molecular dynamics simulations in real-time, for example, requires an interactive visualization of the results, which implies generating the ribbons geometry in real-time for each step as soon as it is received by the visualization software. Under these conditions, a wiser method should be used. More recent techniques optimize this generation process dividing the computation between CPU and GPU [WB11] or generating all the geometry on the

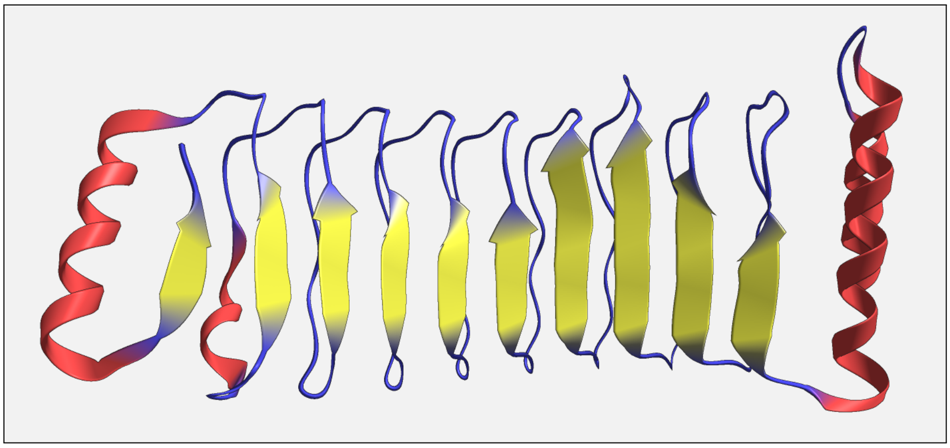


Figure 3.20: *Image of a molecule rendered in ribbons mode with our technique. The yellow arrows represent the β-sheet secondary structures and the red helices the α-helices.*

GPU [KBE08], reducing thus the overhead created by the data transmission between CPU and GPU. Despite these optimizations, they are not able to maintain a real-time frame rate when representing large molecules, due to the high number of triangles generated.

This section introduces a novel technique to generate and render the geometry of the ribbons representation model using the GPU, being able to create only the necessary triangles for a given point of view. Figure 3.20 presents a molecule rendered using this technique. Furthermore, this section describes how the ambient occlusion algorithm presented in Section 3.1 can be adapted to generate this effect on the ribbons model, increasing thus the understanding of the shape of the scene.

The rest of the section is organized as follows: Section 3.3.1 describes the algorithm used to generate and render the geometry. Section 3.3.2 presents how the algorithm was adapted to the GPU architecture. Section 3.3.3 explains how the ambient occlusion is generated. The results obtained, together with comparisons against the existing methods, are shown in Section 3.3.4. And, lastly, the conclusions are given in Section 3.3.5.

3.3.1 Algorithm

The ribbons representation method is focused on visualizing the secondary structures formed by the backbone chain of a biomolecule. As discussed in Section 1.2, these secondary structures are common patterns found in the 3D arrangement of amino acids. The ribbons representation method uses a set of different motifs to visualize these structures. The β-sheet secondary structures are represented by a sheet that covers all the amino acids involved in generating the secondary structure. Commonly, an arrow is added at the end of the sheet to communicate the direction of the backbone (see yellow sheets in Figure 3.20). The α-helix secondary structures are represented by a sheet too, but here, the sheet is folded acquiring a helix shape (see red helix sheets in Figure 3.20). The parts of the backbone which do not form any secondary structure pattern are represented by a tube, driving away the attention to more relevant areas (see blue tubes in Figure 3.20).

All the motifs used to represent the different secondary structures are aligned to the backbone, i. e. they follow the backbone direction. In order to have a smooth secondary structure representation, Carson [Car91] proposed a method which used a B-Spline (generated from the 3D positions of the atoms) as a guide for the secondary structure motifs. To develop our technique, we adopted his approach. He defined the B-Spline using the 3D positions of the C_α atoms of each residue as control points, generating thus a segment of the

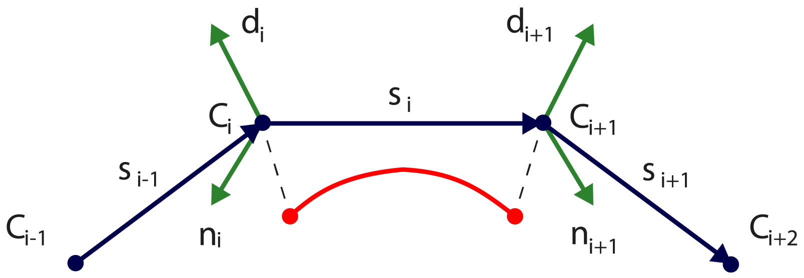


Figure 3.21: *A segment of the B-Spline. Control points are illustrated in blue, the vectors used to align the sheet in green, and the interpolated B-Spline segment is illustrated by the red curve.*

B-Spline for each pair of consecutive amino acids (see Figure 3.21). Notice that the first and last control points have to be repeated in order to generate the first and last segments of the B-Spline. The motifs are then generated on the B-Spline and oriented to illustrate the direction of the hydrogen bonds. As in [KBE08], we use, instead of the hydrogen bond direction, a vector d to orient the sheets, which is defined by the direction between the C_α and the O atoms of each residue. Despite being defined different, they are virtually equal and d can be easily computed from the atoms of each amino acid [CB86]. Two consecutive amino acids in the backbone usually have their side chain oriented to opposite directions, which also makes d vectors point in opposite directions. To have smooth changes of d along the backbone, we compare the d vector of every amino acid against the previous one, and if the angle between them is greater than 90°, we flip d. Moreover, for each control point, we compute a second vector, n, perpendicular to d and the segment direction. To avoid discontinuities at every control point C_i, the n vector is calculated using the direction of the segment $i-1$ and the segment i as described by the following equation:

$$\begin{aligned} savg_i &= \frac{s_i + s_{i-1}}{|s_i + s_{i-1}|} \\ n &= \frac{(d \times savg_i)}{|(d \times savg_i)|} \end{aligned} \tag{3.14}$$

where s_i and s_{i-1} are the directions of the two segments to which the control point belongs. d and n are then used to extrude the geometry of the secondary structure motifs from each B-Spline segment.

One of our main goals during the design of the algorithm was to increase the speed of the existing methods, so a unified algorithm to generate the geometry

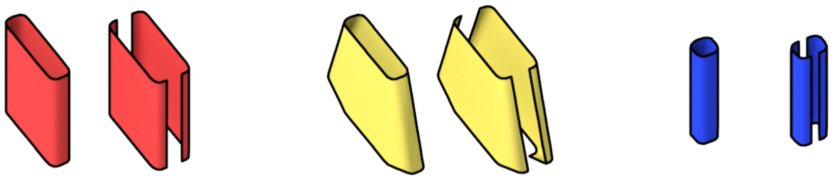


Figure 3.22: *Geometry used to represent backbone segments which belong to different secondary structure type. α-helix on the left, β-sheet on the center, and simple backbone segment on the right.*

of every secondary structure type at the same time was mandatory. Furthermore, we wanted to have smooth transitions between consecutive amino acids belonging to different secondary structures, allowing us to continuously illustrate the formation or destruction of a secondary structure block during a simulation.

In order to achieve that, our algorithm uses two quadrilateral patches to represent each backbone segment (see Figure 3.22). Each patch is aligned to the backbone segment and oriented using the vectors d and n. Then, using the distance towards the camera, the patches are subdivided and triangulated. Since distant patches have a small projection on the screen, we use a low number of triangles to represent them. Close patches, on the other hand, have a big projection on the screen. Therefore, they are subdivided using a higher number of triangles. The positions of the new vertices of the patches are obtained first by interpolating the control points using a B-Spline. Then, the vectors d and n are interpolated too and used to displace the new vertex as is illustrated in Figure 3.23.

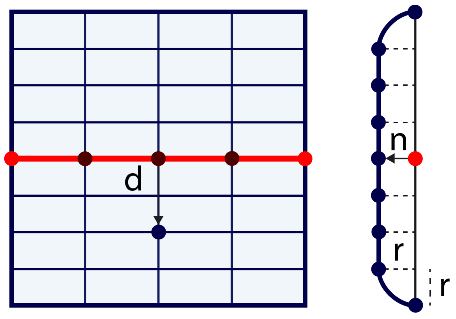


Figure 3.23: *Patch configuration used to obtain the ribbon representation. The vertices of the patch are displaced using the d and n vectors of the B-Spline segment.*

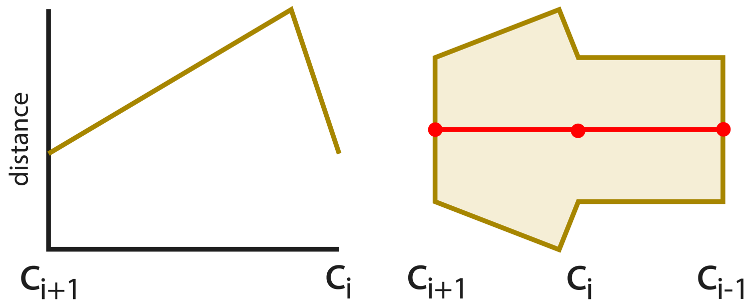


Figure 3.24: *Function used to determine the displacement of the β-sheets end segments. Using a triangle function the algorithm is able to generate arrows at the end of each β-sheet.*

This algorithm is able to generate the geometry of the α-helices, but we also need to generate the geometry of the β-sheets and the rest of the backbone. We need to generate the arrows at the end of the β-sheets and the tubes to represent the backbone. To achieve that, we introduced a little modification on the algorithm: we use a different distance to displace the vertex along the d vector for each type of secondary structure. Thus, we can generate sheets of different widths and, in the case of the backbone, we can generate tubes. These tubes are generated defining the displacements along the d vector equal to the displacement along the n vector, giving, as a result, a prism that represents the backbone (see Figure 3.22-right).

Generating arrows is a bit more complicated since the displacements along the d vector have to vary within the same patch. To do so, we use a triangle function to determine the displacement along d. Figure 3.24 illustrates the function plot and its relationship with the final geometry. In order to be consistent over the whole molecule, these arrows have to be rendered in the last segment of each β-sheet, so we have to identify them and use the linear function as the displacement of their vertices.

At this point, the algorithm is able to generate the geometry of each secondary structure type, but we still had to define how to generate those segments where each endpoint belongs to a different type of secondary structure. At these segments, a linear interpolation is performed in the colors and displacement distances used along d, creating a smooth transition between consecutive secondary structures (see Figure 3.20). Furthermore, this interpolation method allows us to smoothly illustrate the creation and destruction of these secondary structures along a simulation.

3.3.2 Implementation

We have implemented this algorithm fully on the GPU in order to meet our requirements: reduce the amount of information to transfer between CPU and GPU and generate geometry with an adaptive resolution for the current point of view. In this section, we describe how the tasks were distributed through four different stages of the graphics pipeline, in such a way that none of them require complex computations. An overview of the process is shown in Figure 3.25.

3.3.2.1 CPU

The main task of the CPU is to prepare the frame data, loaded from disk or received from the simulation software, for the GPU computations. Using the frame data, the CPU prepares two buffers, a vertex buffer and an index buffer. The vertex buffer contains all the B-Spline control points and the index buffer contains the indices of the B-Splines control points which each segment will use.

For each control point, we store the following information in the vertex buffer: the xyz coordinates of the C_α 3D position, the xyz coordinates of the d vector, and an integer that encodes the type of secondary structure to which the residue belongs. In the index buffer, for each segment, we store four indices pointing the four control points the segment will use to interpolate. Figure 3.26 illustrate the content of the buffers. Notice that the index buffer only has to be computed once for a whole simulation, since the backbone structure always remains the same. The vertex buffer, on the other hand, has to be updated for every new step of the simulation. The 3D positions of the atoms change in every single step and they have to be uploaded to the vertex buffer to represent the current conformation of the molecule.

The buffers are sent through the graphics pipeline, and the algorithm generates a patch for each segment of the backbone. However, in order to generate a correct representation of each segment, the algorithm should generate two patches with opposite orientations instead of only one (see Figure 3.22). Many approaches can be adopted to generate them: e. g. adding extra logic into the shaders or performing two draw calls to send the data twice through the pipeline. We, instead, duplicate the content of the index buffer and invert the order of the duplicated indices. The algorithm, then, will generate automatically the patches with the opposite orientation because the vector n, which defines the orientation of the patch, is computed using the direction of the segments. These new patches are added at the end of the index buffer.

Introducing these inverted indices, however, has a drawback. The sense of direction is lost. Before, the indices followed the direction of the backbone, but,

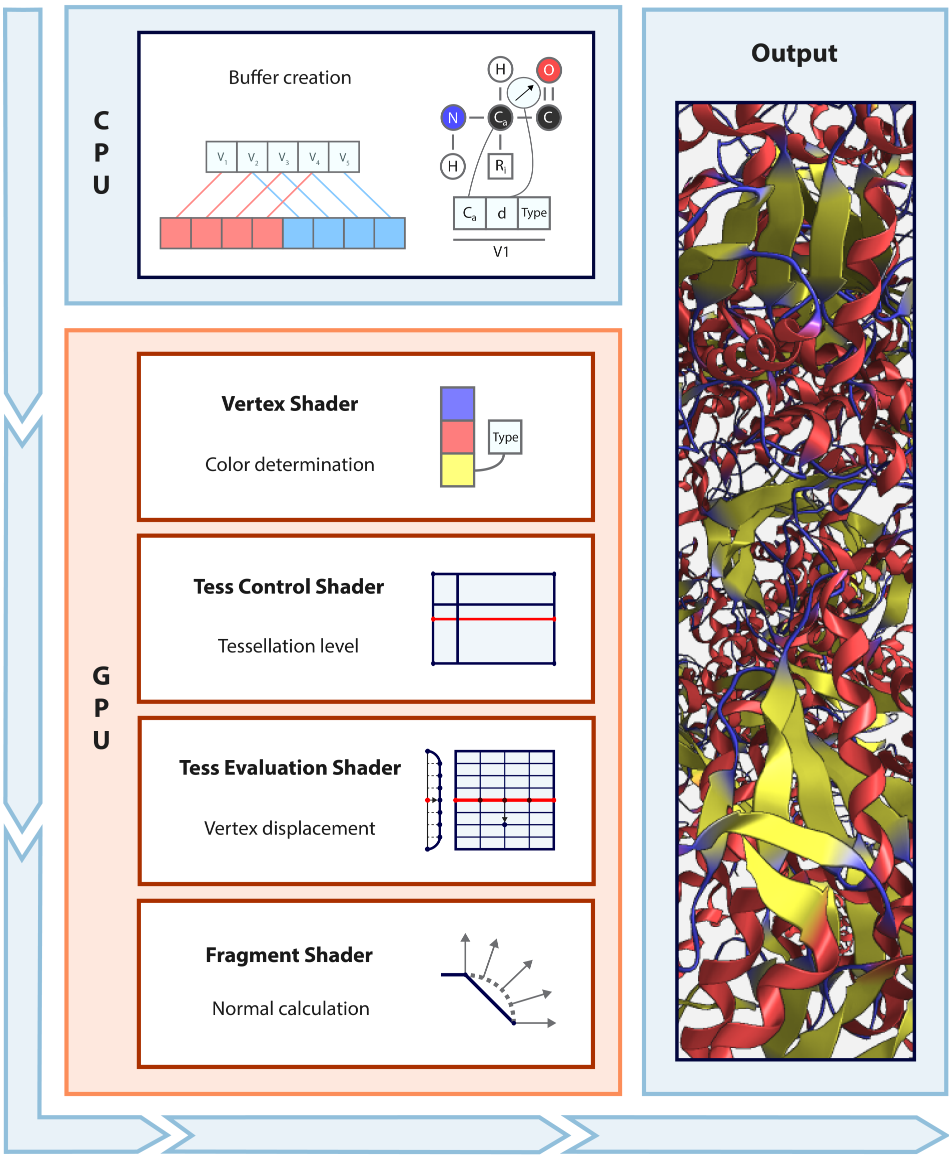


Figure 3.25: *Rendering pipeline: the load is balanced among the different stages of the algorithm. The CPU creates the data buffers, the vertex shader determines the colors, the tessellation stage is in charge of calculating the subdivision level required and performing the tessellation, and finally, the fragment shader recomputes normals to provide a better shading effect.*

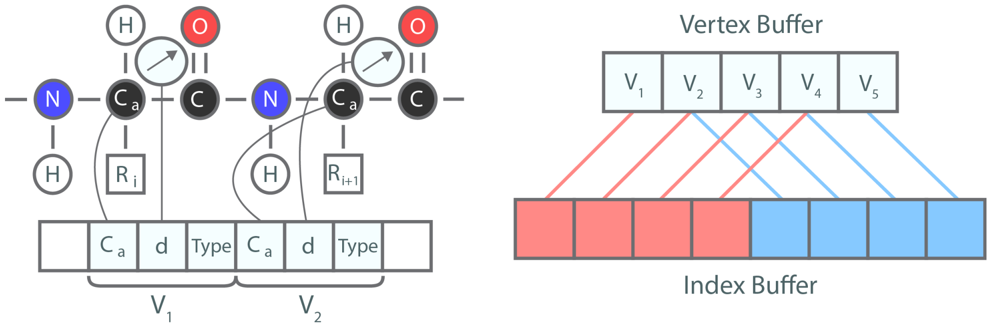


Figure 3.26: *Buffer creation. The CPU generates two buffers, a vertex, and an index buffer. The vertex buffer stores three parameters for each amino acid of the backbone: the 3D position of the C_α atom of the amino acid, the 3D coordinates of the d displacement vector (we use the direction given by the C_α and the O atoms) and the secondary structure type which the amino acid belongs. The index buffer stores the four indices of each backbone segment.*

now, half the buffer is following one direction and the other half is following the opposite one. This double orientation prevents us from identifying the end segments of the β-sheets where we ought to draw an arrow. However, the sense of direction can be easily recovered adding an extra float for each control point that increases its value along the backbone, so the shader can determine the original direction of the backbone and identify the last segment of every β-sheet.

These buffers are sent to the graphics pipeline where each segment is interpreted as a patch of four vertices.

3.3.2.2 Vertex shader

A vertex shader is executed for each vertex of the patch. The main task of this stage is to pass all the information to the next stage and determine the color used to render the patch. To do so, the vertex shader obtains the vertex color from a constant array defined in the shader using the secondary structure type as an index. The array holds the color chosen for each secondary structure type. The colors we have chosen to render the secondary structure types are red for α-helices, yellow for β-sheets, and blue for the rest of the backbone (see Figure 3.20).

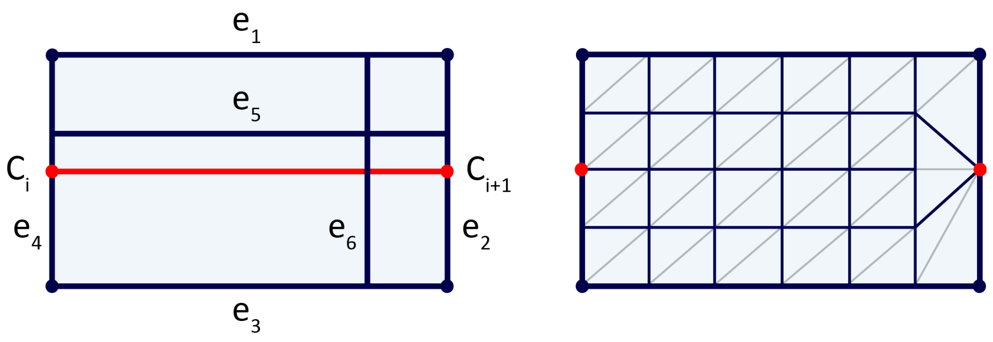


Figure 3.27: *Patch tessellation. As is illustrated in the left image, the patch tessellation is defined through 6 values which determine the tessellation of the different edges of the patch. The right image presents the result of tessellating a patch with the values [6, 2, 6, 4, 6, 4] as the tessellation factors of the edges [e_1, e_2, e_3, e_4, e_5, e_6]*

3.3.2.3 Tessellation control shader

The main goal of this stage is to determine the tessellation level used to subdivide the patch.

The shader first calculates a normalized distance of each vertex of the segment (C_i and C_{i+1}) towards the camera. These normalized distances are in the range [0,1], where zero is the minimum, and one is the maximum, and are calculated using the following equation:

$$Dist(C) = clamp\left(\frac{|C - pos_{cam}| - dist_{min}}{dist_{max} - dist_{min}}, 0.0, 1.0\right) \tag{3.15}$$

where $[dist_{min}, dist_{max}]$ is the range of distances where the subdivision of the patch varies. We have tested several distances to define this range, and we found that $dist_{min} = 10\,\text{Å}$ and $dist_{max} = 160\,\text{Å}$ produce good results without visual artifacts.

Hardware tessellated patches have the configuration illustrated in Figure 3.27 (left), and its subdivision is defined by the inner and outer tessellation factors. The outer tessellation factors define the subdivision levels of the outer edges e_1, e_2, e_3 and e_4, and the inner tessellation factors define the number of subdivisions of the internal edges e_5 and e_6. Figure 3.27 (right) shows an example of a possible patch subdivision with the values [6, 2, 6, 4, 6, 4] as the tessellation factors of the edges [e_1, e_2, e_3, e_4, e_5, e_6]. These patches are connected to generate the representation of the backbone. So, in order to avoid discontinuities or visual artifacts between two consecutive patches, the subdivision levels of the border edges e_2 and e_4 are defined using the distances of the points C_{i+1} and C_i respectively, whilst the rest of edges are defined using the maximum of the aforementioned. The formulas used to compute the factors for the edges are listed below:

$$\begin{aligned}
Tess(e_1) = Tess(e_3) = Tess(e_5) = \\
&max(Dist(C_i), Dist(C_{i+1})) * (BS_{max} - BS_{min}) \\
&+ BS_{min} \qquad (3.16) \\
Tess(e_2) = &Dist(C_{i+1}) * (SS_{max}(Type(C_{i+1})) - SS_{min}(Type(C_{i+1}))) \\
&+ SS_{min}(Type(C_{i+1})) \qquad (3.17) \\
Tess(e_4) = &Dist(C_i) * (SS_{max}(Type(C_i)) - SS_{min}(Type(C_i))) \\
&+ SS_{min}(Type(C_i)) \qquad (3.18) \\
Tess(e_6) = &max(Tess(e_2), Tess(e_4)) \qquad (3.19)
\end{aligned}$$

where BS_{max}, BS_{min} are the maximum and minimum subdivisions of the B-Spline, and SS_{max}, SS_{min} are the maximum and minimum subdivisions of the patch in the extruding direction (which is different for each type of secondary structure).

The tessellation factors of the edges e_1, e_3 and e_5 control the number of subdivisions of the B-Spline (horizontal subdivisions of the backbone in the current segment), and they are in the range $[BS_{max}, BS_{min}]$. In our application, we use twelve and two as BS_{max} and BS_{min}, which provides enough resolution to approximate the B-Spline. These tessellation factors are calculated first by determining which distance between the central control points (C_i and C_{i+1}) and the camera is the maximum when normalized and lastly by translating this distance from the range $[0, 1]$ to $[BS_{max}, BS_{min}]$. However, for the segments that generate the arrows of the β-sheets, the tessellation level of the B-Spline is always the maximum. If an adaptive tessellation level is allowed on these segments, visual artifacts will appear, as the arrow will be changing its shape with the geometric resolution.

The tessellation factors of the edges e_2, e_4 and e_6 instead, control the number of subdivisions of the patch in the extruding direction (vector d in Figure 3.23). Since the patches used to represent the different types of secondary structures have different sizes, the number of subdivisions needed to achieve a good visual quality is also different. Then, these factors use different ranges depending on the type of secondary structure, $[SS_{min}(Type), SS_{max}(Type)]$. In our system, we use eight and two as SS_{max} and SS_{min} if the segment belongs to an α-helix or a β-sheet, and if the segment does not belong to any secondary

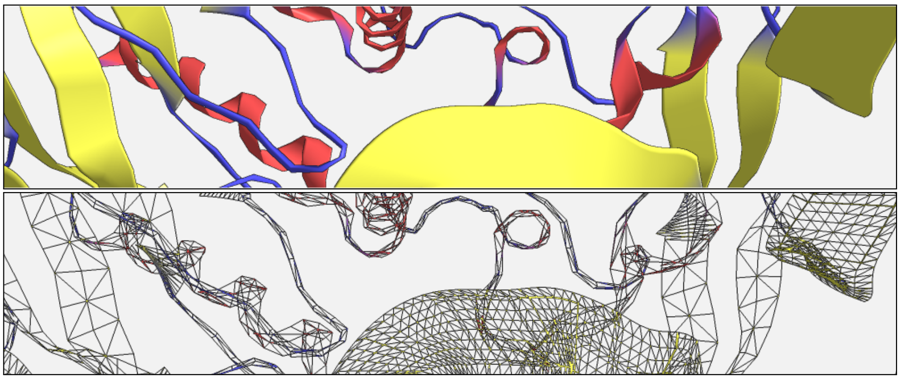


Figure 3.28: *Effect of the progressive tessellation. Note the different geometric resolution between near and far secondary structures (this effect has been exaggerated in this image in order to show the difference).*

structure both, values are set to two. As commented before, in order to avoid discontinuities and visual artifacts, the e_2 and e_4 tessellation factors are defined by selecting the normalized distance of the closest segment point (C_i for the edge e_4 and C_{i+1} for the edge e_2), generating thus the same number of triangles in the joints between two patches. These factors are then translated from the range $[0, 1]$ to $[SS_{min}(Type), SS_{max}(Type)]$. Lastly, the tessellation factor of the interior edge e_6 is defined by selecting the maximum between the e_2 and e_4 factors.

Besides the tessellation factors, the tessellation control shader also calculates the n vectors used to displace the newly generated vertices using the equation 3.14. The n vector is only computed for the central control points, C_i and C_{i+1}, since the information needed to generate the n vectors for the control points C_{i-1} and C_{i+2} is not available. In order to generate the n vectors of the four control points of the patch, the points C_{i-2} and C_{i+3} should be available in the tessellation control shader, allowing the computation of the segments directions s_{i-2} and s_{i+2}.

3.3.2.4 Tessellation evaluation shader

The tessellation evaluation shader determines the final position of the new vertices created during the subdivision of the patches. This shader is executed for each new vertex created by the tessellation pipeline. With every execution, a pair of coordinates (x and y) are available which indicate the position of the new vertex inside the patch. Using these coordinates, we move the new vertex to its corresponding position as described in Section 3.3.1. The x coordinate is

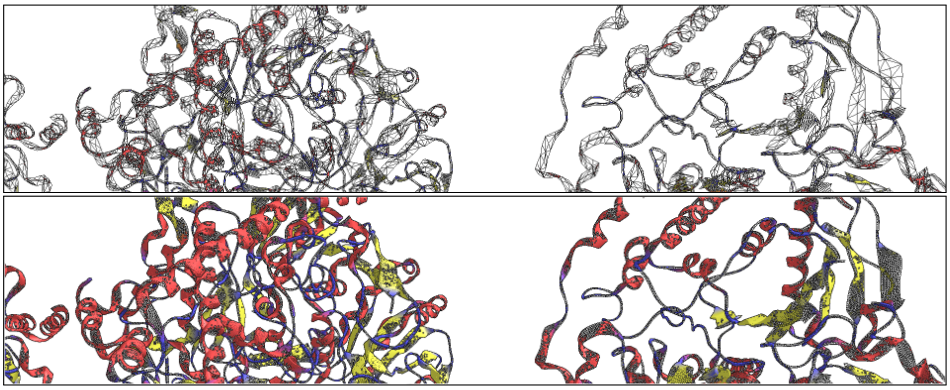


Figure 3.29: *Triangle reduction thanks to our adaptive tessellation (top) versus the regular tessellation that would guarantee correct images upon zooming (bottom). Note how the number of triangles is reduced in the top figure.*

used to evaluate the B-Spline and to obtain a 3D position along it. Moreover, we evaluate another B-Spline, also using the x coordinate, to obtain the interpolated d vector. The n vector, instead, is obtained using linear interpolation, since only the n vectors of two of the four control points are available. The 3D position on the B-Spline is then moved along the interpolated vectors d and n using the y coordinate as in Figure 3.23.

As a result, we have different patch resolutions for the close and far segments of the backbone, which has the main benefit of preserving the scalability of the algorithm. Figure 3.28 presents a molecule which secondary structures are rendered using different resolutions. To provide the close view, we have exaggerated the subdivision levels so that they appear at closer distances than in normal operation of the algorithm. Note how our system allows changing the number of triangles depending on the depth. In order to illustrate the effect of the adaptive tessellation in a real case, we also provide a comparison of a fully tessellated representation and an adaptive one in Figure 3.29. The image on the bottom contains only triangles, but really small since all the ribbons are created using the level that ensures they are correctly visible for near views. On the contrary, the top image shows how we save a lot of geometry with our adaptive method.

3.3.2.5 Fragment shader

The last stage of the pipeline, the fragment shader, is in charge of performing the lighting calculations to shade the surface of the sheet. To avoid artifacts

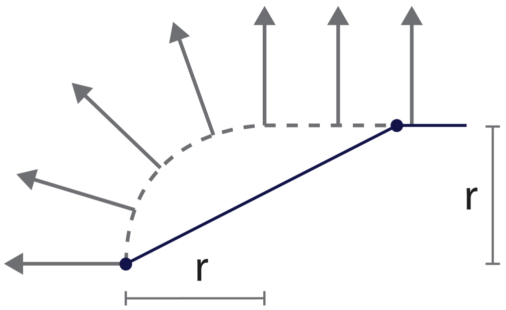


Figure 3.30: *The fragment shader recomputes the normal for each pixel in order to avoid artifacts due to the adaptive resolution of the patches. At the corners of the patch, defined by the displacement distance r, the normal is obtained interpolating between the vectors d and n.*

produced by the adaptive tessellation of the geometry, we calculate the normal also on this stage at pixel level. If the interpolated vertex normal is used instead, the lighting at the borders of the patches will change with the movement of the vertices carried out by the tessellation evaluation shader. The normal at fragment level is obtained linearly interpolating the vectors d and n on the corners of the patch, as illustrated in Figure 3.30. Here, the y coordinate provided by the tessellation stage is used to determine whether the pixel is in the corner of the patch, and the value used to interpolate the vectors.

3.3.3 Ambient occlusion

Besides generating the geometry of the secondary structures on the GPU, we also developed an algorithm to compute the ambient occlusion effect for these models. This algorithm works in real-time and it is based on the algorithm presented in Section 3.1, which was designed to approximate the ambient occlusion factors for molecules rendered using the Space-filling or the Balls & Sticks models. The previous algorithm has the following steps:

- First, we create a coarse representation of the scene that is stored as an occupancy pyramid. Each level of this pyramid is a voxelization of the scene at a different resolution that stores an approximation of the occupancy of each voxel. This occupancy pyramid is updated at every frame (using the compute shader) by performing an intersection test between the scene primitives (spheres and cylinders) and the voxels, and approximating the overlap between them.

- In a second pass, the ambient occlusion factor is calculated for each pixel

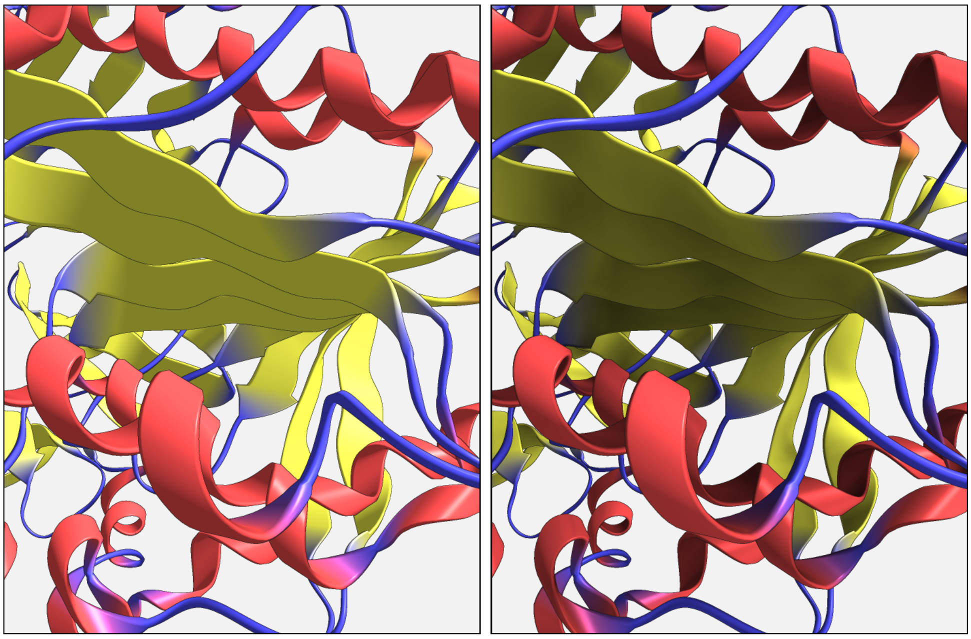


Figure 3.31: *The figure shows the difference between rendering a molecule using standard lighting methods (left) and ambient occlusion (right). On the right image, the shape perception is increased thanks to the darker areas in the interior faces of the α-helices or in the β-sheets occluded by other elements of the scene.*

using the voxel cone tracing algorithm [CNS+11] through the occupancy pyramid.

To adapt this algorithm to our ribbons model, we modified the generation of the occupancy pyramid. We computed the overlap between the patches of the backbone and the voxels of the data structure by approximating each backbone segment by a set of boxes (3 for each segment), each one with a different size. The voxels, on the other hand, are approximated by their bounding spheres. Then, as in the algorithm designed for cylinder-voxel intersections, the minimum distance between the box and the sphere is computed. The overlap between the elements is estimated with the following equation:

$$\frac{\frac{D}{2} - r}{D} \tag{3.20}$$

where r is the minimum distance between the objects and D is the diagonal of the voxel. Attempting to approximate each segment by a variable number

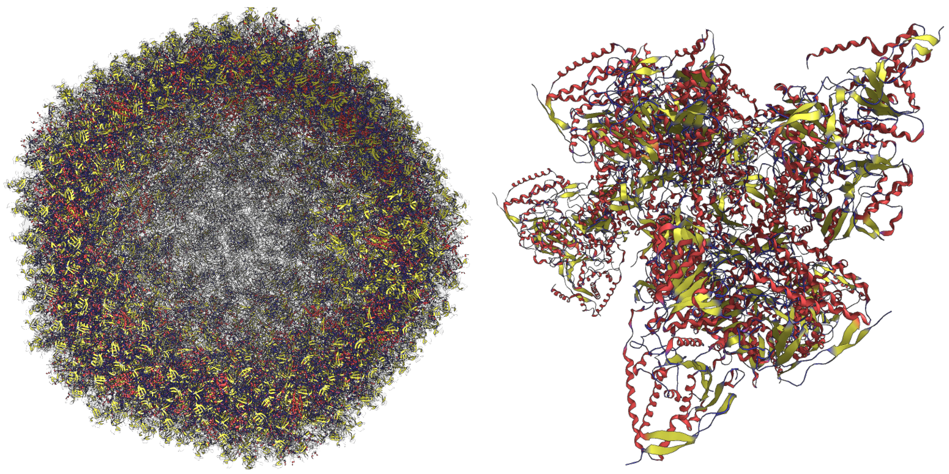


Figure 3.32: *This figure presents two images obtained using our algorithm with two molecules of different sizes, demonstrating the scalability of the proposed method. The molecule (3IYN) on the left, with 749K residues and 5,975 K atoms, is the biggest molecule we tested, while molecule on the right (3EXG) has a medium size of 10K residues and 83 K atoms.*

of boxes (as the geometry of the segments is adaptive too) would be counterproductive since it would lead to visible popping artifacts in the shadowed areas. An illustration of the cylinder-voxel overlap can be found in Figure 3.6 of Section 3.1, which can also illustrate the box-voxel overlap.

We present an example of a molecule rendered using our implementation of ambient occlusion for ribbons models in Figure 3.31.

3.3.4 Results

This section presents the results of the tests carried out to measure the performance of our algorithm. All these tests have been executed on an Intel Core i7 PC, running at 3.5 GHz, with 16 Gb of RAM, and a GeForce 770 GTX, rendering them in a 1280×720 viewport.

We ran a series of tests for molecules occupying a central part of the screen (we call it far or F) and with a zoom-in (called Near, or N)). In the close view, the maximum tessellation level is achieved by the near geometry to see the effect of the screen coverage. The molecules used in our test are diverse, ranging from a simple molecule of 249 residues to a large example of 749K residues (Figure 3.32 left). Note that the latter case is exceptionally large, orders of magnitude larger than the ones used in the pharmacological simulations we are addressing.

Table 3.4: *Performance measured in frames per second for different molecules (N)ear the camera (where the near geometry has the maximum tessellation level) and (F)ar from the camera (where all the geometry has the minimum tessellation levels). "Our" indicates the performance measured with our method and "[KBE08]" are the frames per second measured with the method proposed by Krone et al. [KBE08] (using five segment for each curve section and six edges for the tube's front and back faces). Framerates have been measured without uploading the buffers to the GPU (NU) and uploading them (U).*

Molecule	# residues	VBO	Our		[KBE08]	
name	(# atoms)	update	N	F	N	F
A	249	NU	2,594.21	3,596.47	1,531.91	1,946.81
	(3,967)	U	2,504.29	3,388.34	1,388.87	1,838.98
3EXG	10,781	NU	1,198.53	2,375.83	263.57	287.75
	(83,339)	U	1,059.86	2,093.71	254.99	274.18
1CWP	29,220	NU	760.75	1,330.46	86.84	90.06
	(227 K)	U	667.96	1,078.64	85.44	89.51
3IYJ	171,720	NU	246.47	290.35	14.48	14.46
	(1,356 K)	U	198.72	222.94	14.31	14.56
3IYN	749,340	NU	68.98	72.51	3.52	3.54
	(5,975 K)	U	48.41	50.19	3.36	3.39

We also evaluated the impact of uploading the data every frame to the GPU in order to obtain an insight of the performance in applications where the data is received every frame. Table 3.4 shows the results obtained by these tests. Note how we achieve real-time frame rates even with the largest molecule, and how our method scales very well with the size of the molecule. Moreover, the table shows how there is hardly any difference between the version that uploads the buffers to the GPU at every frame (shown as U) and the method that does not (NU in the table), as the data transmitted from CPU to GPU consists only of the backbone information.

Besides the performance tests, we also compared our method with the method presented by Krone et al. [KBE08] (Figure 3.35). The results of this comparison are also shown in Table 3.4. Their method obtains high frame rates for small molecules, but as the number of residues in the molecule increases the frame rate drops down. Our method, however, scales very well with the

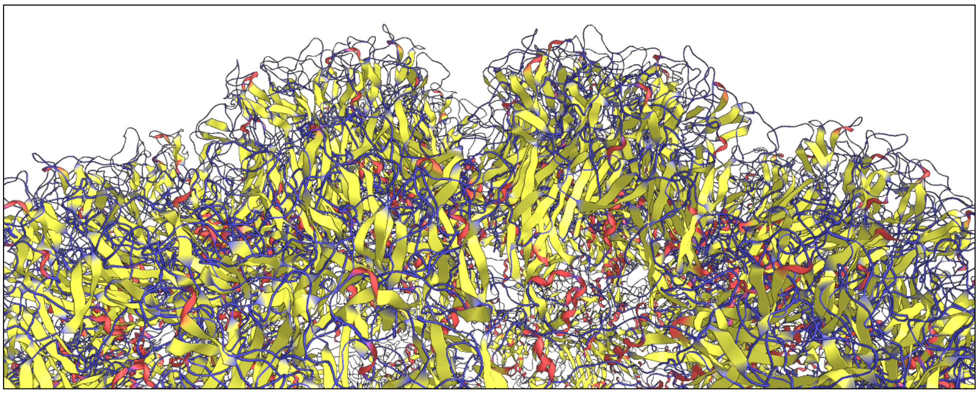


Figure 3.33: *Close up view of the virus capsid 3IYJ composed by 171K amino acids. Thanks to our adaptive resolution we are able to navigate through big scenes as the one presented by this figure.*

size of the molecule.

We also provide a comparison of the performance of our algorithm when adding ambient occlusion. Roughly, the frame rates decay to half (except for the first model, that was actually so fast to render that the AO calculation

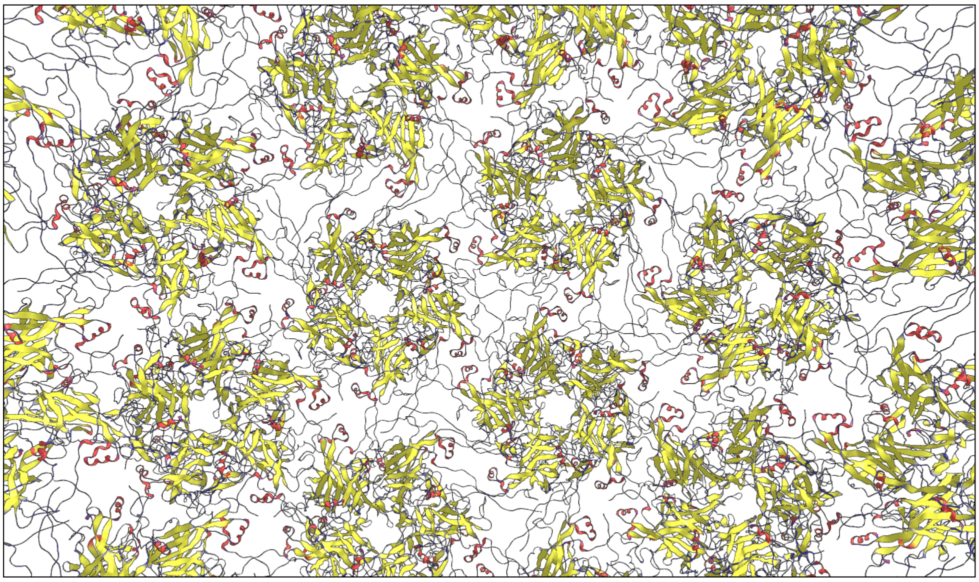


Figure 3.34: *Snapshot taken from inside the virus capsid 3IYJ composed by 171K amino acids and 1,356 K atoms.*

Table 3.5: *Performance measured in frames per second for different molecules without ambient occlusion (NoAO) and with it (AO). Note that even with ambient occlusion activated, our system is able to render complex molecules in real-time.*

Mol name	A	1CWP	3IYJ	3IYN
#residues	249	29,220	171,720	749,340
NoAO	2,582.18	726.41	244.17	70.01
AO	436.83	340.64	117.18	35.06

dominates the cost). In Table 3.5 we can see those frame rates. Note that we still maintain real-time frame rates for the most complex molecules.

3.3.5 Conclusions

We have presented an algorithm capable of generating and rendering the geometry of the secondary structures of very complex proteins on-the-fly using the tessellation stage of the GPU. While other existing algorithms also use the GPU to generate the geometry on-the-fly, ours is capable of generating only the geometry needed for the current point of view, allowing the interaction with bigger molecules in real-time and the use of other rendering effects in order to increase the visual quality of the rendering. We have also adapted an existing method to calculate the ambient occlusion factors for this type of scenes, obtaining real-time frame rates even for macromolecules.

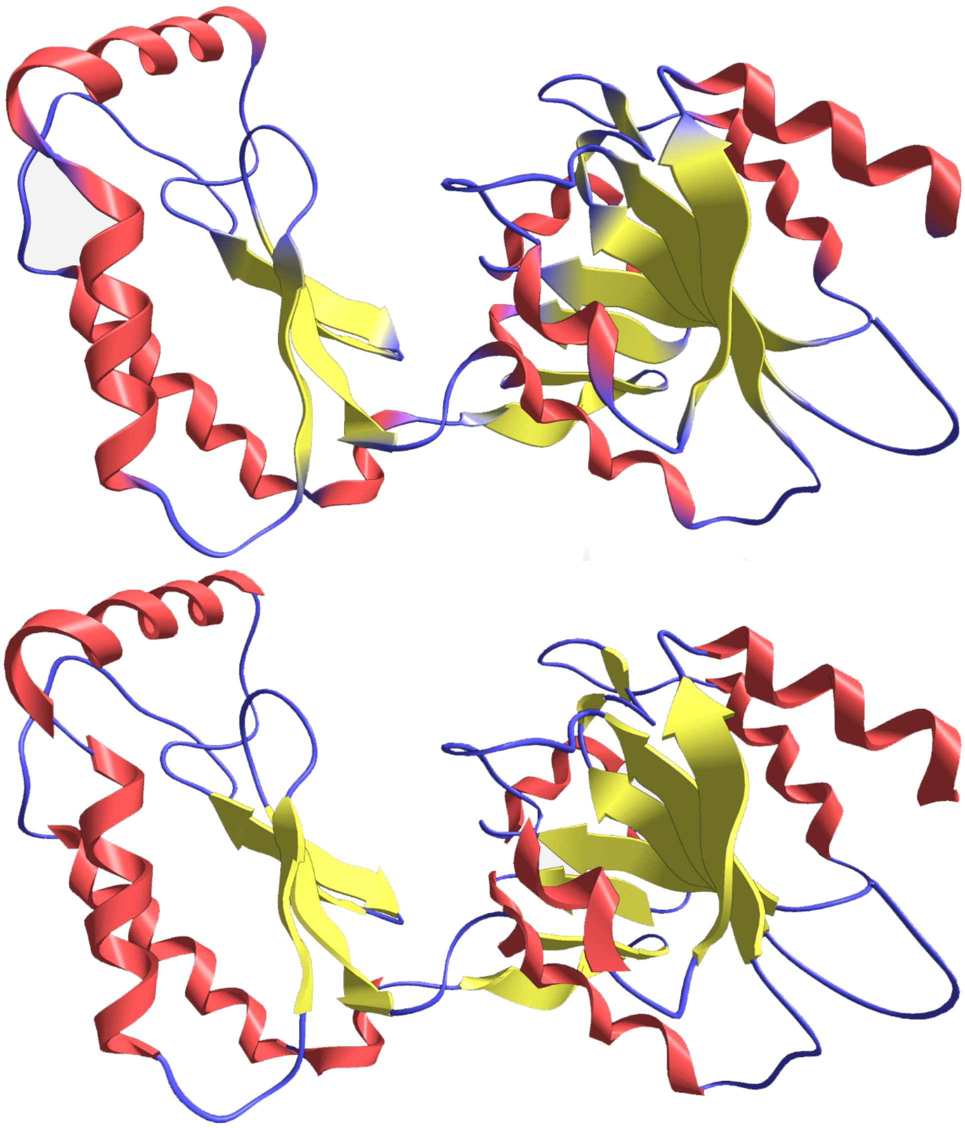


Figure 3.35: *Visual comparison of our method with the method proposed by Krone et al. [KBE08]. The two images are similar since the only differences are the edges of the patches and the transitions between different secondary structures. Krone et al.'s method has sharp edges and transition, while in our method they are smoother. However, the main difference between the methods is that Krone et al.'s method uses always the same geometric resolution while our method is adaptive, allowing the real-time visualization of bigger models.*

3.4 Interactive solvent excluded surface refinement

Molecular surfaces are commonly used methods to analyze molecular structures, as they can illustrate the accessibility to the molecule. In particular, the *solvent excluded surface* (SES) illustrates the reachable areas of the molecule for a specific solvent (represented by a sphere). The SES is generated by rolling a spherical probe (the solvent) over the van der Waals (vdW) surface of the molecule (see Figure 3.36). Thus, the SES is defined as the surface traced out by the probe during this process.

Since the SES is very well suited for analyzing the molecular interface and cavities, its use is widespread for facilitating the analysis of molecular dynamics (MD) simulations. In pharmacology drug design, these simulations calculate the interaction between a molecule and a ligand, and they output a set of consecutive spatial atom configurations (trajectories). Due to the complexity of the SES computation process, most of the existing algorithms compute it in a preprocessing stage, rather than the mapping stage of the visualization pipeline. While this allows for SES generation for static molecules and fixed probe radii, preprocessing is not feasible for dynamic setups, where the atom positions change over time, or the probe radius is altered during a visual analysis.

In the past few years, the development of new algorithms [LBPcH10, KGE11, PV12] allowed the computation of an analytical solution in real-time (or interactive frame rates) for molecules of small-medium size. However, the memory consumption and computational cost of these algorithms grow exponentially with the size of the molecule. More details about of the state-of-the-art methods to compute molecular surfaces are given in Section 2.1.4.

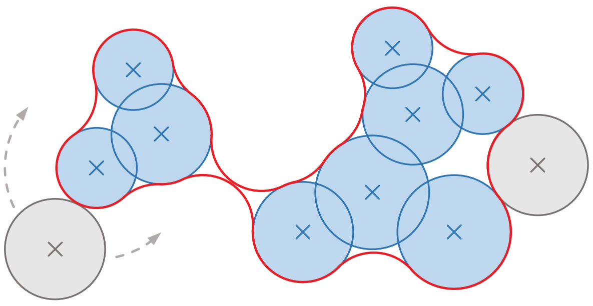


Figure 3.36: *The SES (red) can be defined analytically as the surface that is traced out by a spherical probe (gray, shown in two sample positions) rolling over the vdW surface of the atoms (blue) of a molecule.*

In this section, we propose a novel SES computation algorithm that exploits modern GPUs to support the interactive generation of SESs. The algorithm has been developed specifically for dynamic data. We further do not assume or rely on any particular frame ordering, since modern systems usually generate many of those configurations at once in parallel. To achieve interactive SES updates, we have used a grid-based approach, which is suitable for GPU implementation. And, to make our algorithm scalable, it has been designed in such way that for large molecules, for which real-time frame rates cannot be achieved at full detail, a progressive update of the SES is supported (see Figure 3.37). Moreover, we introduce an interpolation scheme which allows a seamless transition between the different levels of details.

The rest of the section is organized as follows: Section 3.4.1 describes the basic algorithm used to compute the SES, its implementation on GPU, the progressive algorithm in charge of the SES refinement, the coloring method used, and the rendering algorithm which provides a smooth transition between intermediate results. Moreover, this section discusses the error generated by our algorithm and a method to reduce it. In Section 3.4.2 the results of the tests are shown together with the results of the comparisons against different methods. Lastly, the conclusions are presented in Section 3.4.3.

3.4.1 Algorithm

Kozlikova et al. [KKL+16] classified, in their state-of-the-art report, the algorithms to compute the SES in two main categories: those that compute an analytical representation of the surface and those that compute the surface by discretizing the space around the molecule. Our algorithm falls into the second category, as we use a regular 3D grid that represents a signed distance field to the SES. Despite the resulting memory consumption requirements, this representation allows us to compute a coarse representation of the SES in real-time easily, to refine this coarse representation progressively, and to create a smooth transition between different levels of detail.

The workflow of our algorithm is the following: When the application loads a new molecule (or receives a new frame from the simulation software), we compute a signed distance field to the SES using a low-resolution grid. This computation is carried out in milliseconds due to the low resolution of the grid, so the user does not perceive any drop in performance. This coarse representation is then immediately rendered and shown to the user. Meanwhile, the algorithm computes refined versions in the background by increasing the resolution of the grid. Once a new level is computed, the algorithm performs a smooth transition between the current level and the one just computed.

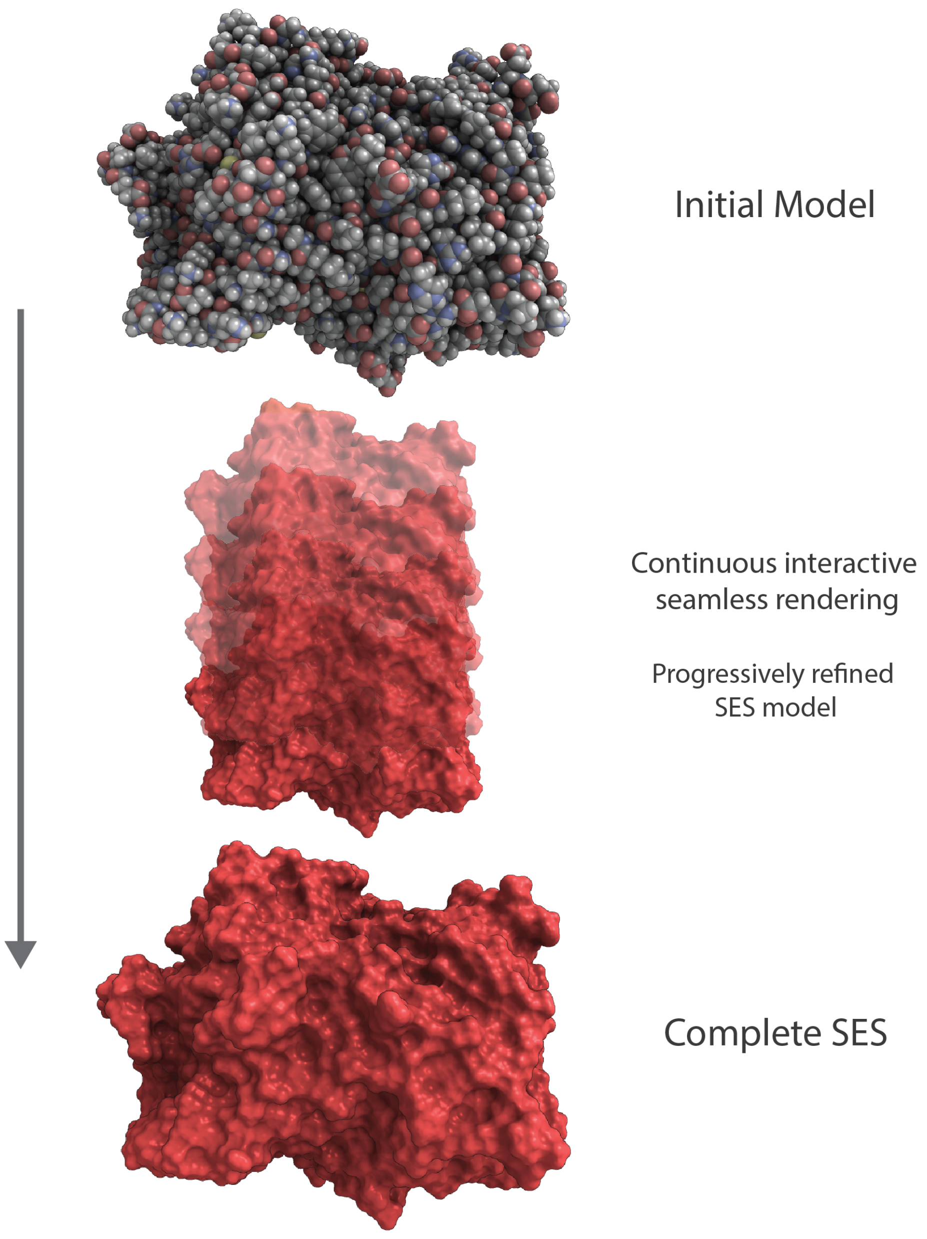

Initial Model
Continuous interactive seamless rendering
Progressively refined SES model
Complete SES

Figure 3.37: *Illustration of the SES generation algorithm. To maintain interactivity, if GPU capabilities are not enough to generate the full SES in a single frame, a coarse model is generated first, which is then progressively refined several steps. Progressive refinement occurs in a seamless way, and the user can explore the molecule meanwhile.*

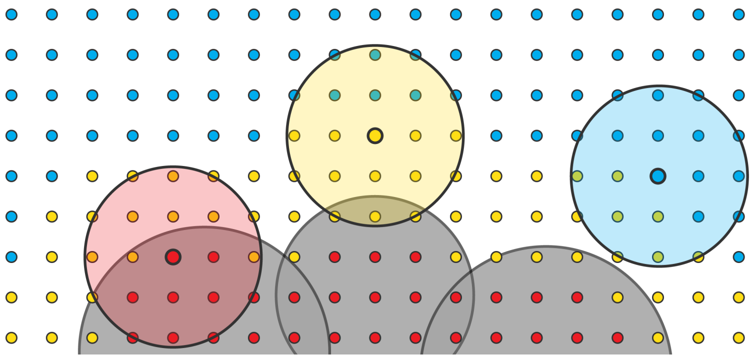


Figure 3.38: *Step 1: Probe intersection. The figure shows how the tests performed at the grid points are used to classify them. Red points are classified as* interior *to the SES, blue ones are classified as* exterior *to the SES, and yellow ones as points on the* boundary *of the SES.*

3.4.1.1 SES computation

Our algorithm to compute the SES is based on the one proposed by Lindow et al. [LBH14]. They developed a grid-based algorithm to compute the *ligand excluded surface* (LES), a generalization of the SES where the surface is defined by rolling a molecule instead of a sphere (more details are given in Section 2.1.4). Our algorithm is also grid-based and it generates a 3D signed distance field with positive values outside the SES and negative values inside. Similar to Lindow et al. [LBH14], we compute the signed distances only in the proximity of the SES, so the range of distances is limited. In our case, this range is $[-r_g, r_p]$, where r_g is the distance between two neighbor grid points and r_p is the probe radius. Points of the grid at longer or shorter distance are clamped to r_p and $-r_g$ respectively.

Our algorithm, unlike the one proposed by Lindow et al. [LBH14] for the LES, has no collision problems and it can be executed in parallel without using synchronization mechanisms, making it suitable for GPU implementations. We accomplished that by dividing the computation into two distinct steps: *probe intersection* and *distance field refinement*.

3.4.1.1.1 Probe intersection

In the first step of the algorithm, the points of the grid are classified as points located outside the SES, inside the SES, or in the boundary of the SES. This classification is based on two simple tests: checking both the center of the grid point and a probe located at the center of the grid point for intersections with

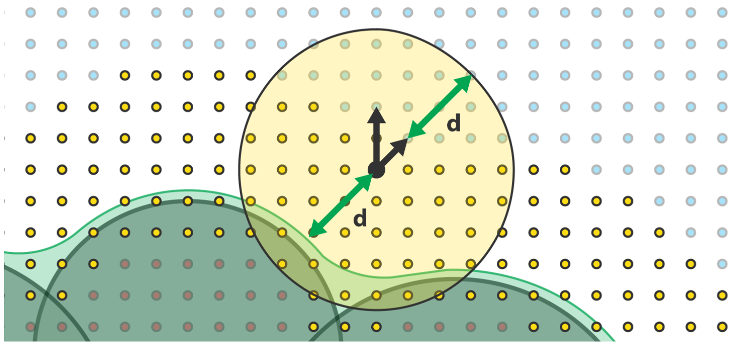


Figure 3.39: *Step 2: Distance field refinement. For each yellow point, the algorithm searches for the closest blue point in a neighborhood. The distance to the SES is then computed as the difference between the probe radius and the distance between these two points.*

the atoms of the molecule (using the Van der Waals radius). An illustration of these two tests is shown in Figure 3.38. The results of these tests are used to classify the points according to the following criteria:

Outside SES: If there is no intersection between the probe and the atoms, the grid point is classified as a point outside the SES, and the algorithm assigns r_p as the distance to the SES (blue points in Figure 3.38).

Inside SES: If the distance between the grid point and at least one atom of the molecule is less than the radius of this atom minus r_g (the distance between two closest neighbors on the grid), the point is classified as an interior point of the SES (red points in Figure 3.38). In this case, the algorithm assigns $-r_g$ as its distance to the SES.

Boundary: If there is an intersection between the probe and an atom, but the distance between the grid point and all the atoms of the molecule is longer than the radius of the atoms minus r_g, the grid point is classified as being on the boundary of the SES (yellow points in Figure 3.38). The distance between such a point and the SES is determined in the second step of the algorithm.

3.4.1.1.2 Distance field refinement

In this second step, the remaining distances to the SES of the points in the border region are computed. Thus, the algorithm searches the neighborhood of these boundary points for adjacent points that are outside the SES. As described in the previous section, on the points outside the SES a probe could be placed without intersecting any of the surrounding atoms, what could make the surface of these probes a good approximation of the SES. The distance of a boundary point p_c is then updated with the maximum distance from p_c to the surface of the probes (being negative if p_c lies outside the probe).

To ensure that all the probes which could affect the distance of our p_c point are tested, its neighborhood is defined by the set of points p_i at a distance r_p+r_g or less from p_c. The algorithm initializes the distance of p_c to $-r_g$, and then it iterates over all the points in the neighborhood and selects the closest p_i that lies outside the SES. If no p_i was found that is outside the SES, the distance of p_c is not updated (e.g., it remains $-r_g$). If there is at least one, the distance of p_c is updated using the following equation:

$$d = r_p - \|pos_i - pos_c\| \tag{3.21}$$

Figure 3.39 illustrates this process.

Note that updating the distances in parallel requires no synchronization. The algorithm only searches the neighborhood of each sample for points outside the SES and does not modify these values in this step; they were computed in the first one.

3.4.1.2 Implementation

We have implemented our algorithm to run on the GPU, using compute shaders for both steps. To represent the signed distance field we used a 3D float texture centered at the molecule which encloses it. Each thread of the first compute shader classifies a grid point of the distance field as described in the first step of the algorithm. Then, each thread of the second compute shader calculates the distance to the SES of one *boundary* grid point. In order to perform these computations efficiently, some optimizations are applied, which we now describe.

In order to classify a point, the algorithm has to perform intersection tests between the grid point and the atoms of the molecule, which can be prohibitive when the number of atoms increases. Therefore, it is crucial only to perform the intersection test with close-by atoms and discard the ones that are farther away. Efficient retrieval of neighboring atoms within a fixed radius is a common problem that is usually solved by using spatial subdivision data structures (see, e.g., [Gre07, BSC15, Hoe14]). We opted for the method of Green [Gre07], which was also used by Krone et al. [KGE11] and Skånberg et al. [SVGR16] in different molecular visualization techniques. This method subdivides the space into a regular grid and sorts the atoms into the grid cells based on their centers. Then, we can obtain neighboring atoms within a fixed radius in constant time. In our case, we set the cell size to the probe radius r_p plus the maximum Van der Waals radius. This cell size guarantees that we only have to visit 27 cells of the grid to obtain all the atoms that possibly intersect with our probe.

As described by Green [Gre07], the distribution of the atoms into the cells can be efficiently implemented using the radix sort algorithm[1]. This algorithm assigns an integer key to each cell based on their position, and, for each atom, the key of the cell they belong is computed. The atoms, then, are sorted based on their keys using the radix sort algorithm, packing together all the atoms belonging to the same cell. The last step determines the starting and ending atom index for each cell. We have implemented the radix sort algorithm using a compute shader for each of the steps of the algorithm.

The compute shader which implements the second step of the algorithm executes a thread for each point classified as *boundary* since these points have no distance assigned yet. However, keeping track of all these points is not a straightforward task in the context of a parallel algorithm. One possible solution is to mark these points in the first step and, on the CPU, pack them into a buffer, executing the second step of the algorithm only for them. Although this is a simple solution, it does not scale well when the resolution of the grid increases (the number of grid points to keep track grows exponentially) and we lose spatial coherence in the execution of a shader workgroup (which can lead to performance issues related to cache misses when accessing the data). Instead, we chose a more GPU-friendly solution. We grouped the grid points into bricks of 8^3, and then, we selected those that need further refinement. Among all bricks of the grid, the ones that need further refinement are defined as the ones that contain at least one boundary point (see the yellow bricks in Figure 3.40). Packing grid point into bricks reduces the workload of the CPU and accelerates the selection process. In addition, it lowers the data transfer between GPU and CPU, and it adds local coherence inside the workgroups in the second step as we use a workgroup size equal to the brick size (8^3).

3.4.1.3 SES progressive refinement

One of our main goals in this project was to ensure real-time interaction. For this reason, in our system, the application computes the SES using a low-resolution 3D grid in real-time and, in the background, it refines the coarse

[1]The radix sort algorithm is an algorithm able to sort a list of integers in constant time ($\mathcal{O}(n)$). More details about this algorithm and its implementation on GPU can be found on [HH11].

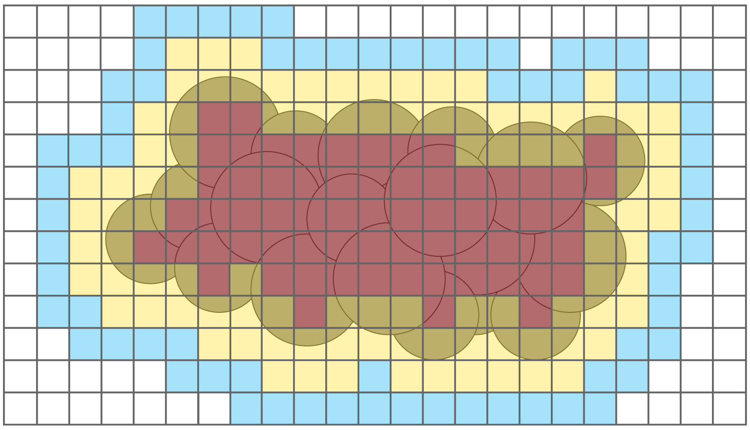


Figure 3.40: *The grid points are packed together into bricks. In the first step of the algorithm, these bricks are classified. Yellow bricks have at least one grid point labeled as SES border. The blue bricks are the ones close to the yellow ones. Red ones have all grid points classified as interior points. The rest of the bricks (white) are not taken into account in the computation of the refined SES.*

surface to provide a more exact SES progressively.

In order to store the different SES resolutions, we selected a data structure composed of a mipmapped 3D array of floats with a base resolution of 512^3. This data structure is initialized with the value of the probe radius since it is the maximum distance allowed by our algorithm. We have chosen this maximum resolution as a compromise between SES quality and memory consumption. For a virus capsid of 500k atoms (1K4R), the grid cells are still below 1 Å, while it only uses 620 Mb.

For a given molecule, the algorithm calculates the SES for a range of levels within the mipmapped array. The base level, l_s, is used to compute the coarse representation of the molecule, while the end level, l_e, is the level used to compute the most refined version of the SES. The algorithm starts computing the SES for the l_s, and, progressively, computes the SES for the rest of levels until the SES at l_e is complete.

The main bottleneck of the SES computation algorithm is in the *distance field refinement* step (described in Section 3.4.1.1.2). When the number of neighboring points that are considered increases, the performance decreases. Thus, at l_s, we choose the smallest neighborhood that keeps an acceptable SES quality and allows its computation in real-time. We choose the highest level in the hierarchy in which the distance between two neighbor grid points is less

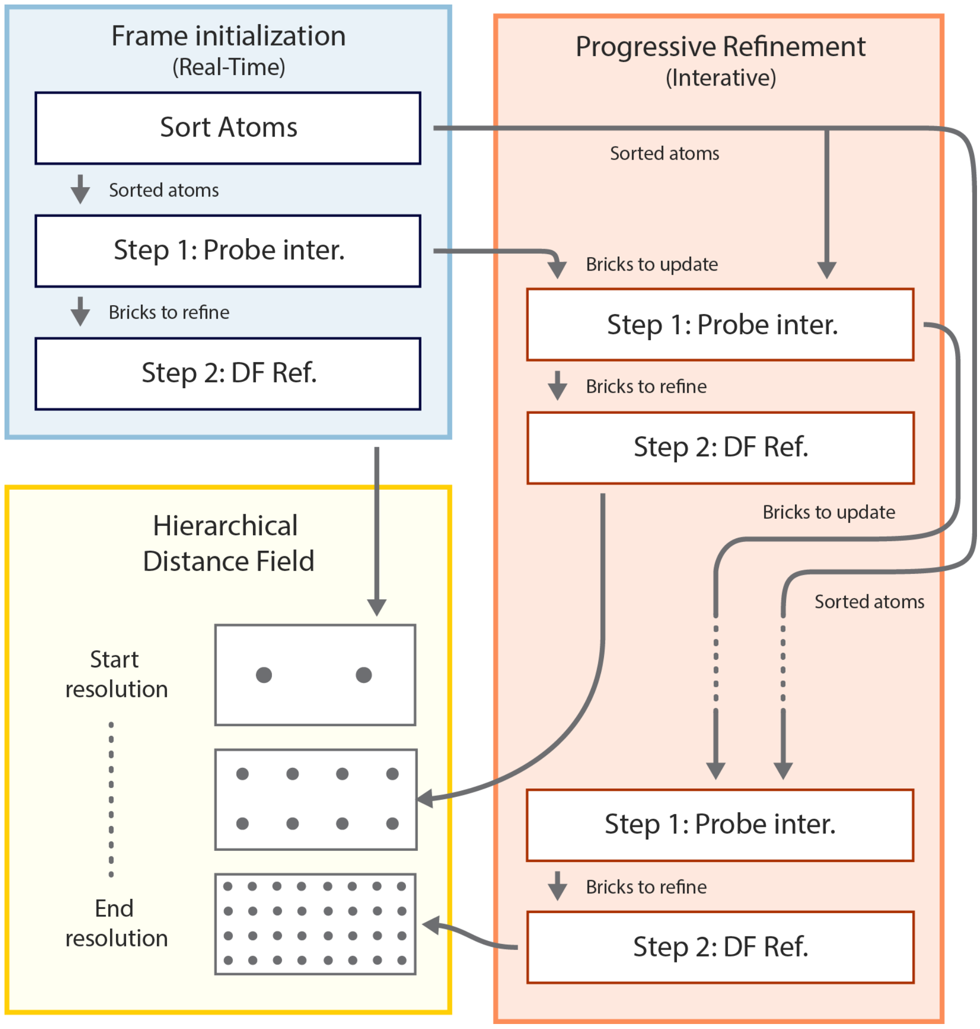


Figure 3.41: *Overview of the progressive refinement process: First, the atoms of the molecule are sorted and then, a coarse version of the SES is computed in real-time. The sorted atoms and the brick list obtained in the first computation are then used to compute a more refined version of the SES. Along the frames, new versions of the SES are computed using the brick list of the previous resolutions until the highest resolution is reached.*

than the probe radius. With this configuration, the algorithm considers for each grid point the cells at a (Manhattan) distance of less than or equal to two, which makes a maximum of 124 neighbors. For large molecules, we do not allow a resolution greater than 128^3 for the level l_s since a bigger resolution could not be computed in real-time. To select l_e, the algorithm follows a similar criterion, choosing the lowest level in the hierarchy where the distance between two neighbor grid points is less than the probe radius divided by 7 (for each grid point the algorithm has to visit seven neighbors in each direction). However, since our data structure has a finite resolution of 512^3, the final level may not exceed this limit. These numbers have been chosen empirically from tests performed on our hardware, but they can be modified to tune the algorithm to different hardware.

Once l_s and l_e are known, the algorithm computes the SES for l_s and presents the result to the user immediately. First, the algorithm sorts the atoms into a spatial subdivision grid G_a (using the GPU radix sort algorithm described in section 3.4.1.2), which will be used for the computation of all levels in the range $[l_s, l_e]$, so it has to be computed only once. Next, the two steps of the algorithm are executed to calculate the coarse version of the SES. In the first step, the classification of the grid points is performed. In addition, this step classifies the bricks of the grid (see Section 3.4.1.2) in four different categories:

Border bricks (B_b) These bricks are defined as the ones containing at least one point on the border of the SES (yellow bricks in Figure 3.40).

Interior bricks (B_i) Interior bricks are only composed of grid points inside the SES (red bricks in Figure 3.40).

Adjacent bricks (B_a) These bricks contain grid points completely outside the SES, and they have, at least, one neighboring brick in the boundary region (B_b) (blue bricks in Figure 3.40).

Outside bricks (B_o) Outside bricks are the rest of the bricks of the grid, the ones which contain grid points completely outside the SES and have all the neighboring bricks classified as B_o too (white bricks in Figure 3.40).

In the second step, the B_b bricks are updated with the distance to the SES. At the coarsest level, this whole computation takes milliseconds and can be performed in real-time during rendering. The output of this stage is composed of: a distance field that represents the coarse SES, the atoms distributed into G_a, and the classification of the bricks. The algorithm immediately renders the coarse SES, so that the user can interact with it. Meanwhile, in the background, the algorithm computes the following levels of the hierarchy using the information obtained from the coarse level calculation.

In order to reduce the computation of the next level, only the volume of the SES and its vicinity are recomputed. Therefore, we execute our SES algorithm

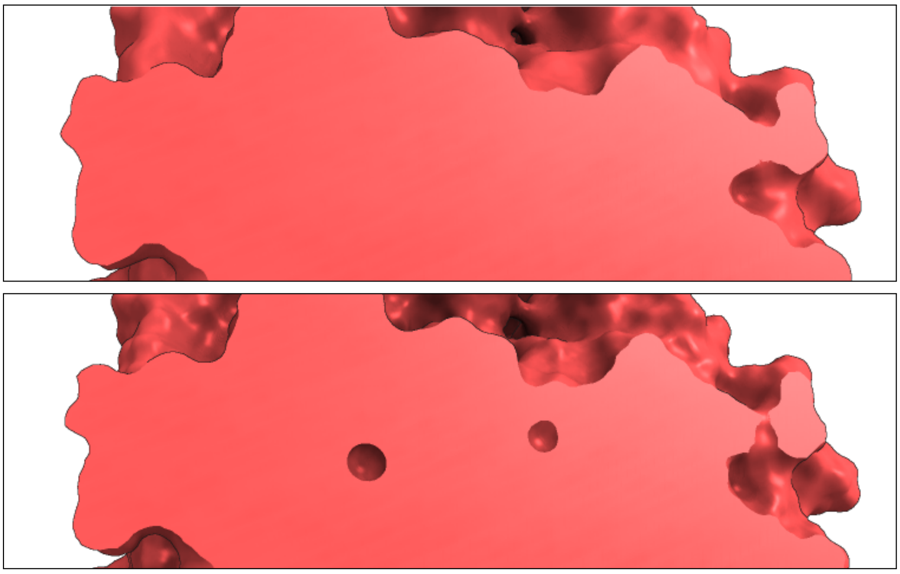


Figure 3.42: *Discrete sampling can miss features inside the surface (e.g., cavities or tunnels). This image shows how a cavity is not detected in a surface generated with a low grid resolution (top, generated with a* 128^3 *grid resolution), while it is in a high resolution (bottom, generated with a* 256^3 *grid resolution). Updating interior bricks in all levels is mandatory to overcome this limitation of the discrete sampling.*

on the volume defined by the B_b, B_i and B_a bricks of the coarse level, l_s. Note that a brick on the level l_i covers the same volume as eight bricks in the level l_{i-1} since the resolution is doubled in each axis. We execute, then, our SES algorithm in those bricks contained by the B_b, B_i and B_a bricks of the coarse level. The recomputation of the bricks classified as B_d is mandatory as they contain the actual surface, but the recomputation of B_i and B_a may not seem so obvious. B_i and B_a bricks are also recomputed to avoid artifacts in the refined versions. B_i bricks contain points completely inside the SES, and they are recomputed to do not miss internal features of the SES (cavities or tunnels). Small interior features of the SES are missed if the surface is poorly sampled, which can happen in the first coarse levels of the hierarchy. Recomputing these areas with a higher sampling in the lower levels can recover these missing features (see Figure 3.42).

B_a bricks contain grid points completely outside the SES that have at least one neighboring brick in the boundary region (B_b). These bricks have to be recomputed in the remaining levels to avoid incoherent distance values between

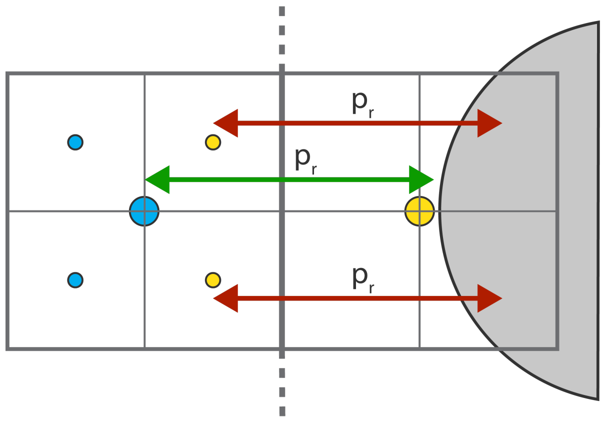


Figure 3.43: *The big blue point on the left belongs to a brick that does not need to be refined but has a neighbor that belongs to another brick that needs to be refined: the big yellow point to the right. In a higher resolution, some of the points of the left brick became yellow (a probe placed at them have an intersection with the molecule) even if all the points in the previous resolution had no intersection. These bricks have to be updated in the higher resolution levels to avoid artifacts on the resulting surface.*

adjacent grid points. A brick can be composed only of grid points outside the SES at a certain resolution, but, when more samples are used, that could not be the case anymore. The new samples can be part of the SES boundary as illustrated in Figure 3.43. Hence they need to be updated.

The SES computation of the next level is performed using the following pipeline: First, the classification of the bricks carried out at the coarse level is downloaded to CPU memory. Then, on CPU, the bricks that need to be recomputed are packed together in a buffer. In the background, while the user inspects a coarser version of the SES, the application executes the algorithm described in Section 3.4.1.1 for the selected bricks. The algorithm works exactly in the same way as it works for the coarse level. The result is presented to the user using a smooth transition between the previous level and the new refined one. This process is repeated for further levels in the hierarchy until the algorithm computes the last one, l_e (always using at level l_i the brick classification calculated in the previous level l_{i+1}). This algorithm is executed in parallel to the rendering, so for each frame, a fixed number of bricks are processed. To maintain an interactive frame rate, we process only 512 bricks per frame, but this number could be reduced on slower GPUs (we have used a NVidia GTX 970 in our experiments).

3.4.1.4 Detection of missing features

Due to the discrete nature of our representation, the volume of the SES can be overestimated. Each point of the grid stores only an approximation of the distance between the point and the surface. The real distance, however, is in the range $[d, d + (\sqrt{3} \cdot r_g)]$, where d is the approximated distance stored in a grid cell and r_g the distance between two neighboring cells along one axis. That is, r_g defines the maximum error of our solution.

Consequently, the probability of missing internal features of the molecule (cavities or tunnels) increases with r_g. Simply speaking, the chances of missing a small cavity are higher if the space around the molecule is sampled by a small number of points (i.e., a coarser grid).

These limitations may not be relevant for small r_g, but, when r_g is larger than a certain value, the result can deviate significantly from the analytical solution. Thus, we developed a method that refines the last level of our SES, l_e, in certain areas. The algorithm analyzes the bricks of the highest resolution grid to find areas with possible missing features. These bricks have been defined as having at least one point that satisfies the following conditions. First, this point has to be located inside the SES but outside of the atoms of the molecule. Second, it has to be surrounded only by interior points. And third, one of its neighbors also has to be outside of the atoms and, together, they have to satisfy the following condition: $d_{p1} + d_{p2} + |pos_1 - pos_2| \geq 2 \cdot r_p \cdot \mu$, where d_{p1} and d_{p2} are the distances between the points and their closest atoms, $pos_1 \in \mathbb{R}^3$ and $pos_2 \in \mathbb{R}^3$ are the positions of the grid points, r_p is the probe radius, and μ is a user-defined parameter that controls the number of selected bricks. If two points satisfy these conditions, a probe might fit between them. Note how two neighboring points may not satisfy the conditions for $\mu = 1$ and still miss a cavity since the probe can be placed at some distance from the line defined between them. Moreover, two neighboring points can satisfy the conditions for $\mu = 1$ and do not miss any internal feature (which often happens for large r_g). Parameter μ helps the user to limit the extra computations according to her needs and the size of the molecule.

For the volume contained by the bricks that passed this test, our algorithm is executed with a resolution of 64^3 with a small modification. If a probe can be placed in one of these new points, our mipmap is updated with the newly calculated distances at the highest resolution. These updates have to be performed using data access synchronization (atomic operations) since more than one new sample could try to modify at the same time the value of an original grid point.

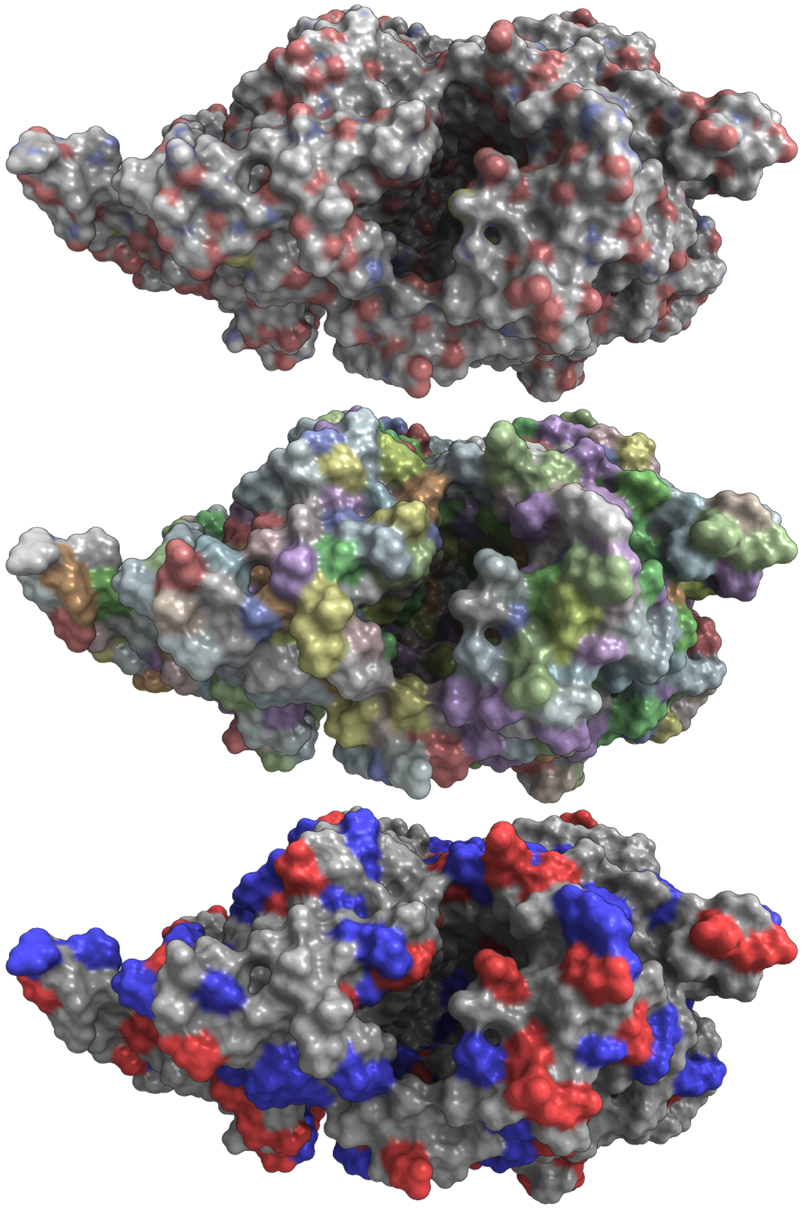


Figure 3.44: *Top: the SES is colored using the CPK color convention to identify the atoms; center: color mapping showing the residue types; Bottom: (per residue) electrostatic potential.*

3.4.1.5 SES coloring

Researchers often use the SES to detect tunnels or cavities in the surface, but it is also important for them to be able to identify the atoms/residues, the electrostatic potential, the hydrophobicity, and other properties in the surface. To support the visual analysis, we store another 3D texture at the lowest resolution selected for this molecule, l_s, and, for each texel, we store the color or property associated with the nearest atom. By using this texture, the surface can be smoothly colored according to the stored properties.

We chose to encode these properties at the lower resolution to have smooth transitions between colors since higher resolutions would make the borders between colors clearly visible. Moreover, it does not add extra computational costs (it can be computed together with the coarsest level) and it uses limited extra memory, as the selected resolution is never greater than 128^3. Different examples are presented in Figure 3.44.

3.4.1.6 SES rendering

In this section, we describe the rendering algorithm used to visualize the SES stored as a distance field. The most commonly used techniques to visualize a surface stored in a distance field are either by extracting a mesh using marching cubes (MC) [LC87] or by ray-marching it directly [HSS$^+$05]. We chose to ray-march the distance field, as it does not require extra storage and it can be done efficiently. The classical algorithm of ray-marching takes samples along the ray, separated by constant distance, but we use, instead, the values stored in the grid points as the distances between consecutive samples. This optimization speeds up the rendering and removes some possible artifacts (e.g. missing the surface on the border). When the ray hits the surface, we use a Sobel filter [GW06] to compute the normal at that point. This filter compares the values of the 26 neighboring points against the values stored at the hit point. On hardware with low performance, a simpler filter could be used instead, reducing thus the number of texture lookups (e.g. central differences [BLM96]).

Another advantage of ray-marching over MC is that it allows us to perform smooth transitions between the different levels of detail easily. When a new level l_i of the distance field hierarchy is computed, the renderer has to perform a transition from the current level l_{i+1} to the new level l_i along a time frame t. This transition is done by the hardware trilinear interpolation. The render performs a linear interpolation of the level identifier along time, from $i+1$ to i. This interpolated value is then used as the LOD parameter in the *textureLod* call during the ray-marching. With this function call, the hardware performs a linear interpolation between the distances stored at the different levels, which is translated into a smooth surface transition between levels. When the hit point is reached and the normal has to be calculated, the algorithm also linearly interpolates the distance between neighbor samples. This interpolated

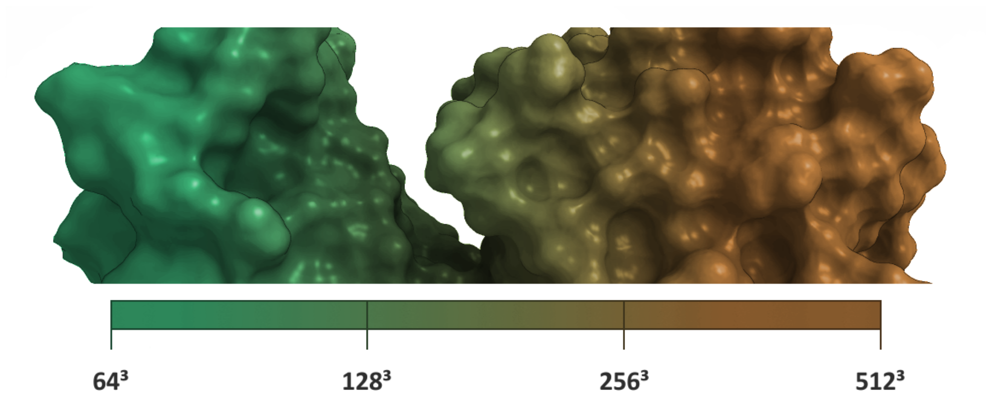


Figure 3.45: *The image shows from left to right the progressive refinement in the SES generation process. Note the continuity through the different steps thanks to our algorithm.*

distance is used to determine the neighbor positions in order to compute the normal using the Sobel filter. This simple algorithm gives us a smooth transition between grid resolutions for both surface shape and lighting. Figure 3.45 illustrates how a molecule was rendered using different grid resolutions from left to right, applying smooth transitions between them.

Moreover, we improved the performance of our ray-marching algorithm using a well-known technique from GPU-based volume raycasting. In order to reduce the ray traversal distance, we render the B_b bricks of the desired level to encode a texture with the entry and exit points of the rays, skipping thus the empty space of the volume. This simple technique substantially increases the frame rate of our rendering algorithm, and, since we already keep track of these bricks in our SES computation, it does not add extra computational cost.

Furthermore, we used ambient occlusion to increase the understanding of the surface's shape and help the visual identification of cavities and tunnels. Since the SES of a molecule is derived from its Space-filling representation and they are very similar, we used the ambient occlusion algorithm described in Section 3.1 to calculate the occlusion factors of the SES. This algorithm computes a 3d occupancy pyramid, which approximates the volume occupied by the Van der Waals spheres of the atoms, and then uses it to compute the ambient occlusion factor of each pixel. To compute ambient occlusion factors on the SESs, we used the occupancy pyramid calculated with the Van der Waals spheres. Then, for each pixel, we approximate its occlusion factor applying the algorithm *Voxel Cone Tracing* over the previously calculated occupancy

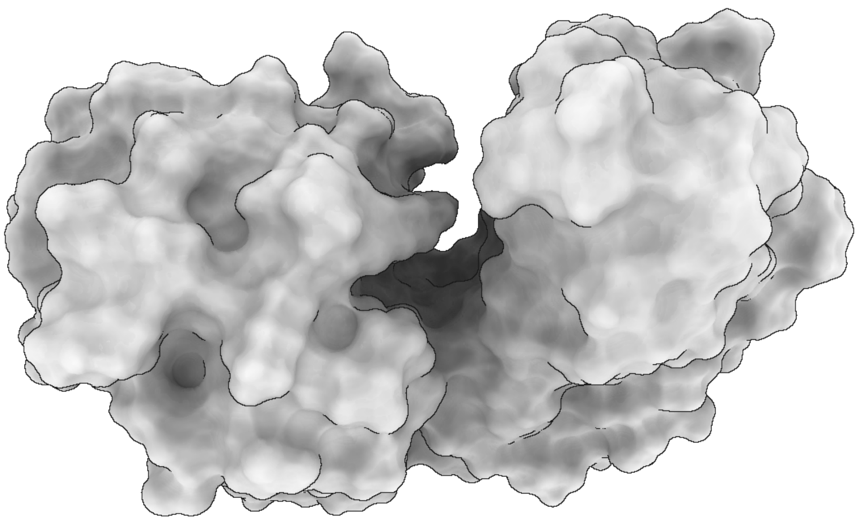


Figure 3.46: *Ambient occlusion factors computed for the SES model generated by the methods presented in this section. Note how the ambient occlusion algorithm improves the shape perception of the surface, highlighting the cavities, one of the key features of the SESs the experts are looking for during analysis.*

pyramid. Figure 3.46 illustrates the ambient occlusion factors computed for an SES.

3.4.2 Results

3.4.2.1 Performance

The presented technique was tested in different molecules using a workstation with the following configuration: Intel Core i7 PC, running at 3.5 GHz, with 16 Gb of RAM, and a GeForce 970 GTX, at a screen resolution of 1024×768 px. The results of these tests are presented in Table 3.6. The columns of the table present: the time spent sorting the atoms of the molecule, the time required to compute the coarse SES, the start and end resolution used for each molecule, the time required to compute the most refined version of the SES, the frames per second of the application during the refinement process, and the cell size of the higher resolution grid. The computation of the coarse version of the SES is performed in real-time even for big molecules up to 545 K atoms, while the computation of the most refined SES takes a bit more than 4 seconds in

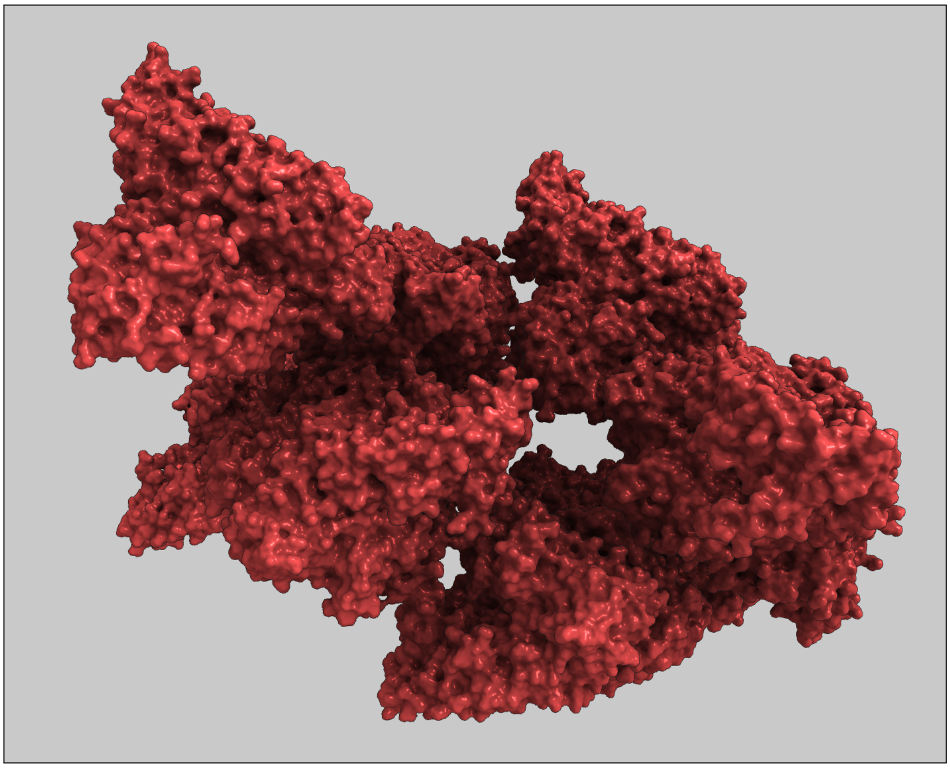


Figure 3.47: *SES of 3EXG molecule with 83 K atoms generated using a distance field resolution of* 512^3.

the worst case. During the computation of the different levels, the application is able to maintain a real-time frame rate, allowing the interaction with the intermediate results. The progressive SES computation thus enables a fast rendering of large simulation trajectories on the fly. Notice that for small test cases, like *Traj 3* in Table 3.6, our algorithm takes longer to complete the computation of the highest resolution, whereas times descend for larger molecules. For small molecules, the cells in the high-resolution grid are very small, and there is a large number of them inside the probe. Therefore, the algorithm needs to visit many neighbors and slows down. This can be easily avoided in practice since such high resolutions are not needed for small models. On the other hand, for big molecules like *1K4R*, the algorithm computes the most refined version in less than 2 seconds, but, however, the cell size used is remarkably higher, which can easily lead to missing features. In these cases, the algorithm proposed in Section 3.4.1.4 has to be used to avoid missing these tunnels and cavities.

Table 3.6: *Performance obtained for different molecules using the following hardware configuration: GeForce GTX 970 with 2 GB of memory, Intel i7 and 12 GB RAM. The different columns show the time required to sort the atoms in the regular grid [2], the resolution of the distance field in the base level [3] and the time in milliseconds needed to compute it [4], the resolution of the distance field used in the last refined level [5], the time in milliseconds needed to compute all the refined levels [6], the mean frames per second obtained during the refinement process [7] and, in the last column [8], the cell size of the last refined level.*

Molecule (#atoms) [1]	Sort (ms) [2]	l_s res. [3]	l_s (ms) [4]	l_e res. [5]	$[l_s, l_e]$ (ms) [6]	FPS comp. [7]	Cell (Å) [8]
Traj 1 (2,066)	0.16	64^3	5.42	256^3	775	51.61	0.21
Traj 2 (3,967)	0.20	64^3	5.23	256^3	464	79.74	0.28
Traj 3 (11,224)	0.47	128^3	10.09	512^3	4,149	45.55	0.20
2G47 (16,962)	0.60	128^3	9.72	512^3	2,151	90.63	0.29
1S3S (22,367)	0.80	128^3	7.58	512^3	930	133.29	0.40
3J3A (46,276)	1.55	128^3	8.94	512^3	1,822	136.66	0.43
3EXG (83,339)	2.18	128^3	8.81	512^3	1,242	165.06	0.51
1CWP (227 K)	6.24	128^3	11.52	512^3	2,468	164.51	0.58
1K4R (545 K)	11.96	128^3	15.89	512^3	1,649	177.08	0.98

Since no other methods to compute the SES calculate the result progressively we decided to compare our technique with the faster existing method, the GPU-based contour-buildup method by Krone et al. [KGE11], which is implemented using CUDA and included in the publicly available visualization system MegaMol [GKM+15]. The main drawback of the CUDA implementation of the contour-buildup in MegaMol is that it requires a huge amount of GPU memory. For the test data set 1S3S with 22 K atoms, it requires 1.9 GB of memory (the SES computation takes 32 ms on a Nvidia GTX 970), while a protein of about 30 K atoms (PDB ID: 3K19) requires already 2.7 GB of memory. Consequently, even newer consumer graphics cards with 4 GB VRAM or more will quickly run out of memory for large structures like virus capsids. This prevents the use of the CUDA-contour-buildup for such large structures. In contrast, our technique has almost constant memory consumption, making it very scalable. It requires just 620 MB for a data structure with a maximum

Table 3.7: *Performance in frames per second when rendering the distance field for different resolutions and molecules using the ray-marching algorithm. Moreover, the last column shows the frames per second of our method using ambient occlusion. When we compare our method with MegaMol (the column labeled as* RC *shows the performance in fps of that system), we see that our system is capable of achieving sustained interactive frame rates even for large molecules. The timings were computed by averaging the rendering time from 512 distinct random directions that uniformly sample a sphere, to even out variations due to camera placement.*

Molecule	#atoms	128^3	256^3	512^3	RC[KBE09]	$512^3 + \mathrm{AO}$
Traj 1	2,066	250.80	233.69	—	—	—
Traj 2	3,967	254.34	248.42	—	—	—
Traj 3	11,224	225.11	237.54	227.90	—	176.76
2G47	16,962	212.77	221.33	205.89	188.1	175.08
1S3S	22,367	229.86	223.73	212.74	126.3	151.46
3J3A	46,276	202.17	185.91	137.60	72.0	114.79
3EXG	83,339	200.18	202.94	180.56	69.2	137.81
1CWP	227 K	175.47	167.71	158.93	27.7	110.57
1K4R	545 K	165.71	162.27	144.35	12.6	76.40

resolution of 512^3, which is constant for all the molecules, and 32b for each atom of the molecule. Moreover, the CUDA-contour-buildup, as was pointed out before, requires 32 ms to compute a molecule with 22 K atoms (1S3S), while our algorithm is able to present a coarse version of the SES in one-fourth of the same time.

Moreover, we compared the rendering speed of our method to that of MegaMol. While our method uses volume ray-marching (see Section 3.4.1.6), MegaMol uses GPU-based ray casting to render the implicit patches of the SES [KBE09]. A comparison of the frame rates can be found in Table 3.7. Despite having similar performance for up to medium-sized proteins (2G47 of 16,962 atoms), ray-marching clearly outperforms ray-casting for very large data sets even when using high-resolution volumes. The rendering performance on MegaMol is limited by the number of implicit patches generated (i.e. the number of atoms), while our rendering method, on the contrary, depends on the grid resolution and the shape of the molecule.

Furthermore, we compared the performance of our algorithm to that of

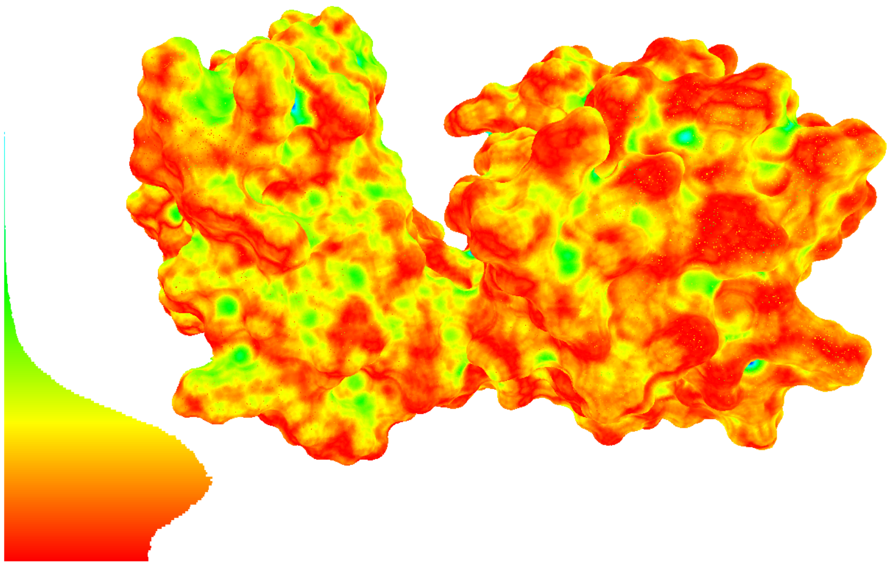


Figure 3.48: *Visualization of the distance between our result and the analytical solution. The distance is represented by a color scale from red (distance 0) to blue (distance equal to the probe radius). On the left part of the image, a histogram of the distances is shown.*

EDTSurf [XZ09] (see Table 3.8) since this algorithm also uses a grid to compute the SES. Although the comparison is not precise, we only use powers of two as grid resolutions and EDTSurf uses the CPU instead of the GPU to compute their solution, it showed that our algorithm is consistently faster. In all the tests our algorithm took less than one second to compute the SES using a grid resolution of 256^3, being more than 20 times faster than EDTSurf in most of the cases.

3.4.2.2 Evaluation

Lastly, and to ensure the correctness of our results, we compared the SES generated by our method with the exact SES. The exact surface was obtained using Chimera for a molecule of 3,967 atoms (this computation is not practical for very large molecules on this software), and the result of our algorithm was obtained using a resolution of 256^3. To this end, we sampled 2,676,613 random points uniformly distributed over the surface given by our algorithm and computed their distance to the exact surface generated by Chimera [PGH+04] (see Figure 3.48). The result was an average distance of 0.231743Å with a

Table 3.8: *Performance comparison between our method and the EDT-Surf software [XZ09]. The table shows the resolution of the discretization used by the EDTSurf algorithm and the time in seconds needed to compute the SES with the CPU implementation provided on their website. We compared these results with the time required by our technique to compute (progressively) the SES until a grid resolution of* 256^3 *was reached (we use power-of-two grid resolutions and they do not).*

Molecule	#atoms	EDTSurf Grid Resolution	EDTSurf Time (s)	Our method at 256^3 (s)
Traj 1	2,060	$203 \times 181 \times 172$	1.64	0.78
Traj 2	3,967	$217 \times 194 \times 271$	3.22	0.46
Traj 3	11,224	$275 \times 225 \times 299$	5.19	0.20
2G47	16,962	$201 \times 299 \times 234$	3.34	0.16
1S3S	22,367	$254 \times 299 \times 89$	1.99	0.09
3J3A	46,276	$299 \times 287 \times 299$	4.44	0.19
3EXG	83,339	$216 \times 225 \times 299$	3.64	0.16

root mean square error of 0.269498Å. Notice that this amounts to a standard deviation of roughly 0.14Å, so even if there are points at zero distance of the exact SES, and others at up to 1.43Å, they are extremely rare (see histogram on Figure 3.48). This means that our algorithm yields a surface with a small outward bias (of the order of magnitude of the spacing between samples) originated by its conservative nature. Occasionally, it may miss some small cavity, but this will only happen if inspecting a whole molecule, at a scale at which this cavity cannot be seen.

3.4.3 Conclusions

This section presented a new GPU-accelerated algorithm that computes the SES on the fly. In contrast to previous approaches, we are able to progressively refine the result, which allows its calculation and rendering at interactive frame rates for very large models like the virus capsid showed in Figure 3.49. Our system requires no precomputation and thus, we can handle dynamic models. The progressive component of our algorithm is achieved using a space discretization. At first sight, this might be considered a disadvantage, since too coarse a refinement might lead to missed cavities or incorrect surface shapes.

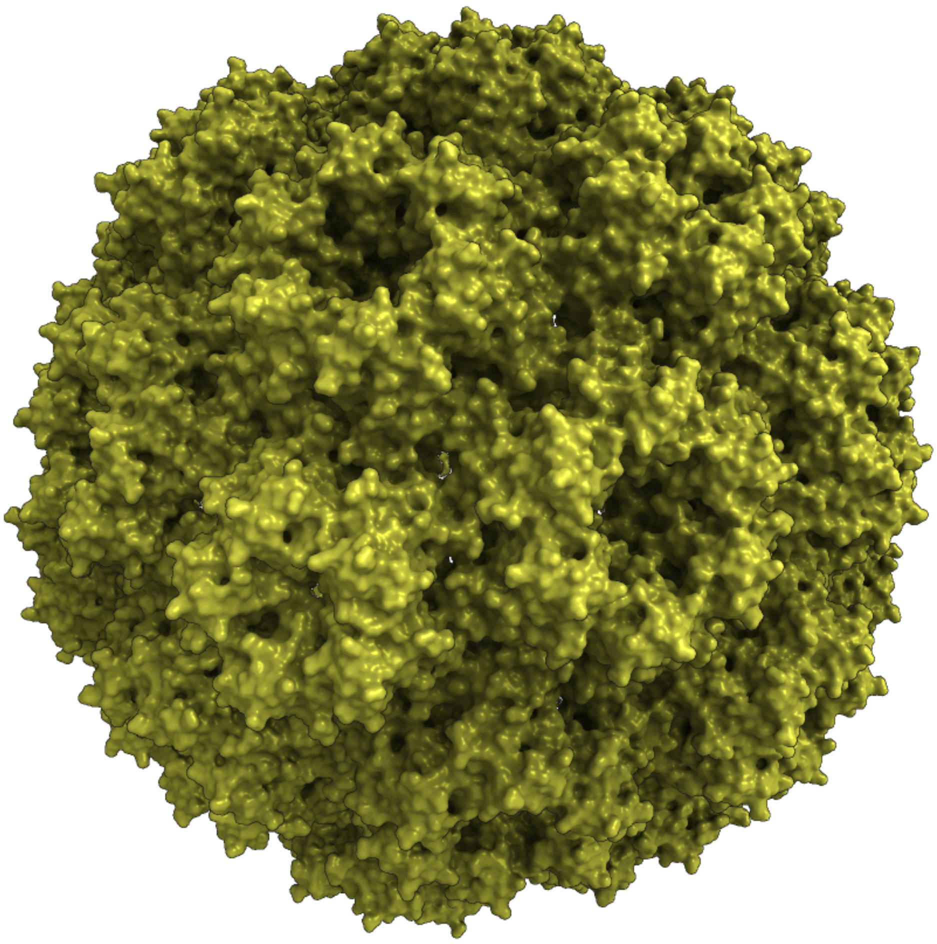


Figure 3.49: *SES of the virus capsid 1CWP with 227K atoms generated using a distance field resolution of* 512^3.

However, the resolution we use is fine enough to avoid such problems, as shown by the tests where we compared the surface obtained by our method to the analytical solution and found that they only had small discrepancies (see Figure 3.48). Moreover, we proposed a method to detect and compute the missing features skipped by our algorithm when the resolution is not fine enough. Another advantage of our approach is that we support smooth transitions between refinement levels, thus, the progressive improvement happens seamlessly. Finally, we have also shown how we can encode atom properties on the generated surface using color to support visual analysis.

4 Visualization of molecular interaction forces

Molecular design procedures, such as drug design and protein engineering, are complex processes, largely benefiting from computational resources but also from the human analysis. In drug design, for example, a costly iterative loop involves simulations requiring long computation times, followed by a data analysis phase, which is conducted by domain scientists using numerical analysis and visualization tools. Once some clues favoring or hindering binding have been understood, the ligand is modified by taking into account these clues, and another iteration is performed. In typical cases, computer simulation times range from hours to weeks, depending on the complexity of the molecules and the methods used. In our scenario, instead, the simulation software running on a supercomputer allows the computation of many simulation paths at the same time. This amount of available data implies that the requirements for analysis tools become more demanding.

As a result of the computational resources getting more affordable, human resources are becoming more and more the limiting factor in the computer-assisted molecular design process. In fact, most of the data analysis is performed in meeting rooms where different specialists discuss the outcomes of the simulation and the next design step. To enable these experts during comprehension and decision making, it is of great importance to provide effective data examination and visualization tools. In this chapter, we will focus on one of the key aspects required to make informed decisions in the molecular design process: understanding which parts of the molecule influence the binding of the ligand. This information is key, as it enables the domain expert to hypothesize which residues can be altered in the subsequent design process in order to improve the ligand's affinity. Unfortunately, communication of this information results in several challenges. First, the binding information must be

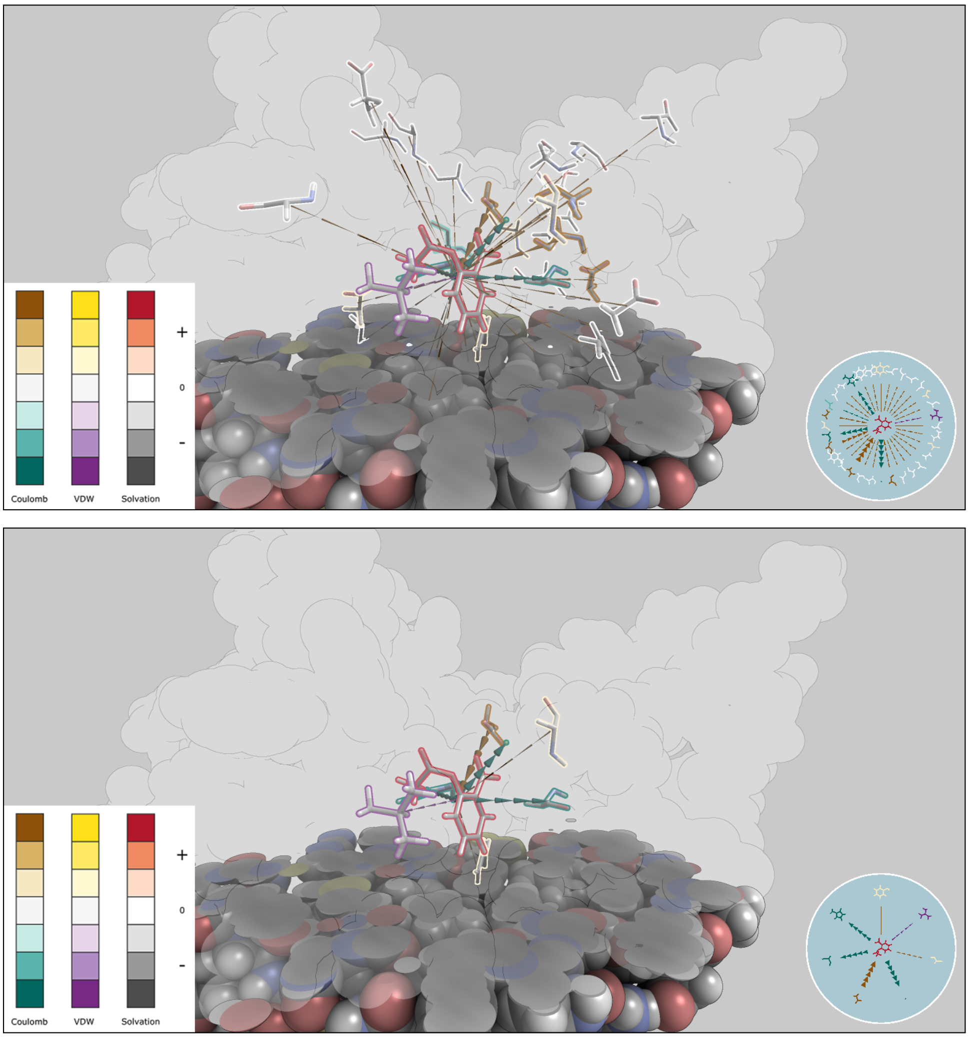


Figure 4.1: *Successive steps during the visual analysis of the binding of aspirin and the phospholipase A2 protein. We compute and visualize all essential interaction energies represented by 2D and 3D arrows. The orientation of the depicted arrows encodes the sign of the energy, i.e., attracting vs. repelling force. The width of the arrows, as well as the color of the residue's silhouettes, support energy quantification. During the visual analysis, energies are computed and depicted on-the-fly to support interactive hypothesis testing (top), and residues can be filtered based on energy and distance to obtaining a more focused view (bottom). Additionally, a 2D visualization helps to obtain total energy values in an uncluttered manner (Figure 4.2).*

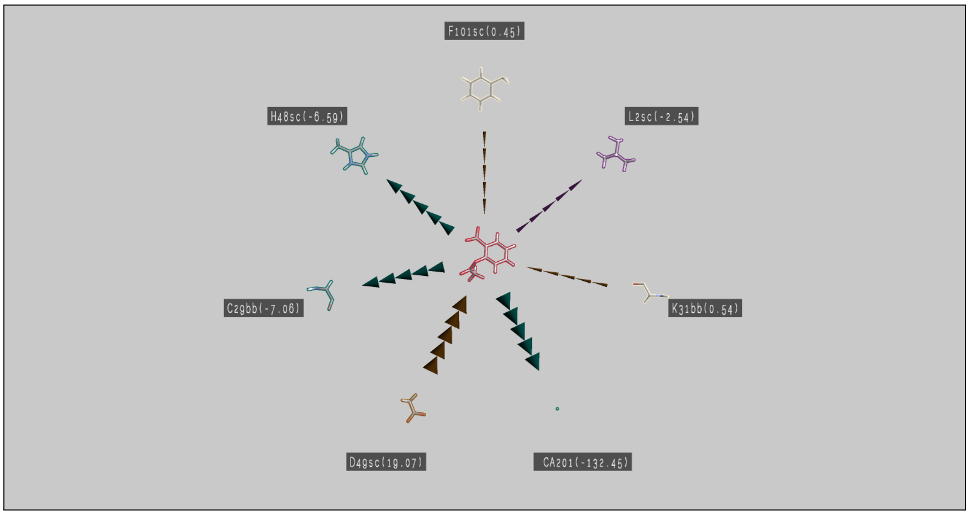


Figure 4.2: *2D abstract representation of the main energies involved in the simulation step visualized in Figure 4.1 (bottom).*

available instantly; for instance, if the domain expert selects a new step of the simulation, this information must be updated. Second, visual clutter, resulting from the multitude of displayed forces, which especially arises when considering long-range forces, needs to be reduced. Third, domain experts must be able to identify the involved residues and to quantify the related energies. When considering the usually dense representation of complex molecules, it becomes clear that a single 3D visualization will not be able to meet all these challenges. Therefore, we combine 2D and 3D visualizations together with brushing-and-linking in order to communicate which residues influence the ligand. Furthermore, we propose how to perform real-time computations of the three main energy components, for the energy model used, which enables the domain expert to explore entire trajectories consisting of multiple snapshots interactively. Moreover, with the proposed brushing-and-linking setup, it becomes possible for the first time also to analyze long-range interactions, which play an important role when a ligand is initially approaching a molecule. We not only hope that this long-range analysis sheds new light on the entire docking process but also expect that it helps to reduce required computation resources, as it allows for early intervention with the running simulation. Thus, we support a more effective, computer-based molecular design process by making the following main contributions:

- We propose visual analysis techniques for the real-time computation and inspection of interaction energies arising between a molecule's residues and the ligand.

- We propose a linked visualization setup communicating the computed interaction energies, by reducing visual clutter and enabling direct identification of the individual residues.

- We enable domain scientists through the means of brushing-and-linking to explore the underlying interactions, which in particular allows them for the first time to also inspect long range energies.

As illustrated in Figures 4.1 and 4.2, we combine these contributions, such that the user can interact with the input data to gather new knowledge, to formulate and assess hypotheses, and provide visual explanations of the discoveries.

The rest of the section is organized as follows: Section 4.1 addresses the design requirements and discuss the application background; Section 4.2 describes the algorithm used to calculate the different energy components; the visualization motifs and techniques used to communicate these energies are presented in Section 4.3; in Section 4.4 we study how our application improved the understanding of the underlying forces driving the docking process on different simulations and we also compare our application with the available software; and, lastly, in Section 4.5, we present the conclusions.

4.1 Application-Driven visualization design

Before describing the different features of our application, in this section, we provide, first, a discussion of the background of our application (the energy model used) and, second, a description of the different visual design requirements.

4.1.1 Background

In computational drug design, as well as in other molecular modeling areas, a key aspect is to estimate the interaction energy between the protein and the ligand (or substrate in, e.g. enzymatic catalysis).

The free energy G of the protein-ligand system is a powerful tool to understand the binding process. Computing ΔG_{bind}, the difference of free energy between the bound state and the unbound state where the protein and ligand stay free in the solvent, can be used as a measure of the binding strength or affinity between them (more negative values of the energy mean a stronger binding). Typical energy models are additive, allowing to understand the main contributions or the key interactions that would favor (or disfavor) ligand binding. The energy model used by our application is based on three terms:

- *Van der Waals interaction energy*, $E_{\text{inter,VDW}}$, which shows how well the protein and ligand molecules pack together.

- *Electrostatic interaction energy* (the interaction in vacuum plus the solvent screening), $E_{\text{inter,ele}}$, which shows the strength of the interaction between the protein and ligand charges, screened by the effect of the solvent.

- *Change in solvation energy*, ΔG_{solv}, which shows how much the protein and ligand prefer to be bound together, instead of being free in the solvent (regarding exclusively the interaction with the solvent molecules, and including entropy).

So the binding energy of a given simulation step is computed as follows:

$$\Delta G_{\text{bind}} = E_{\text{inter,VDW}} + E_{\text{inter,ele}} + \Delta G_{\text{solv}} \tag{4.1}$$

A more detailed development of the formula for ΔG_{bind} leads to:

$$\begin{aligned}\Delta G_{\text{bind}} = E_{\text{inter,vdW}} + E^{\text{vacuum}}_{\text{inter,ele}} + \Delta G^{\text{solv}}_{\text{inter,pol}} + \\ \Delta G^{\text{solv}}_{\text{rest,pol}} + \Delta G^{\text{solv}}_{\text{np}}\end{aligned} \tag{4.2}$$

where the *Electrostatic interaction energy* ($E_{\text{inter,ele}}$) expands into the interaction in vacuum ($E^{\text{vacuum}}_{\text{inter,ele}}$) plus the solvent screening ($\Delta G^{\text{solv}}_{\text{inter,pol}}$), and the *Change in solvation energy* (ΔG_{solv}) expands into a polar ($\Delta G^{\text{solv}}_{\text{rest,pol}}$) and a non-polar term ($\Delta G^{\text{solv}}_{\text{np}}$).

The Van der Waals and Electrostatic interaction terms are calculated using the OPLS 2005 force field [BBC+05], while the polar solvation energy follows the generalized Born model OBC [OBC04], and the ACE model [SBK98] is used for the non-polar solvation energy. Together, they create an energy model which deals with the protein and ligand in atomic detail, while considering the solvent as a continuum. The equations used to compute each term, derived from the different models listed above, are the following:

$$E_{\text{inter,vdW}} = \sum_{\substack{i\in\text{protein}\\ j\in\text{ligand}}} 4\sqrt{\epsilon_{ii}\epsilon_{jj}}\left(\frac{(\sigma_{ii}\sigma_{jj})^6}{r_{ij}^{12}} - \frac{(\sigma_{ii}\sigma_{jj})^3}{r_{ij}^6}\right) \tag{4.3}$$

$$E^{\text{vacuum}}_{\text{inter,ele}} = \sum_{\substack{i\in\text{protein}\\ j\in\text{ligand}}} \frac{q_i q_j C}{r_{ij}} \tag{4.4}$$

$$\Delta G^{\text{solv}}_{\text{inter,pol}} = -\sum_{\substack{i\in\text{protein}\\ j\in\text{ligand}}} (1-\frac{\mathrm{e}^{-\kappa f_{ij,\text{GB,bound}}}}{\varepsilon_{\text{solv}}})\frac{q_i q_j C}{f_{ij,\text{GB,bound}}} \tag{4.5}$$

$$\begin{aligned}\Delta G^{\text{solv}}_{\text{rest,pol}} = &-\frac{1}{2}\sum_{\substack{j\neq i\\ (i,j\in\text{protein or}\\ i,j\in\text{ligand})}}\left[(1-\frac{\mathrm{e}^{-\kappa f_{ij,\text{GB,bound}}}}{\varepsilon_{\text{solv}}})\frac{q_i q_j C}{f_{ij,\text{GB,bound}}} -\right.\\ &\left.-(1-\frac{\mathrm{e}^{-\kappa f_{ij,\text{GB,unbnd}}}}{\varepsilon_{\text{solv}}})\frac{q_i q_j C}{f_{ij,\text{GB,unbnd}}}\right]-\\ &-\frac{1}{2}\sum_i\left[(1-\frac{\mathrm{e}^{-\kappa\alpha_{i,\text{bound}}}}{\varepsilon_{\text{solv}}})\frac{q_i^2 C}{\alpha_{i,\text{bound}}} -\right.\\ &\left.-(1-\frac{\mathrm{e}^{-\kappa\alpha_{i,\text{unbnd}}}}{\varepsilon_{\text{solv}}})\frac{q_i^2 C}{\alpha_{i,\text{unbnd}}}\right]\end{aligned} \tag{4.6}$$

$$\Delta G^{\text{solv}}_{\text{np}} = \sum_i 4\pi b_i(R_i+R_s)^2\left[\left(\frac{R_i}{\alpha_{i,\text{bound}}}\right)^6 - \left(\frac{R_i}{\alpha_{i,\text{unbnd}}}\right)^6\right] \tag{4.7}$$

where some subterms expand as follows:

$$f_{ij,\text{GB,bound}} = \sqrt{r_{ij}^2 + \alpha_{ij,\text{bound}}^2\mathrm{e}^{-D_{\text{bound}}}} \tag{4.8}$$

$$D_{\text{bound}} = \frac{r_{ij}^2}{(2\alpha_{ij,\text{bound}})^2} \tag{4.9}$$

$$\alpha_{ij,\text{bound}} = \sqrt{\alpha_{i,\text{bound}}\alpha_{j,\text{bound}}} \tag{4.10}$$

$$f_{ij,\text{GB,unbnd}} = \sqrt{r_{ij}^2 + \alpha_{ij,\text{unbnd}}^2\mathrm{e}^{-D_{\text{unbnd}}}} \tag{4.11}$$

$$D_{\text{unbnd}} = \frac{r_{ij}^2}{(2\alpha_{ij,\text{unbnd}})^2} \tag{4.12}$$

$$\alpha_{ij,\text{unbnd}} = \sqrt{\alpha_{i,\text{unbnd}}\alpha_{j,\text{unbnd}}} \tag{4.13}$$

$$\kappa = 0.73\times10^{-10}\sqrt{\frac{2N_A e_{\text{unit}}^2 1000 I}{\varepsilon_{\text{solv}}\varepsilon_0 k_{\text{B}} T}} \tag{4.14}$$

and the constants and variables are defined as:

- $C = 332.0637$
- r_{ij} : distance between atoms i and j (in Å).
- q_i : charge of atom i (in atomic units).
- q_j : charge of atom j (in atomic units).
- ϵ_{ii} : OPLS epsilon parameter for atom i (in kcal/mol).
- ϵ_{jj} : OPLS epsilon parameter for atom j (in kcal/mol).
- σ_{ii} : OPLS sigma parameter for atom i (in Å).
- σ_{jj} : OPLS sigma parameter for atom j (in Å).
- N_A : Avogadro number. $N_A = 6.02213670 \times 10^{23}$.
- e_{unit} : Unit charge of a proton (in Coulombs). $e = 1.602177330 \times 10^{-19}$.
- I : Ionic strength (in mol/liter). For these studies, we used $I = 0.1\,\text{mol/L}$
- k_{B} : Boltzmann constant (in J / K). $k_{\text{B}} = 1.3806580 \times 10^{-23}$
- T : Temperature (in Kelvin). In these formulas, a room temperature of 298.0 K is used.
- ε_0 : Permittivity constant of vacuum, $8.8541878160 \times 10^{-12}\,\text{Coulombs}^2/Jm$
- $\varepsilon_{\text{solv}}$: Relative permittivity constant of solvent: 80.0.
- $\alpha_{i,\text{bound}}$: Born radius of atom i (in Å) in bound state.
- $\alpha_{j,\text{bound}}$: Born radius of atom j (in Å) in bound state.
- $\alpha_{i,\text{unbnd}}$: Born radius of atom i (in Å) in unbound state.
- $\alpha_{j,\text{unbnd}}$: Born radius of atom j (in Å) in unbound state.
- R_i : Atomic radius of atom i parameterized for ACE model.
- R_s : Solvent probe molecule radius (1.4,Å).
- b_i : ACE solvation parameter for the atom i.

These parameters are mostly constants and some of them are defined for each type of atom, as ϵ_{ii}, σ_{ii} or q_i. However, the Born radius of each atom change at each step of the simulation and it has to be recomputed. To compute the Born radius of an atom we used the definition given by Onufriev et al. in [OBC04], where the radius is computed through small contributions from the rest of the atoms of the system. Thus, to compute the Born radius of all the atoms of a simulation step, this method needs to compute N^2 small contributions where N is the number of atoms of the system.

The Van der Waals (equation 4.3) and Electrostatic interaction terms (equations 4.4 and 4.5) model the interaction of the ligand with the protein, involving only computations between pairs of atoms where one belongs to the ligand and the other to the protein. Since the number of atoms of the ligand is usually small, we can consider it as a constant, so that the cost of computing these terms is linear in the number of atoms of the protein, $\mathcal{O}(n)$. The polar solvation energy term (equation 4.6), on the contrary, has a quadratic cost in the number of atoms of the protein, $\mathcal{O}(n^2)$. This term is composed of two summations. The first one adds the contributions to the change in solvation energy of all pairs of atoms in the protein and all pairs of atoms in the ligand, which leads to a quadratic computational cost in the number of atoms of the protein (since we considered the number of atoms of the ligand a constant). The second summation adds together the change of solvation self-energy of each atom of the system, being linear in the number of atoms of the protein, $\mathcal{O}(n)$. Therefore, the computational cost of the polar solvation energy term is $\mathcal{O}(n^2 + n) = \mathcal{O}(n^2)$. The non-polar solvation energy term (equation 4.7), as the second summation of the polar term, evaluates the energy change in each atom of the system, which translates to a linear computational cost too.

Since the interaction energies are visualized for each pair of ligand and protein chemical group, we compute the interaction energy of each pair by adding all the corresponding interactions between any pair of atoms, one belonging to the ligand and the other to the protein chemical group, as shown in the above formulas. The change in solvation energy of a given chemical group, instead, is the sum of its atomic values; as shown above. Some solvation contributions are actually calculated for a pair of atoms, so those energies are divided among the two atoms.

4.1.2 Design requirements

In this chapter, we propose visualizations which have been developed with the goal to help domain experts understand the forces acting in molecular processes, through analyzing the main components of the binding energy. In drug design, for example, domain experts need to analyze whether the ligand will or will not dock at the intended position. To answer this question, domain experts have to inspect the numerical results of the main interaction energy components which are usually provided in result tables. While such a table could be analyzed for a single ligand, comparing them for several ligands is not practical. However, since it is often necessary to study more than one bound structure, as an ensemble of structures will aid in a better characterization of the bound complex, domain experts need to be able to analyze this type of data effectively. In addition, studying the drug migration pathway, from the solvent to the bound complex, might better help in addressing the binding (or its absence) mechanism and locate key interactions that could facilitate (or hinder) binding [EHB+15]. Similar conclusions can be observed in enzymatic catalysis [AFFM+16]. The detailed mechanistic knowledge provided by the binding energy analysis should locate those parts of the molecule/receptor that enhance or prevent docking, facilitating the following design steps.

The interaction of molecules is atom-based, but atoms group naturally into residues or chemical groups accountable of collective responses, such as ionic groups (carboxylic groups, etc.), aromatic groups (phenyl, etc.) or an entire residue, for which one is often interested in the whole group interaction. This is the reason why we analyze interaction energies in chemical groups and/or residues, not single atoms.

With the help of domain researchers, we have identified the following questions as being essential to be answered in the visual analysis process:

- Q1: Which are the most active groups in the interactions between molecules?
- Q2: Which are the most powerful binding energy components at a certain simulation configuration?
- Q3: Is the proximity of the drug causing instability in any residue of the protein?
- Q4: Is the ligand solvation force favoring or rejecting binding?
- Q5: Which residues (if any) prevent drug delivery?

By creating a visualization method that illustrates the different components of the molecular interaction energies, we enable domain experts to answer these and other questions, thus gaining a detailed knowledge of the binding mechanism. Importantly, our system not only computes the energy components on the fly but also provides a series of filters that let the user select distances or energy ranges to inspect them in fine detail. To support such an interactive visual analysis, besides effective visualizations, an efficient implementation is essential. Therefore, we exploit a data structure computed by the GPU that facilitates queries such that residues and groups can be filtered in real-time. The filtered information is then communicated using the proposed visualization techniques. This way we can provide an interactive visual analysis system that lets the user inspect the set of energies that are interacting at any time in a simulation. In the following sections, we will describe this system by focusing on the two main parts:

Interaction energy calculation: To compute the interaction energies on the fly, a GPU-based set of programs is used that computes an array of energies for each residue (and for each atom in the case of solvation energy), each time a new step of the simulation is selected (or a structure is loaded). Furthermore, a specialized energy data structure is used to accelerate the queries issued by the domain expert through the visualization front end.

Interaction energy visualization: During the visualization phase, the domain expert can interact with the data by means of widgets. These provide several visualization motifs and filters that facilitate the data analysis and inspection, as well as data presentation.

4.2 Interaction energy calculation

Given a configuration of the molecules, the computation of the energies involved is not inexpensive. Some of the quantities involved (the Born radius and the polar solvation energy,, $\Delta G^{\text{solv}}_{\text{rest,pol}}$ in Equation 4.2, change due to interacting charges) have a cost which is quadratic in the number of atoms. Since we want our application to be able to adapt to changes in the configuration (because of user interactions or a change of frame in a simulation trajectory) interactively, we employ GPU computation at each change. Therefore, when the user selects a new step of the simulation, first we dispatch the computation of the Born radius to a set of compute shaders. In order to compute the Born radius of each atom, we must visit all other atoms in the molecule, yielding a quadratic cost. Each thread of our compute shader calculates 16 of these interactions and adds the result to a shader storage buffer using atomic operations. Once the interactions are computed, and all the individual contributions are added for each atom, another compute shader is resort. This compute shader executes a thread for each atom of the molecule where its final Born radius is computed based on the interactions with the rest of the atoms previously calculated. Note

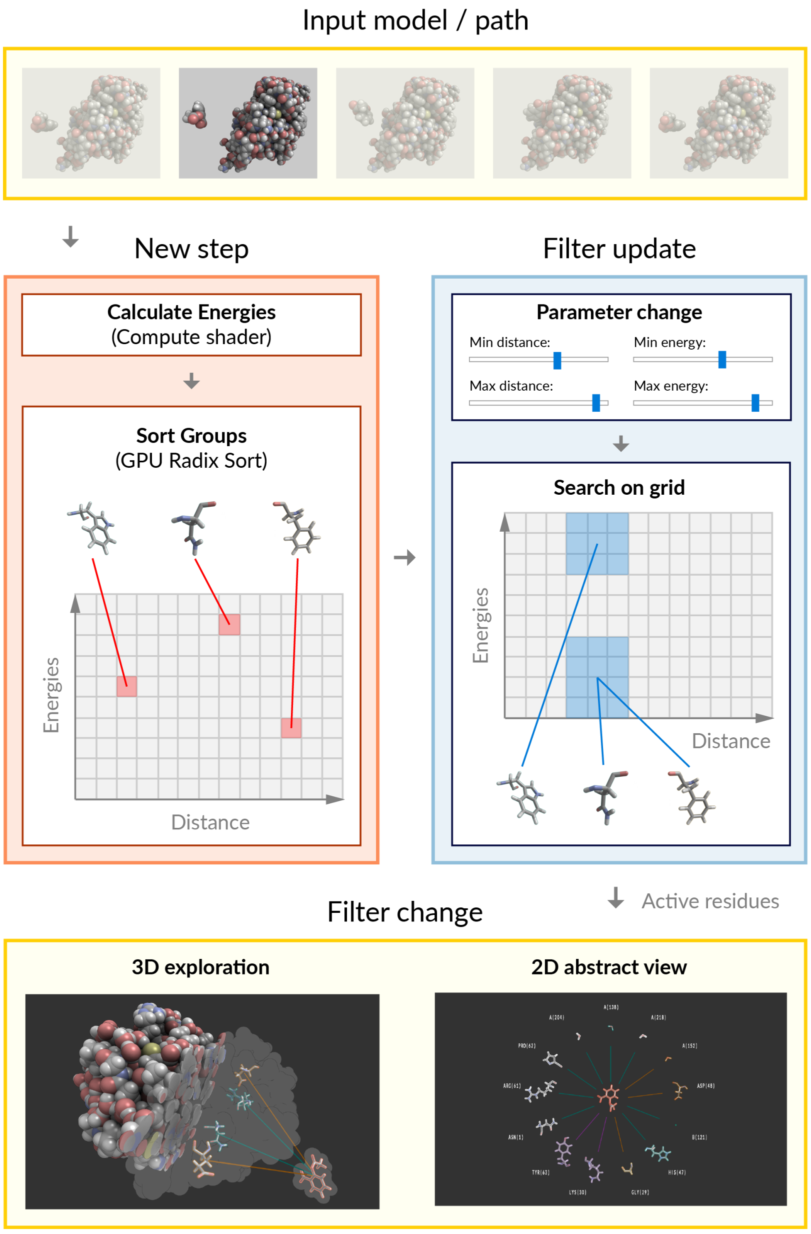


Figure 4.3: *Overview of the data flow underlying our application. When the current step or the configuration change, the system automatically computes the forces being exerted by each residue in the compute shaders* (left)*. When the user modifies the filters, the selected residues are quickly gathered from the indexing data structure computed by the previous step* (center) *and then used to render the 3D or the 2D abstract views* (right)*.*

that this radius is computed in parallel for the bound and unbound state at the same time.

After computing the Born radius, our system computes the solvation energy terms for each atom (using equations 4.6 and 4.7). Again, equations 4.6 and 4.7 require N^2 separate operations, so we execute another compute shader to perform this task, assigning 16 of these computations to each thread as in the Born radius calculation.

When all of these terms have been computed for all atoms, we launch another compute shader that computes the VDW (equation 4.3) and Electrostatic terms (equations 4.4 and 4.5) and the final value of the energy between the ligand and the atom groups. Each thread of this compute shader is responsible for computing these energies for a single interaction between the ligand and an atom group. With this strategy, we are able to compute the energy of the system more than 20 times per second for molecules up to 18 K atoms (using a computer with a processor Quad Core i7 at 3.7 GHz, 16 Gb of RAM and a GeFroce GTX 980).

The previously described method allows the fast computation of all the energies involved in the interaction between the ligand and the protein, but visualizing all of them at the same time generates a too cluttered image where they are hardly identifiable. To reduce the amount of information shown, we provide different filtering methods based on the energy of the interaction and the distance between the ligand and the atom group. Iterating over all the atom groups to check which ones meet the filtering requirements can slow down the performance of the application when the number of groups is too large. To ensure an interactive filtering, we build an auxiliary data structure that allows us to find the selected groups in constant time. This data structure is a 2D grid composed of 128×128 cells, where each one stores the groups within a certain energy and distance range. Both energy levels and distances are discretized into 128 possible segments with an energy range of [-3.0, 3.0] and a distance range of [0, 50]. With those parameters, we assure a fixed number of groups per cell for all the examples we have tried. In order to distribute the atom groups among the cells, each time the configuration changes, or the user shifts his attention to a different aspect turning on or off some energy component, the array holding the energies of all the groups is sorted according to the key of the cell where the atom groups belong. This sorting key is constructed concatenating the energy and the distance bucket index (this is represented by the bottom portion of the "New step" box in Figure 4.3). The sort is carried out using a radix sort algorithm, implemented in four compute shaders corresponding to the three steps in [HH11] plus an additional step to building the table which holds in each cell the start and end position in the buffer of the entries corresponding to that cell.

Once the sort is complete, the results obtained are downloaded to the CPU. When the user modifies some of the filter ranges (“Filter update” box in Figure 4.3), the cells containing the chosen ranges are determined, and using the index table, all the groups in those cells are checked for the current filter ranges (as the cells will contain some neighbor values as well, because of the discretization). The filtered groups are made visible and, for each group, an identifier according to their energy level is computed to determine the appropriate colors of the silhouette. The start and end points of the connection arrows are also uploaded to the GPU with the same identifier that will determine their color.

4.3 Interaction energy visualization

The workflow in our system has the following steps (see Figure 4.3): The user opens a file that contains a structure or a simulation path. At this point, and every time the user switches between path frames (“New step” box), the GPU calculates the energy terms, updates the energies-distances data structure and downloads it to the CPU. Then, the user may freely update the filters. When those are changed, the data structure is queried (“Filter update” box) to determine active groups, energies, and so on. The user can then freely inspect the 3D and 2D views or update the filters again. All of this happens in real-time. In this section, we describe how the visualization tool has been designed and implemented.

4.3.1 Idioms and filters

In order to support domain experts in answering the questions listed in Section 4.1.2, we have developed interactive visualizations communicating the binding energy factors computed in real-time. By employing filtering, we can ensure that only residues currently of interest appear in the view. The visualization of these elements is then enhanced with visual idioms that provide information so that domain experts can easily understand the important forces in the current step of the simulation path. We have realized this interactive visual analysis by including the following visual idioms: filtering, focus and context, feature enhancement, and interaction. Most of them are illustrated in Figure 4.4.

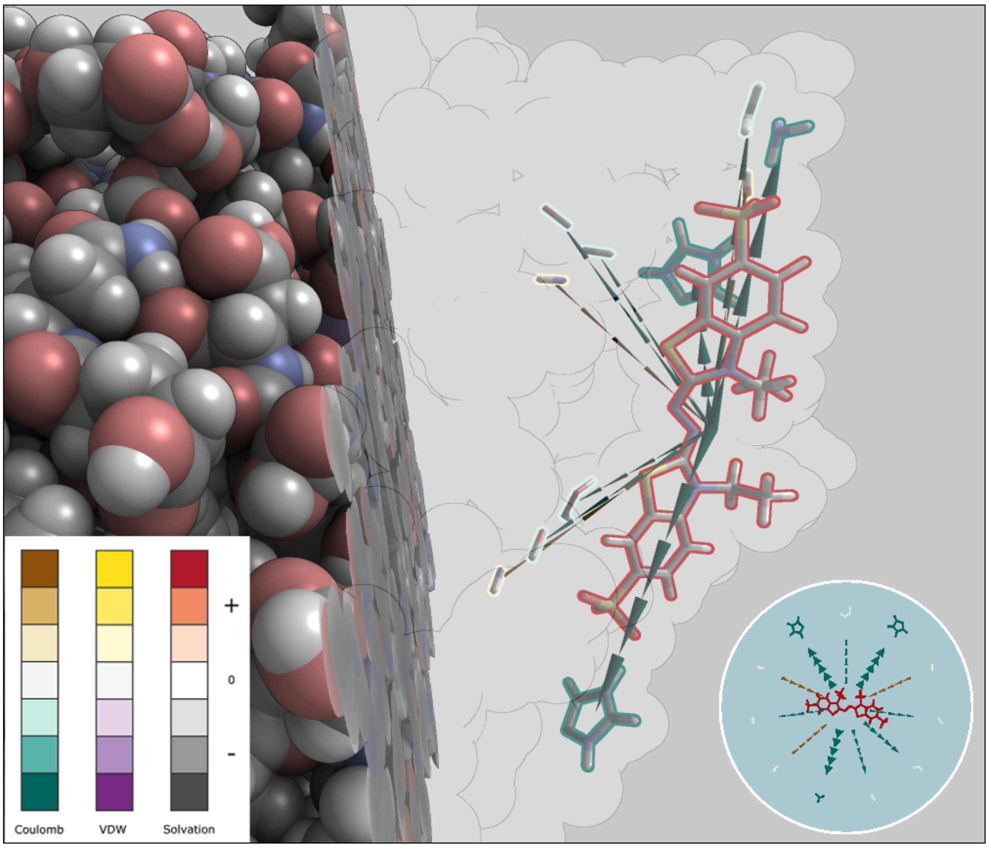


Figure 4.4: *The snapshot corresponds to the docking position of an artificial substrate, ABTS, to the manganese peroxidase 4. In Figure 4.7 and Figure 4.8 the docking path is illustrated. This zoomed view shows the main idioms used to communicate energy: cones for the direction of forces, their thickness and color to encode energy and intensity, and colored highlights of residues as described in Section 4.3.*

4.3.1.1 Filtering

In several cases the interaction energies are omnipresent, despite the fact that for several groups the absolute energy is rather low, e.g., the ligand will have electrostatic interactions with most of the groups of the protein. If all these energies were communicated, our visualization would be too cluttered. Therefore, we support filtering to restrict the energies and the groups to be visualized to those which fulfill certain criteria. Currently, we support three types of filters: *i)* distance filtering, *ii)* energy level filtering, and *iii)* energy type filtering.

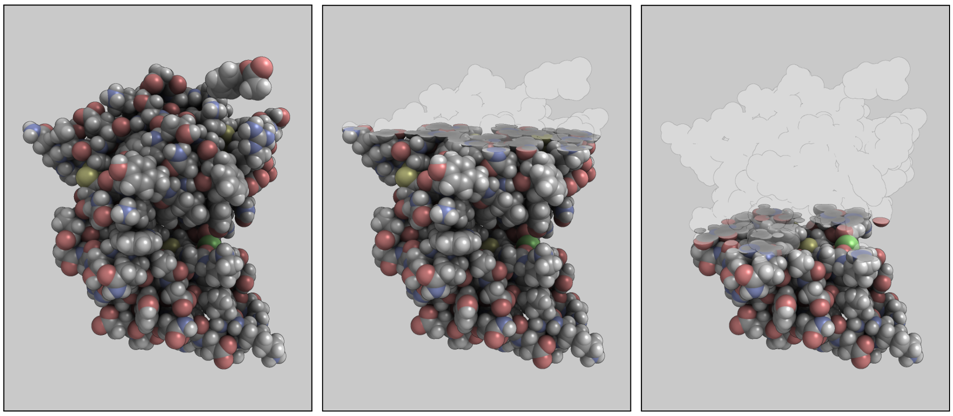


Figure 4.5: *In order to provide context to the ligand and the interaction groups, we provide a clipping plane which determines the part of the molecule that will be visible and the part that will be rendered using transparencies. These figures illustrate how the user, through a slider, modifies the value of the clipping plane, moving the plane along the molecule.*

Thus, by selecting any (or all) of the energy types, changing the range of distances at which interactions are considered and the amount of energy, the user can finely analyze individual or group interactions. In Figure 4.1 we have applied distance filtering to confine the visualized residues to those having a short distance interaction with the ligand only.

4.3.1.2 Focus and context

The central entity that guides any exploration within our application is the energy enhancement. As a result, we always enhance the groups that are active, i.e., whose energy is within the limits determined by the used energy level filter. However, to embed the currently selected groups, it is important to add context to the focused elements. Therefore, we provide the visual context in two flavors: *i)* Eliding information using a user-defined clipping plane, and *ii)* Superimposing a layer that renders, using semi-transparency and silhouette enhancement, the information concerning the non-active groups (see Figure 4.5). We allow the user to define the plane direction by selecting the current view orientation, and, once the plane direction has been fixed, the user is able to move the plane along it using a slider. Moreover, the parts of the molecule that are clipped away by the plane are then rendered using the semi-transparent layer. Both context idioms can be freely combined.

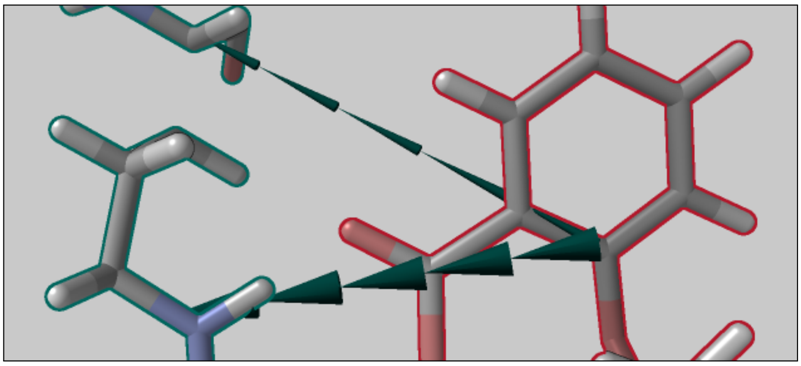


Figure 4.6: *Energy communication through a set of cones. This picture presents two different atom groups where one strongly interacts with the ligand while the other has a weak interaction. The magnitude of the interaction is encoded by the size of the cones. In this image, the cones on the bottom represent the stronger interaction while the cones on the top, thinner than the bottom ones, represent the weak interaction. Moreover, this image illustrates how both atom groups exert an attractive force on the ligand, which is represented by the orientation of the cones. The orientation of the cones indicates the direction in which the ligand would move due to the interaction. In addition, the cones encode the dominant energy component of the interaction in their color, being the electrostatic component the dominant one in the two examples of the image.*

4.3.1.3 Feature enhancement

In order to facilitate comprehension of the affecting energies, we color-code the dominant energy type on the silhouettes of the respective groups (see Figure 4.4), where each energy type is encoded using a color scale with two different hues (which represent the positive and negative ranges). Further communication of the energy and sign is provided by means of arrows, drawn along the axis that links the ligand with the active group. These cones are color coded with the hue of the energy type and indicate the direction in which the ligand would move due to a single interaction. The size of their base also encodes the amount of total energy, so harder interactions are more likely to stand out in cluttered scenes where many groups are active 4.6. For an individual energy analysis or only repulsion/attraction analysis, the user could also modify the application to highlight only the total energy using a color scale with two hues, where one hue represents the positive range and the other the negative.

4.3.1.4 Interaction

During the whole visual analysis process, the user may freely inspect the 3D view by modifying the viewpoint, zoom, pan, and so on. Moreover, the user can modify the filters interactively and, to visualize a trajectory, the path step can be manually selected or an animation can be triggered to see the full path. To support a more exact quantification, a 2D view, with details on the amino acids is also provided. This view is created by projecting all active residues in a circular layout around the ligand. By exploiting linking, we ensure that when filters are updated or the step of the path changes, the view is updated accordingly. Figure 4.7 and Figure 4.8 show the last steps of the docking path of an artificial substrate.

4.3.2 3D visualization

To provide visual cues for the users to understand the energies involving each residue at any point in the simulation path, we need to visualize both, the binding energy and the elements affecting it. Therefore, we choose a default representation that lets the user identify the groups and facilitates the incorporation of energy information around the molecules.

4.3.2.1 Atom representation

We chose to render the active elements using the licorice method since it represents the molecules with a set of thin cylinders and spheres to reduce the footprint on the screen. This representation is a good balance between space occupied and information. Since the cylinders are encoded with the traditional colors of the atoms, they are easy to identify, and additional information is communicated via a thick silhouette around them. In order to reduce clutter, only the active residues or groups are shown, and to provide 3D context, the rest of the molecule is visualized using van der Waals surfaces. To do not occlude the active groups a user-defined cutting plane (described in Section 4.3.1.2) is used to determine which part of the molecule will be rendered semi-transparent and which part will be rendered opaque. Moreover, we compute ambient occlusion factors for each pixel (using the algorithms described in Chapter 3), providing thus a better understanding of the location of the groups within the molecule.

We selected these representation methods as they provide the best balance between communication of the selected atom groups and understanding of the molecule conformation. However, our application allows the modification of these representation methods, permitting any combination of the methods described in Chapter 3. Examples of different configurations are presented in

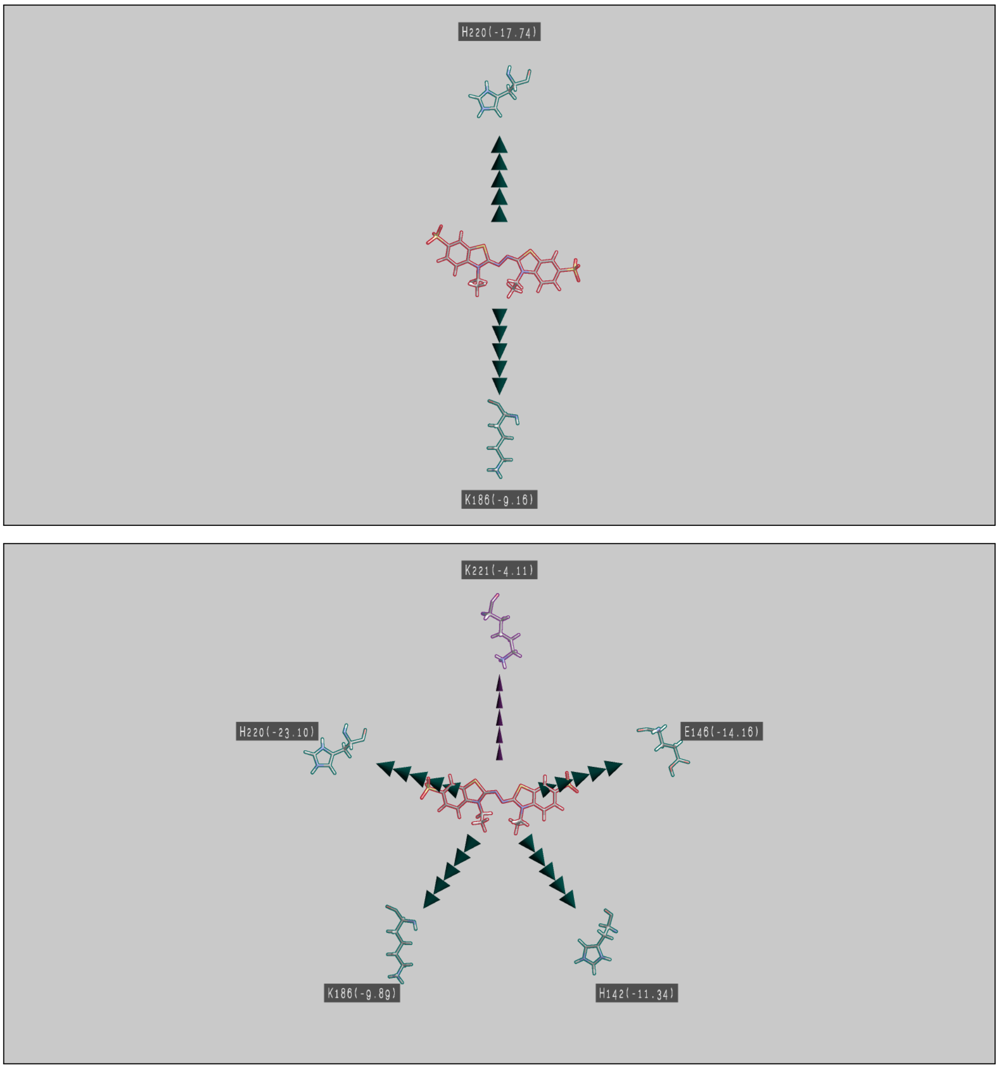


Figure 4.7: *The interaction through the 2D views (Figure 4.7 top, Figure 4.7 bottom, Figure 4.8 top and Figure 4.8 bottom represent four steps of the simulation) visually explains the docking procedure of the ABTS, an artificial substrate, to the manganese peroxidase 4. Once the histidine 220 (H220) has established an attraction connection (top), the substrate does not leave the surface of the protein and finally docks (Figure 4.8 bottom), also attracted by a lysine (K186) and another histidine (H142). This can be seen interactively by hoping between path steps.*

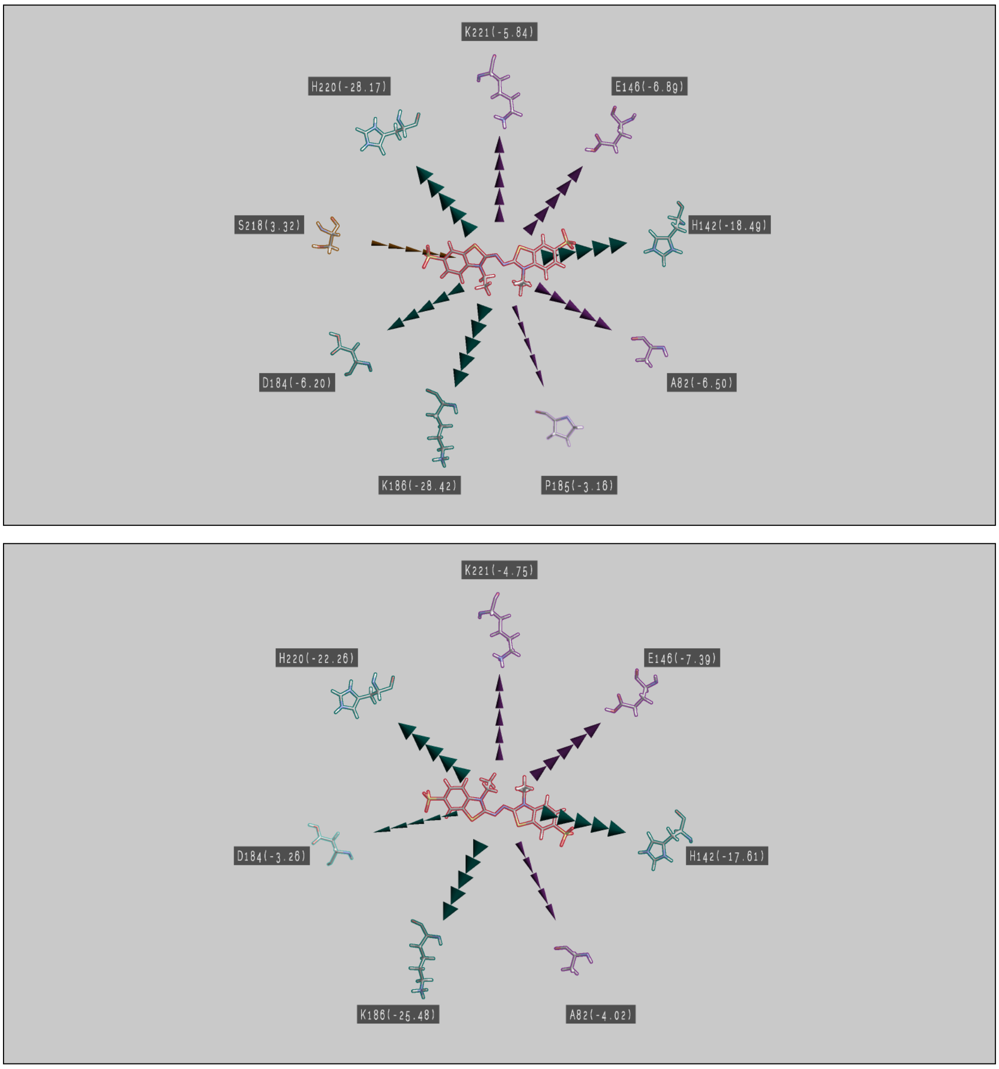


Figure 4.8: *Last steps of the docking procedure of the ABTS, an artificial substrate, to the manganese peroxidase 4. The first simulation steps are illustrated in Figure 4.7.*

Figure 4.9.

4.3.2.2 Energy representation

Energies are typically signed, so we will use diverging hue representations to show them. The hues for the range selections fulfill the following requirements:

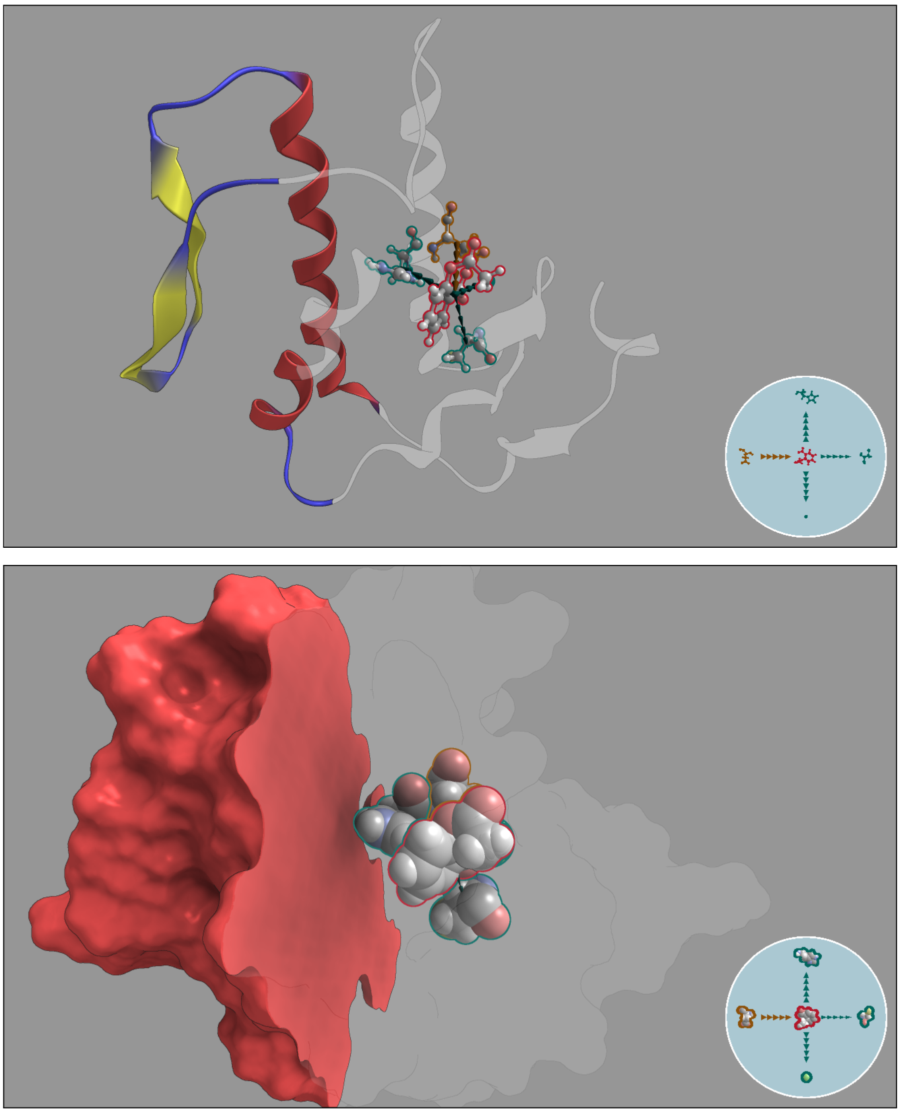


Figure 4.9: *Alternative representation methods. The top image presents a possible visual configuration where the context is given by the molecule represented with ribbons and the ligand and the interacting groups represented with the Balls & Sticks model. The bottom image presents another possible configuration. Here the context is given by the molecular SES and the ligand and interacting groups are represented with Space-filling model. Although we use licorice and Space-filling as our default configuration, we give the opportunity to the user to freely select the representation method to use, since each method has its benefits and drawbacks.*

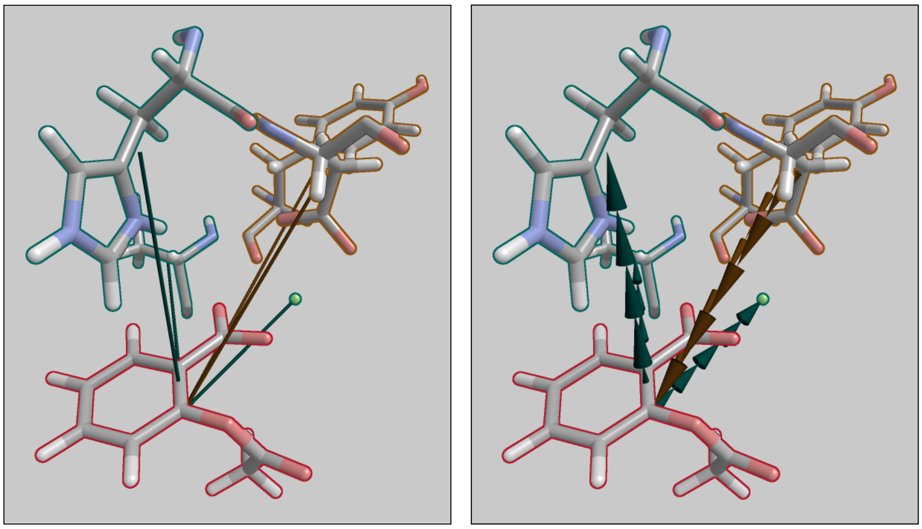
i) Avoid blue-red hues, since these are commonly used for polarization, *ii)* Avoid the colors commonly used to represent proteins (e.g. some gray, red and blue hues), *iii)* enhance color distinction by reducing the amount of tones and using perception-based selections, *iv)* avoid the repetition of hues to make them unambiguous. Under these conditions, we decided to represent the dominant energy of each interaction with a 7-point diverging hue scale where white represents neutral or close to neutral values.

We selected green-brown for electrostatic energies (typically spread in both sign directions), violet-yellow hues for van der Waals energies keeping the violet hues for negative, far more common in vdW than positive values. Finally, a gray-desaturated red scale was used for solvation energy. The solvation energy is encoded in the color of the silhouette of the ligand since it is an important information that communicates whether the ligand is comfortable in the solvent or uncomfortable, which might favor binding. Although having the same neutral color for all the energy types might seem confusing, the energies represented by these colors are not relevant for the understanding of the simulation as their magnitudes are small (see Figure 4.1). This guides the attention of the domain experts to the groups with high magnitudes, as they are represented with highly saturated colors. These magnitudes are also communicated through other more precise means (see Visualization Configuration below, and the 2D view described in Section 4.3.3).

In all cases, the energy magnitudes are rendered as a thick silhouette around the licorice representations of the active bonds. These color combinations and molecular representations result in a quite understandable way to encode interactions, facilitating the comprehension of simulation results. The main idea behind this is to avoid the requirement of checking other regions of the screen (tabular representations of values are also commonly used) and thus keeping the attention of the user on the task.

4.3.2.3 Visualization configuration

To further guide the attention of the users to active groups, we also highlight the interactions with geometric elements that go from the center of the ligand to the center of the group of interest. The user can select between two types of geometric objects, cylinders or cones (see Figure 4.10). The first ones connect the ligand and an atom group by a cylinder, whose color indicates the sign and type of dominant energy of the interaction (selecting the first or last color from the corresponding diverging scale). The second ones draw a set of cones between the ligand and the atom groups, whose orientation indicates the direction the ligand would follow as a consequence of the influence of the



(a) Lines. (b) Cones.

Figure 4.10: *Geometry objects encoding the interaction energy. We provide two methods to represent the interaction between the ligand and the atom groups (besides the silhouette color): (a) Lines and (b) Cones. Both of them encode the dominant energy term of the interaction in their color, but the cones also encode the sign and magnitude of the total energy in their direction and size. Moreover, we allow the user to activate and deactivate this feature since these objects can lead to too cluttered images.*

corresponding group of interest. Besides of using the same coloring method as the cylinders, the cones have base areas proportional to the total energy level. To avoid possible occlusions introduced by these objects, the user can freely toggle these elements on and off.

Furthermore, the user can activate a set of overlays which communicate the actual value of the energy involved in the interaction together with a code that identifies the atom group (see Figure 4.11). The code is composed of first the letter identifying the residue type followed by the residue number. Then a two letter code indicating if the atom group belongs to the backbone (*bb*) or the side-chain (*sc*) of the residue is added at the end.

4.3.3 2D visualization

When generating a 2D projection of the groups of interest, it is important to facilitate an easy mental linking with the 3D visualization. As a consequence,

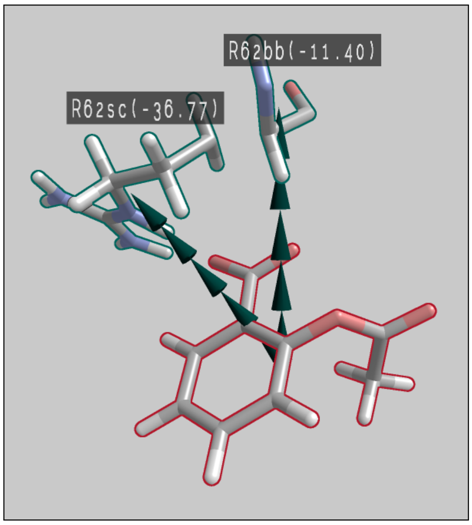


Figure 4.11: *Our tool provides a set of overlays which present additional information. This information is composed of the residue type letter, the residue index within the structure, a two letter code indicating if the atom group belongs to the backbone (*bb*) or the side-chain (*sc*) of the amino acid, and the energy value of the interaction.*

our proposed algorithm for generating the 2D visualizations takes into account the 3D arrangement of the groups and exploits the same connections (cones or cylinders) as used in the 3D view, as well as the same color coding for the silhouettes of the groups. The individual steps of the algorithm can be summarized as follows:

1. Calculate the vector that goes from the center of the ligand to the center of the scene. This vector is then used as the direction of the virtual viewing plane of step 4.

2. Determine the number of active groups.

3. Subdivide the virtual space around the ligand into as many equal sectors as there are active groups.

4. For each group, calculate the projection to a virtual viewing plane centered in the ligand, and exploit clockwise sorting to assign the respective partition.

5. Project the residues at a fixed distance from the center, and centered in their sector, maximizing the projected area.

To facilitate the interpretation of the projected groups, we project each residue to the 2D view with a (different) projection direction that maximizes its area. The optimal direction is achieved by performing a *Principal Component Analysis*[1] of the group's atoms' positions, whereby the smaller eigenvector of the matrix determines the optimal projection direction. With this strategy, we achieve an ordering of the residues that is directly related to their 3D position in space, thus facilitating the inspection in both 2D and 3D views at the same time.

Furthermore, we draw the overlay indicating the id of each residue and the energy value together with its representation, packing in a single view all the relevant information for the current step. Figure 4.13 shows a single simulation step represented by this 2D view.

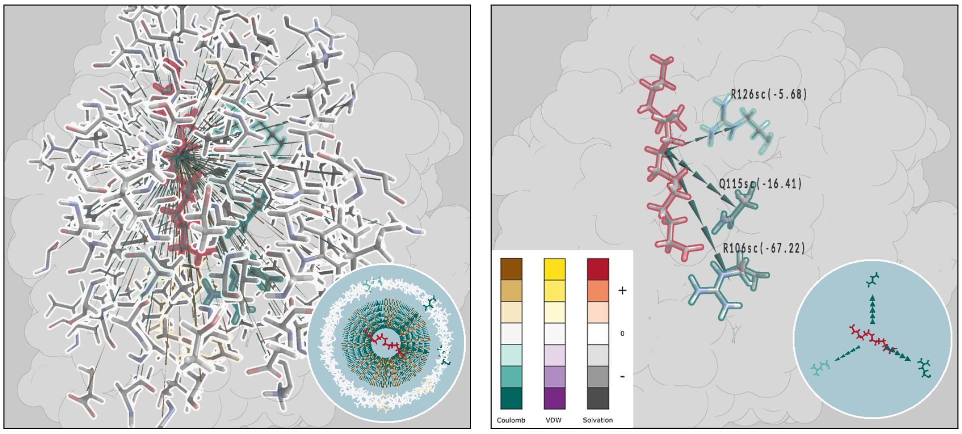
4.4 Application cases

The proposed visual analysis techniques have been integrated into an application which is flexible and offers a large range of features, and thus allows to analyze different aspects of biomolecular interactions, with applications, for example, in drug design and protein engineering processes. We can analyze data from the point of view of the agnostic scientist, just trying to gather new knowledge, or we can use it to assess some hypothesis. Usually, hypothesis testing will lead to simpler scenarios because we already have an initial guess about which parameters to analyze. In the following subsections, we will discuss application cases describing how the presented visual analysis techniques enable new insights. We will start by discussing a single conformation analysis process, before discussing the insights achieved when applying our approach to a more complex trajectory analysis.

4.4.1 Single conformation analysis

An initial scenario where we can use our visualization is to understand which molecular forces are predominant in a given structure, such as the bound conformation obtained from measurements or simulations. The proposed visualizations enable to spot key residues and chemical groups in the interaction between protein and ligand, both enhancing or opposing binding. This information is paramount for the scientist in order to understand the binding

[1]Principal Component Analysis is a technique used to reduce the dimensionality of the data, which, in the case of computer graphics, can be used to approximate a set of points in $\mathbb{R}^3$ by a plane with a minimum error. For more information, the reader can address to [Jol14].



(a) Initial exploration step.

(b) Energy restriction to absolute value below -5 kcal/mol.

Figure 4.12: *Interaction between the palmitate ligand and the intestinal fatty acid protein. Initial exploration does not let us see the important interactions happening close to the palmitate acid. By carefully filtering energy terms larger than -5 kcal/mol, we can see how palmitate interacts strongly at electrostatic level (dark green interactions) with several residues, notably with arginines (R126 and very strongly with R106) as shown in the image. The values between parentheses indicate the total energy.*

mechanism, as well as to choose ligand groups in drug design or protein amino acids in enzyme engineering for mutation, whereby the mutations are performed with the goal to improve or disfavor molecular interaction. Suggested changes can later be confirmed or discarded by analyzing a new simulation with the modified protein-ligand system.

With the following example, investigating the binding of palmitate, a fatty acid ligand, to the intestinal fatty acid-binding protein, we further illustrate how we can quickly assess molecular interaction hypotheses from a crystal structure inspection. The bound structure was obtained from a Nuclear Magnetic Resonance Spectroscopy (NMR) experiment (PDB id 1ure, [HPC96]), after minimization with the OPLS-AA force field using the PELE [MSG13]. This system is interesting since palmitate's binding could show important contributions from the electrostatic, vdW and solvation energy terms, and our analysis is focused on question Q1, asking for the most active groups. Thus, the first hypothesis is that several arginine (Arg or R) residues, which are positively charged amino acids, should have an important role at the electrostatic level because they are in the vicinity of the negative polar extreme of the fatty

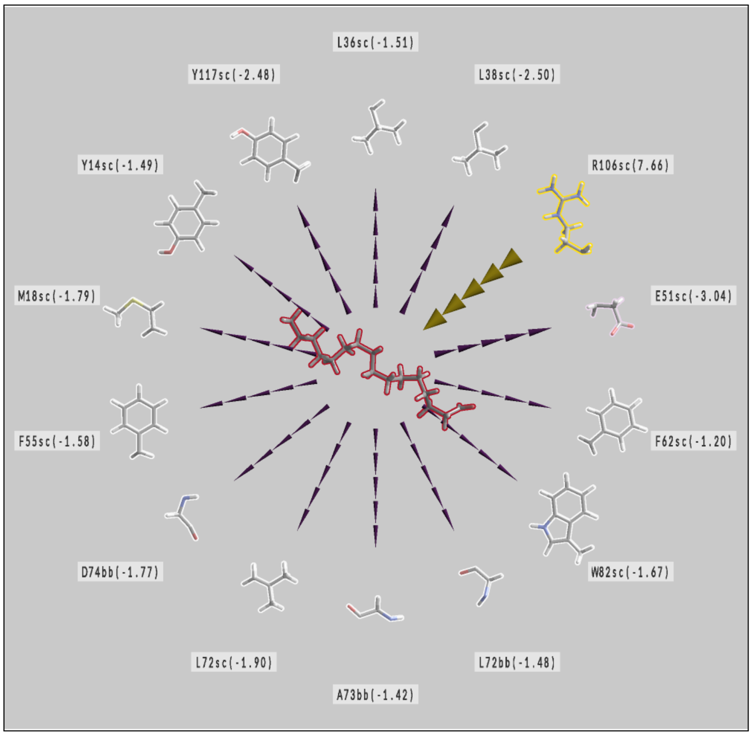


Figure 4.13: *Interaction of the palmitate ligand with several leucines and phenylalanines residues at vdW level. Note the strong yellow silhouettes that indicate strong interactions.*

acid, a carboxylic group. We can verify this assumption in Figure 4.12, where we checked all Coulomb contributions lower than -5 kcal/mol. Through the visual analysis, three arginines having large electrostatic stabilizing contributions can be clearly identified, whereby the strength directly correlates with the cone radii. This is also seen in Figure 4.12, where the numerical value of each contribution indicates that Arg106, the one closer to the polar ligand group, is the main stabilizing residue.

A second hypothesis that can be easily tested is the nature of some of the vdW interactions. Since palmitate has a long aliphatic tail, some hydrophobic residues should have important vdW interactions, such as leucines (Leu or L) and phenylalanines (Phe or F). We can assess this by inspecting the vdW energies as shown in Figure 4.13. Here, several contributions can be identified, and we can see the interacting residues, i.e., leucine side chains (sc) and backbones(bb) (L36sc, L38sc, L72bb, and L72sc), and phenylalanine side chains (F62sc and F55sc). Interestingly, we can observe a strong destabilizing vdW

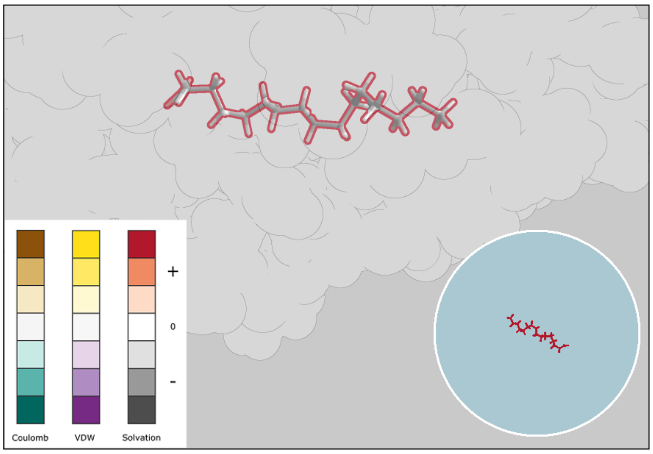


Figure 4.14: *The solvation term in the palmitate ligand, when inside the protein, indicates that, contrary to the hypothesis, it does not favor binding. Note the high energy value color coded in its silhouette. Being a fatty molecule, the expected value would be low (gray).*

component from arginine 106 (R106), induced by the large ionic attraction seen above. This example constitutes a nice (didactic) illustration of force field terms balance and is directly related to our question Q2 asking which ones are the most powerful binding energy components.

We can see another example of how the visualization can help to confirm or reject hypotheses. In this case, related to question Q4, questioning the solvation force, the fatty acid has a long aliphatic chain. Consequently, it is expected that removing it from a water environment, which is polar, and placing it in its bound protein conformation would be associated with a (de)solvation energy gain. However, when using the visualization to assess the results, the domain experts were surprised by a desolvation loss instead. As a consequence, a further detailed study was triggered and then it was discovered that the charged Carboxylate group in the ligand actually opposes this, and the total effect is a desolvation energy loss. This finding could be made by referring to Figure 4.14, where all the energies are removed except the solvation force that is encoded in the silhouette of the ligand. We can observe that the energy is strongly positive, and can thus reject the hypothesis that solvation energy terms would facilitate palmitate's binding.

4.4.2 Trajectory analysis

A more complex scenario involves the analysis of multiple structures, obtained, for example, from molecular dynamics or Monte Carlo simulations. We can run the whole ligand migration path, asking the application to highlight residues with dominating interaction energies at each frame of the simulation by simply stepping through simulation time. Note, that the term frame usually refers to a step in the simulation performed for the study of the protein-ligand binding. Similarly, multiple experimental structures can be analyzed simultaneously.

Figure 4.15 and Figure 4.16 shows three different snapshots along the aspirin migration simulation in the phospholipase A2 protein. The top image in Figure 4.15 shows the ligand in the bulk solvent, far from the protein surface, where no energies acting on the ligand are detected. The bottom image in Figure 4.15 illustrates the ligand approaching the surface and how our visualization shows the initial protein-ligand recognition forces, two long-range electrostatic contributions that guide the ligand towards the protein. Electrostatic forces are dominant usually when the ligand is relatively far from the protein. This is one example of question Q1 mentioned earlier. Here we also observe again the key stabilizing role of the calcium ion (green cones) together with some minor destabilizing electrostatic contributions (brown cones) from other calcium coordinated groups (having the same sign as aspirin). This partially deals with our questions Q2 and Q3, asking for the most powerful binding energy component and about drug-induced instability. In addition, we can clearly observe some vdW smaller interactions from hydrophobic residues with the aromatic group of the aspirin ligand. This addition of van der Waals forces only appears at short range interactions. Finally, the top image in Figure 4.16 shows the ligand close to the docking position. We observe a strong interaction between the aspirin ligand and a calcium ion associated with the protein.

The crystal structure of the complex (PDB id 1oxr, [SEJ$^+$05]) shows, from a structural perspective, that this interaction exists. Our application enriches this information from an energetic point of view, as well as it allows to relate, in a qualitative manner, the strength of this interaction with that of the aspirin to other important interacting residues, such as histidine and aspartic acid residues. The calcium ion is clearly visible as the green sphere in the middle of the protein. An inspection of the abstract view (bottom image in Figure 4.16) reveals all the interacting residues, and we can see clearly how histidine (H48sc and H48bb) and aspartic acid (D49sc) residues are also interacting, as predicted. It can be further concluded, that some of these strong electrostatic attractive contributions, importantly, from the active site calcium (Ca) ion (with and overall +2 charge), are responsible for driving the ligand

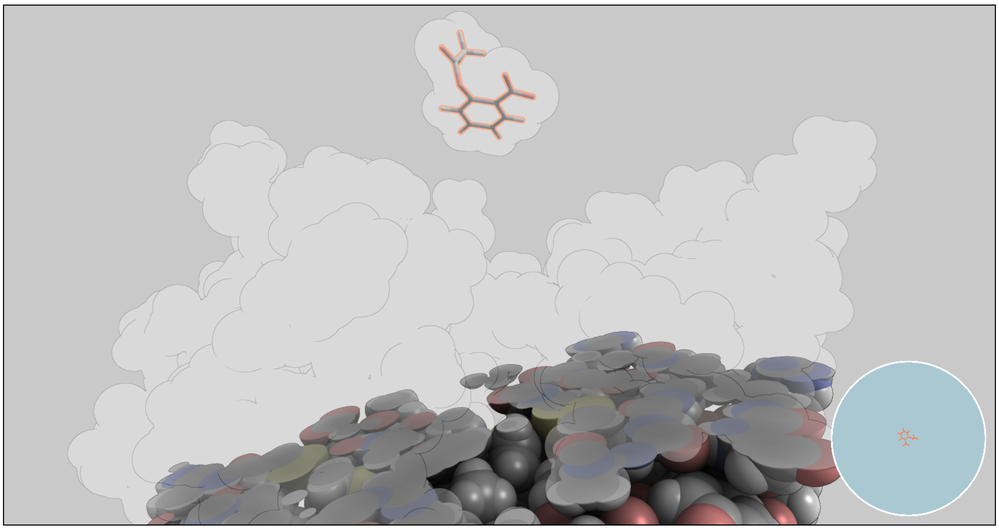
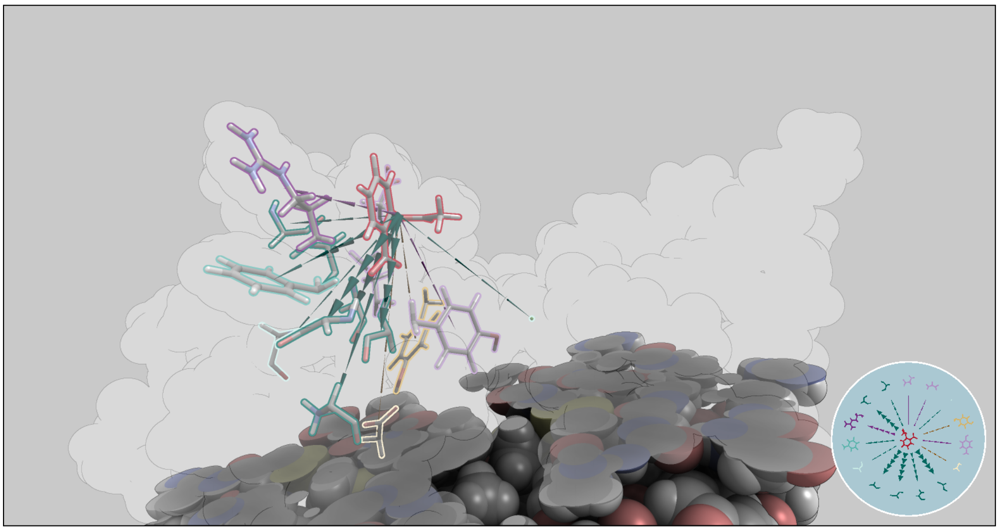



Figure 4.15: *Three different stages of the aspirin docking to the phospholipase A2 protein (following in Figure 4.16). The top image shows the aspirin in the bulk solvent, so no energies are exerted. The bottom image shows the ligand closer to the protein, thus electrostatic energies appear.*

to its final bound position. Thus, this study constitutes a nice example of how this tool allows studying the ligand binding mechanism at atomic detail.

As we could show in the discussed application cases, the proposed visual analysis techniques can be used to answer the stated questions Q1 to Q4.

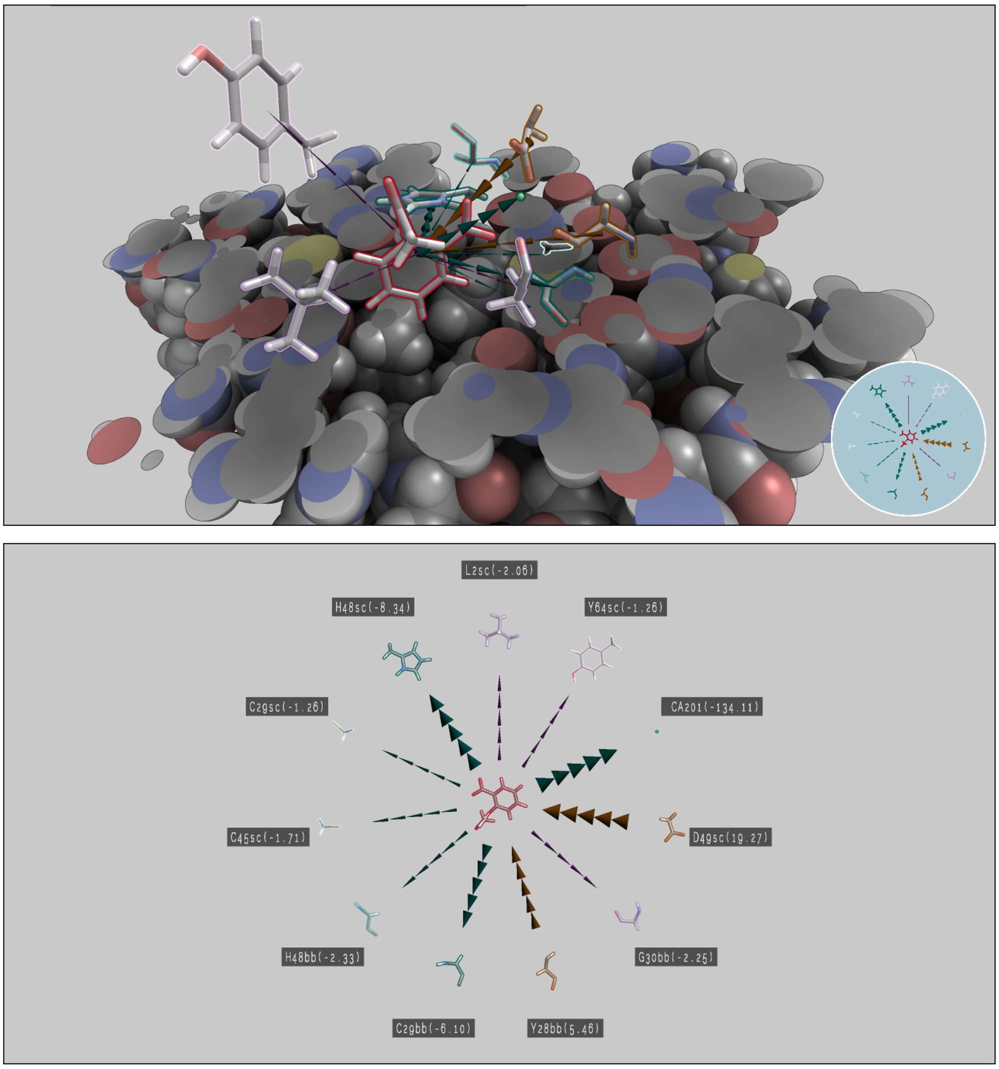


Figure 4.16: *The top image shows a frame near the docking position. At this point, the calcium is strongly attracting the aspirin (green thick cones), where other groups exhibit repulsive electrostatic energies (brown cones). The bottom image shows the abstract view with all the active residues.*

Unfortunately, we could not discuss any application case where we could answer question Q5, asking for residues preventing drug delivery. This is due to the fact that the simulation data analyzed in this work has already been analyzed before with conventional methods. Accordingly, only those simulations resulting in a successful binding were at our disposal.

4.4.3 Evaluation

The development of the system has been done in close collaboration with domain experts. However, we also asked other experts, external to our team, in order to gather opinions on the design and implementation of our visualization tool. In particular, we asked the opinion of two more experts: a chemist that works in protein engineering, and a computational chemist working in computer-aided drug design. After a small demo of the tool, we let them play for thirty to forty minutes with it. During and after the session, we gathered their comments.

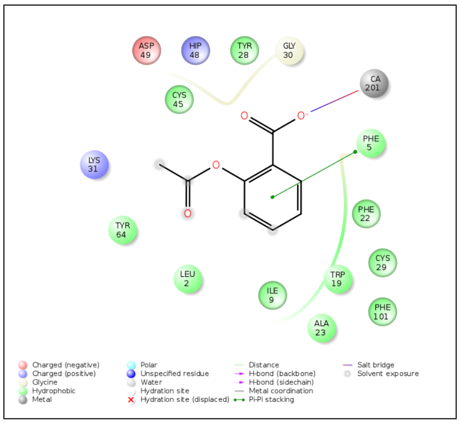
Both of them found the tool very useful, with large applicability in protein engineering tasks (e.g. to inspect the result of protein mutations and to analyze protein-protein interactions), molecular dynamics (to better understand the molecules behavior), electrostatic steering (in order to analyze the effect of electrostatic fields in guiding the ligand's path), and also enzyme engineering. Moreover, they also found the application useful for results presentation, to help other collaborators "understand what is really happening". They also commented on the interface, where they referred to it as "very intuitive", since the interacting groups are easily identified. Finally, the semi-transparent visualization of the molecule was deemed very interesting, since it helps grasping how the ligand is getting close to the molecule without rendering it completely.

From the design point of view, the use of cones of different sizes was found appealing because the users can identify the interactions and infer their strength quite simply. Furthermore, the domain experts valued the possibility of filtering the energy magnitudes, energy types, and distances very positively.

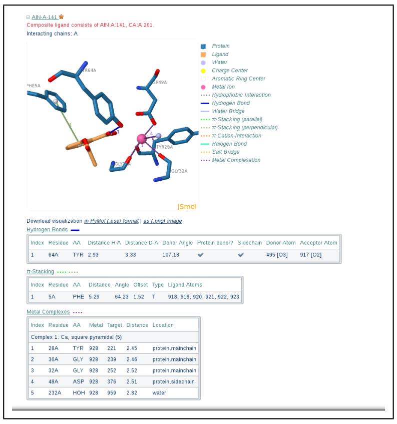
4.4.4 Comparison with other methods

To investigate the benefits of the proposed system, we have performed a comparison to previous systems capable of visualizing molecular interaction forces. Due to their popularity, we have included the Maestro system [Sch16] as well as PLIP in this comparison [SSH$^+$15]. To obtain a common frame of references, we have used both systems to visualize the binding between aspirin and the phospholipase A2 protein, an interaction visualized using our system in Figure 4.1, 4.2, 4.15 and 4.16. Figure 4.17 (a) shows the result achieved with Maestro, and Figure 4.17 (b) shows the result obtained with PLIP.

When comparing the results obtained with Maestro and PLIP to the results of our system, besides the different visualization design communicating interaction intensity and signage, three conceptual differences immediately become



(a) Maestro



Composite ligand consists of AIN:A:141, CA:A:201.

Interacting chains: A

Download visualization in PyMol (.pse) format | as (.png) image

Hydrogen Bonds

Index	Residue	AA	Distance H-A	Distance D-A	Donor Angle	Protein donor?	Sidechain	Donor Atom	Acceptor Atom
1	64A	TYR	2.93	3.33	107.18	✓	✓	495 [O3]	917 [O2]

π-Stacking

Index	Residue	AA	Distance	Angle	Offset	Type	Ligand Atoms
1	5A	PHE	5.29	64.23	1.52	T	918, 919, 920, 921, 922, 923

Metal Complexes

Index	Residue	AA	Metal	Target	Distance	Location
Complex 1: Ca, square.pyramidal (5)						
1	28A	TYR	928	221	2.45	protein.mainchain
2	30A	GLY	928	239	2.46	protein.mainchain
3	32A	GLY	928	252	2.52	protein.mainchain
4	49A	ASP	928	376	2.51	protein.sidechain
5	232A	HOH	928	959	2.82	water

(b) PLIP

Figure 4.17: *Full results obtained from Maestro (a) and PLIP (b) when using the systems to visualize the binding of aspirin and the phospholipase A2 protein.*

clear, and we analyze them one by one.

First, while our system employs linked views in order to communicate the interacting forces, Maestro and PLIP use a single view only. Interestingly, Maestro uses a 2D visualization, while PLIP uses a 3D visualization. We see this as an indicator that both 2D and 3D have their benefits, and that our design considering the linking of a 2D and a 3D view might be helpful. Furthermore, we believe that having two linked views makes it more intuitive to map the binding information to the 3D structure of a molecule.

The second major difference is the consideration of time-varying data. While Maestro and PLIP show a single frame only, our system enables researchers to interactively explore arbitrary frames in a time-varying dataset (an essential prerequisite for understanding protein dynamics, as shown in Figures 4.1, 4.15 and 4.16), where the exploration of a sequence of frames is vital to understand the whole docking process and the energies that intervene.

Finally, in contrast to Maestro and PLIP, our system supports a full interactive analysis. This was one of the advantages noted by the domain experts when evaluating our system. In our case, the outcome is not only based on a close contact analysis, but molecular interactions can also be filtered by strength or distance, or by energy type. Moreover, all visualizations are updated in real-time and the user can freely explore the 3D rendering at will, with real-time updates. Among other benefits, this enables for the first time the exploration of long-range interactions, as already noted. A second important feature is the flexibility we provide for interactively selecting views for data presentation.

4.5 Conclusions

This chapter presented visualization concepts developed for the analysis of binding forces in drug design and protein engineering. The proposed visual analysis workflow provides domain experts with several tools that let them perform a detailed analysis of the most relevant energies that intervene in a docking simulation: electrostatic energies, van der Waals energies, and solvent energies. This way, it becomes possible to gain an understanding of how simulations perform, why a ligand is getting or not getting to the docking position, and which are the residues crucial to such reactions. This chapter discussed the interaction computations used, outlined how to visualize the obtained results, and showed how filtering can help in the analysis process.

Furthermore, we showed examples where domain experts can see at a glance which are the dominant energies throughout the simulation, or how easy it is to determine the strongest interactions at a certain point of the path. Through the filtering, the system instantly highlights the important residues and the energies can be seen as color-coded to indicate their relevance. Thanks to the GPU-based energy calculation, which evaluates the binding equations in real-time, the application may be extended to any simulation path generated by other systems. The only necessary changes would be in the input files. In contrast to current approaches, our approach is able to deal with full simulation paths, instead of only single frames. Moreover, we may deal with information by residue, as well as per atom. This facilitates the analysis since most forces are exerted at residue level. Moreover, the application provides a set of widgets that further facilitate the inspection, such as user-defined colors and transparency, configurable clipping plane, high-quality illumination, and different rendering modes to add contextual information to the selected view.

5 Conclusions

The amount of data generated by Monte Carlo simulations, and the time and resources required to compute them, impose new requirements on the analysis tools used to retrieve information from their results. These tools should facilitate the analysis of whole trajectories, instead of focusing on single step analysis. Moreover, they should allow the inspection of intermediate results of unfinished simulations. Therefore, the techniques used by these tools must work in real-time without expensive preprocesses, presenting the results as soon as they are requested. Throughout the development of this thesis, we have proposed new techniques to visualize such results, improving the rendering quality, speed and the information presented, which work completely in real-time, complying with the aforementioned requirements.

Chapter 3 presents a set of techniques to improve the quality and rendering speed of different representation methods. Section 3.1 introduces a new method to compute ambient occlusion factors in molecules (represented by the Space-filling or the Balls &Sticks models) by using a hierarchical occupancy grid data structure. Besides improving the image quality, this technique also increases the molecular shape perception, allowing the easy identification of tunnels and cavities within the molecular structure. Moreover, this technique achieves real-time frame rates even for big molecules up to 1300 K atoms, providing a trade-off between rendering speed and image quality not found in previous methods. In Section 3.2, we describe a technique which is able to highlight, using the same data structure computed by the ambient occlusion algorithm, certain parts of the molecule via a well-known illustrative visualization technique: halos. With this technique, the user can easily follow the ligand or certain amino acids of the biomolecule through a simulation. Furthermore, this section proposes an extension to the basic halo algorithm which allows the positions visited by the selected molecule over a certain time range to be highlighted. Despite obtaining real-time frame rates for the examples tested, this technique has a high impact on the performance, since no rendering optimizations were applied. Nevertheless, by using proxy geometry or a smarter sampling scheme, we could easily increase the performance, making this technique available to hardware with low performance too.

The next section, Section 3.3, presents a technique to visualize secondary structures using ribbons. For each frame, this technique computes the geometry of the ribbons directly from the vertex data provided by the simulation software (or an external file) on the GPU and then, renders them. With this configuration, no precomputation is needed and the data transferred between CPU and GPU at every frame is minimized. Moreover, this technique selects, also in real-time, a different resolution for each segment of the backbone according to the distance to the camera, using lower resolutions for distant objects and a higher resolution for closer ones. This adaptive resolution allows the rendering of big molecules up to 750 K residues in real-time using the ribbons model, something not possible with previous methods.

The last section of Chapter 3, Section 3.4, describes a technique to compute and render solvent excluded surfaces (SES) interactively. One of the main limiting factors on the usage of these surfaces during the analysis of molecular simulations is the time required to compute them. Computing these surfaces for a whole trajectory could slow down the analysis process or even make it unpractical. The technique presented in this section, instead of computing the complete surface, computes an approximation of the SES in milliseconds and presents the result immediately to the user. Then, a refinement process starts in the background, where more accurate versions of the surface are computed until a fine enough resolution is achieved. Our technique, contrary to previous methods, assures interactivity during the analysis of whole simulations even for big molecules up to 545 K atoms. Moreover, this section presents a method to obtain smooth transitions between levels with different resolutions. Despite the interactivity of this technique, the resolution used could not be enough to compute all the internal features of the surface for big scenes. This section also proposes a method to retrieve these missing features in a refinement step carried out at the end of the process.

In order to demonstrate the significant improvement of the techniques presented in Chapter 3 compared with the available methods, we performed different comparison tests against state-of-the-art methods and the results were presented in their respective sections. These results showed that all the techniques presented improved the rendering quality or the computation speed, or both.

The last contribution of this thesis is presented in Chapter 4: a tool to visualize the forces driving the interaction between a biomolecule and a ligand. This tool, using different energy models to represent the interactions between atoms, calculates in real-time the energy between every pair of atoms in a simulation step. These single interactions are then grouped and shown to the user using well-known visualization techniques. The number of interacting groups can be filtered to avoid cluttered images and to show only the relevant information. The groups are presented to the user encoding the energy of their interaction by a set of colors and motifs. Context of the whole system is provided by using a clipping plane and transparencies to visualize the rest of the molecule. Furthermore, a 2D abstract view, linked to the 3D main view, allows easy identification of the selected atom groups, displaying extra information about the groups and their interaction. Moreover, the end of this chapter presents the results of some use cases, in which the utility of the tool was demonstrated, together with a comparison with the existing software, which highlights several deficiencies that were fulfilled by our proposal.

In conclusion, the techniques presented in Chapter 3 improved the efficiency and quality of the most commonly used techniques to visualize molecular simulations. These improvements now allow the use of the aforementioned methods whilst inspecting real-time simulations of big molecular scenes, which was impossible for most techniques until now. At the same time, they provide high-quality visualization, which not only helps in presenting the results but also helps to identify different features of the molecule. Chapter 4 presents the last contribution of this thesis, the design of a new tool which analyzes the underlying forces driving the simulations, an aspect which did not receive much attention until now and provides new insight into the behavior of such molecules.

5.1 Context of the thesis

This thesis was developed in close collaboration with the *Electronic And Atomic Protein Modeling* group of the *Barcelona Supercomputing Center* (BSC). This group developed a Monte Carlo simulation software called *Protein energy landscape exploration* (PELE) [MSG13, BVAG05], which, using protein structure prediction algorithms, is able to simulate the interaction between ligands and biomolecules much faster than previous methods. The researchers of the group use PELE in their daily work to simulate the interactions between different ligands and proteins in a workflow that sometimes can be tedious. The contributions of this thesis resulted directly from real needs of these researchers, who gave us feedback during all the design process. This collaboration helped us to find real problems to solve and to obtain feedback from the final users of our techniques, Additionally, it will help the researchers in their daily work, as they are tasked with integrating the software developed by the author during these years within their simulation software.

5.2 Future research

Molecular visualization is a growing area with many open research problems, some of which were addressed in the development of this thesis. Ambient occlusion is a powerful tool to enhance the perception of the molecular shape, but existing algorithms only focus on generating occlusions at a certain scale. Our work (presented in Section 3.1) provides a method to generate occlusions at the atomic and molecular level simultaneously. Despite the quality achieved, when we attempting to visualize molecular aggregations, such as whole cells, a more sophisticated technique should be used to take into account the occlusions generated between the vast number of molecules in the system, making this problem a possible future research direction.

Another interesting line of research is the adaptation of the algorithm proposed in Section 3.3 to other areas in visualization and computer graphics. Since our method is able to adaptively generate the geometry to visualize a curve, the same principles can be applied to enhance the rendering of other elements with a similar geometry, such as hair, brain fibers, enhanced lines and so on.

The computation of solvent excluded surfaces has attracted much attention, since this surface is a highly useful tool in the analysis of molecular shapes. However, its computation still is an expensive process. Much research has been conducted related to its computation. In this thesis, we presented a method to compute this surface progressively for cases in which the size of the molecule does not allow its computation in real-time. Nevertheless, its real-time computation remains an open problem when dealing with molecules with a very large number of atoms.

Although all the presented topics appear to be promising research directions, the visualization of the interacting forces between molecules is the topic which raises more challenges. The tool presented in this thesis visualizes the interactions between the ligand as a whole and the atoms of the molecule gathered into groups. To provide a more detailed inspection, the interactions could be divided and shown by atom pairs. Moreover, the area around the ligand could be divided into regions and the energy of each one could be visualized, indicating which parts of the ligand are affected by certain energies. Another important topic for future research is the use of a clustering method based on these computed energies (or other criteria) to improve the inspection of the results generated by the simulation software.

Besides the research directions described, we are also planning to study the workflow followed by chemists of different expertise to identify steps in which a smart and innovative visualization method could improve their daily work, allowing for more efficient research.

5.3 Publications

The resulting algorithms and tools developed in this thesis led to the following publications:

- P. Hermosilla and V. Guallar and A. Vinacua and P.P. Vázquez. High quality illustrative effects for molecular rendering, *Computers & Graphics*, 54:113-120, 2016.
- P. Hermosilla and V. Guallar and P.P. Vázquez and A. Vinacua. Adaptive on-the-fly molecular ribbons generation, *Poster EuroVis*, 2015.
- P. Hermosilla and V. Guallar and P.P. Vázquez and A. Vinacua. Instant Visualization of Secondary Structures of Molecular Models, *In Proc. of Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 51-60, 2015.
- P. Hermosilla and J. Estrada and V. Guallar and T. Ropinski and À. Vinacua and P. P. Vázquez, Physics-Based Visual Characterization of Molecular Interaction Forces, *IEEE Transactions on Visualization and Computer Graphics*, 23(1):731-740, 2017.
- P. Hermosilla and M. Krone and V. Guallar and P.P. Vázquez and À. Vinacua and T. Ropinski, Interactive GPU-based generation of solvent-excluded surfaces, *The Visual Computer*, 33(6): 869-881, 2017.

Bibliography

[AFFM$^+$16] Sandra Acebes, Elena Fernandez-Fueyo, Emanuele Monza, M Fatima Lucas, David Almendral, Francisco J Ruiz-Dueñas, Henrik Lund, Angel T Martinez, and Victor Guallar. Rational enzyme engineering through biophysical and biochemical modeling. *ACS Catalysis*, 6(3):1624–1629, 2016.

[BBC$^+$05] Jay L Banks, Hege S Beard, Yixiang Cao, Art E Cho, Wolfgang Damm, Ramy Farid, Anthony K Felts, Thomas A Halgren, Daniel T Mainz, Jon R Maple, et al. Integrated modeling program, applied chemical theory (impact). *Journal of computational chemistry*, 26(16):1752–1780, 2005.

[BDST04] C. Bajaj, P. Djeu, V. Siddavanahalli, and A. Thane. Texmol: interactive visual exploration of large flexible multi-component molecular complexes. In *Visualization, 2004. IEEE*, pages 243–250, Oct 2004.

[BG07] S. Bruckner and E. Gröller. Enhancing depth-perception with flexible volumetric halos. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1344–1351, Nov 2007.

[BGB$^+$08] Katrin Bidmon, Sebastian Grottel, Fabian Bös, Jürgen Pleiss, and Thomas Ertl. Visual abstractions of solvent pathlines near protein cavities. *Computer Graphics Forum*, 27(3):935 – 942, 2008.

[BLM96] M. J. Bentum, B. B. A. Lichtenbelt, and T. Malzbender. Frequency analysis of gradient estimators in volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 2(3):242–254, Sep 1996.

[BSC15] Jens Behley, Volker Steinhage, and Armin B Cremers. Efficient radius neighbor search in three-dimensional point clouds. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015. to appear.

[BSD08] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. Image-space horizon-based ambient occlusion. In *ACM SIGGRAPH 2008 Talks*, SIGGRAPH ’08, pages 22:1–22:1, New York, NY, USA, 2008. ACM.

[BSN12] Pranav D Bagur, Nithin Shivashankar, and Vijay Natarajan. Improved quadric surface impostors for large bio-molecular visualization. In *Proceedings of the Eighth Indian Conference on Computer Vision, Graphics and Image Processing*, page 33. ACM, 2012.

[Bun05] M. Bunnell. Dynamic ambient occlusion and indirect lighting. *GPU Gems2*, pages 223–233, 2005.

[BVAG05] Kenneth W. Borrelli, Andreas Vitalis, Raul Alcantara, and Victor Guallar. Pele: Protein energy landscape exploration. a novel monte carlo based technique. *Journal of Chemical Theory and Computation*, 1(6):1304–1311, 2005.

[BWF$^+$00] Helen M Berman, John Westbrook, Zukang Feng, Gary Gilliland, T N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[Cab13] Ségolène Caboche. Leview: automatic and interactive generation of 2d diagrams for biomacromolecule/ligand interactions. *Journal of cheminformatics*, 5:40, 2013.

[Car91] Mike Carson. Ribbons 2.0. *Journal of Applied Crystallography*, 24(5):958–961, 1991.

[CB86] Mike Carson and Charles E Bugg. Algorithm for ribbon models of proteins. *Journal of Molecular Graphics*, 4(2):121–122, 1986.

[CCW06] Tolga Can, Chao-I Chen, and Yuan-Fang Wang. Efficient molecular surface generation using level-set methods. *Journal of Molecular Graphics and Modelling*, 25(4):442 – 454, 2006.

[CG07] Gregory Cipriano and Michael Gleicher. Molecular surface abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1608–1615, November 2007.

[CG12] Cyril Crassin and Simon Green. Octree-based sparse voxelization using the gpu hardware rasterizer. In Patrick Cozzi and Christophe Riccio, editors, *OpenGL Insights*, pages 303–318. CRC Press, 2012.

[CNS$^+$11] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum (Proceedings of Pacific Graphics 2011)*, 30, sep 2011.

[Con83] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16(5):548–558, Oct 1983.

[CP53] Robert B. Corey and Linus Pauling. Molecular models of amino acids, peptides, and proteins. *Review of Scientific Instruments*, 24(8):621–627, 1953.

[Dal10] John Dalton. *A New System of Chemical Philosophy*, volume 2 of *Cambridge Library Collection - Physical Sciences*. Cambridge University Press, 2010.

[DDG$^+$12] Ron O Dror, Robert M Dirks, JP Grossman, Huafeng Xu, and David E Shaw. Biomolecular simulation: a computational microscope for molecular biology. *Annual review of biophysics*, 41:429–452, 2012.

[Ede99] Herbert Edelsbrunner. Deformable Smooth Surface Design. *Discrete and Computational Geometry*, 21:87–115, 1999.

[EHB$^+$15] Karl Edman, Ali Hosseini, Magnus K Bjursell, Anna Aagaard, Lisa Wissler, Anders Gunnarsson, Tim Kaminski, Christian Köhler, Stefan Bäckström, Tina J Jensen, et al. Ligand binding mechanism in steroid receptors: From conserved plasticity to differential evolutionary constraints. *Structure*, 23(12):2280–2290, 2015.

[EM94] Herbert Edelsbrunner and Ernst P. Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, 1994.

[FJB$^+$17] Katarína Furmanová, Miroslava Jarešová, Jan Byška, Adam Jurčík, Július Parulek, Helwig Hauser, and Barbora Kozlíková. Interactive exploration of ligand transportation through protein tunnels. *BMC Bioinformatics*, 18(2):22, 2017.

[FKE13] Martin Falk, Michael Krone, and Thomas Ertl. Atomistic visualization of mesoscopic whole-cell simulations using ray-casted instancing. *Computer Graphics Forum*, pages 195–206, 2013.

[FKRE09] Martin Falk, Michael Klann, Matthias Reuss, and Thomas Ertl. Visualization of signal transduction processes in the crowded environment of the cell. In *Proceedings of the 2009 IEEE Pacific Visualization Symposium*, pages 169–176, Washington, DC, USA, 2009. IEEE Computer Society.

[FM08] Dominic Filion and Rob McNaughton. Starcraft 2, effects and techniques. Advancesin Real-Time Rendering in 3D Graphics and Games Course - SIGGRAPH 2008, 2008.

[GB78] J Greer and B L Bush. Macromolecular shape and surface maps by solvent exclusion. *Proceedings of the National Academy of Sciences*, 75(1):303–307, 1978.

[GBCG$^+$14] David Gunther, Roberto A Boto, Juila Contreras-Garcia, Jean-Philip Piquemal, and Julien Tierny. Characterizing molecular interactions in chemical systems. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2476–2485, 2014.

[GBM$^+$12] Sebastian Grottel, Philipp Beck, Christoph Muller, Guido Reina, Johannes Roth, Hans-Rainer Trebin, and Thomas Ertl. Visualization of electrostatic dipoles in molecular dynamics of metal oxides. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2061–2068, December 2012.

[GG01] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering.* A.K. Peters, 2001.

[GKM$^+$15] S. Grottel, M. Krone, C. Muller, G. Reina, and T. Ertl. Megamol – a prototyping framework for particle-based visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 21(2):201–214, Feb 2015.

[GKSE12] S. Grottel, M. Krone, K. Scharnowski, and T. Ertl. Object-space ambient occlusion for molecular dynamics. In *Pacific Visualization Symposium (PacificVis), 2012 IEEE*, pages 209–216, Feb 2012.

[GO92] David S Goodsell and Arthur J Olson. Molecular illustration in black and white. *Journal of molecular graphics*, 10(4):235–240, 1992.

[Goo03] DS Goodsell. Illustrating molecules. *The Guild Handbook of Scientific Illustration, Hodges ERS,(Ed.)*, 2:267–270, 2003.

[GRDE10] Sebastian Grottel, Guido Reina, Carsten Dachsbacher, and Thomas Ertl. Coherent Culling and Shading for Large Molecular Dynamics Visualization. *Computer Graphics Forum*, 29(3):953–962, 2010.

[Gre07] S. Green. White paper: Cuda particles. Technical report, 2007.

[GS01] Jason D Gans and David Shalloway. Qmol: a program for molecular visualization on windows-based {PCs}. *Journal of Molecular Graphics and Modelling*, 19(6):557 – 559, 2001.

[Gum03] Stefan Gumhold. Splatting illuminated ellipsoids with depth correction. In Thomas Ertl, editor, *VMV*, pages 245–252. Aka GmbH, 2003.

[GW06] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition).* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[GWCD15] Anthony J.F. Griffiths, Susan R. Wessler, Sean B. Carroll, and John Doebley. *An Introduction to Genetic Analysis.* WH Freeman, 11 edition, 2015.

[HDS96] William Humphrey, Andrew Dalke, and Klaus Schulten. Vmd: Visual molecular dynamics. *Journal of Molecular Graphics*, 14(1):33–38, February 1996.

[HEG$^+$17] P. Hermosilla, J. Estrada, V. Guallar, T. Ropinski, À. Vinacua, and P. P. Vázquez. Physics-based visual characterization of molecular interaction forces. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):731–740, Jan 2017.

[HGVV15a] Pedro Hermosilla, Víctor Guallar, Àlvar Vinacua, and Pere-Pau Vázquez. Instant visualization of secondary structures of molecular models. In *In Proc. of Eurographics Workshop on Visual Computing for Biology and Medicine*, pages 51–60, 2015.

[HGVV15b] Pedro Hermosilla, Víctor Guallar, Pere-Pau Vázquez, and Àlvar Vinacua. Adaptive on-the-fly molecular ribbons generation. In *Poster EuroVis*, pages 1–3, 2015.

[HGVV16] P. Hermosilla, V. Guallar, A. Vinacua, and P.P. Vázquez. High quality illustrative effects for molecular rendering. *Computers & Graphics*, 54:113 – 120, 2016. Special Issue on CAD/Graphics 2015.

[HH11] Takahiro Harada and Lee Howes. Introduction to GPU radix sort. http://www.heterogeneouscompute.org/wordpress/wp-content/uploads/2011/06/RadixSort.pdf, 2011. Online; accessed 2016-03-29.

[HKG$^+$17] Pedro Hermosilla, Michael Krone, Victor Guallar, Pere-Pau Vázquez, Àlvar Vinacua, and Timo Ropinski. Interactive gpu-based generation of solvent-excluded surfaces. *The Visual Computer*, 33(6):869–881, 2017.

[Hoe14] R. C. Hoetzlein. Fast fixed-radius nearest neighbors: Interactive million-particle fluids. In *GPU Technology Conference*, 2014.

[Hof65] A Hoffmann. On the combining power of atoms. *Proceedings of the Royal Institution*, 4:401–430, 1865.

[HOF04] A. Halm, L. Offen, and D. Fellner. Visualization of complex molecular ribbon structures at interactive rates. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, pages 737–744, July 2004.

[HOF05] Andreas Halm, Lars Offen, and Dieter W. Fellner. Biobrowser: A framework for fast protein visualization. In Ken Brodlie, David J. Duke, and Kenneth I. Joy, editors, *EuroVis*, pages 287–294. Eurographics Association, 2005.

[HPC96] Michael E Hodsdon, Jay W Ponder, and David P Cistola. The NMR solution structure of intestinal fatty acid-binding protein complexed with palmitate: application of a novel distance geometry algorithm. *Journal of molecular biology*, 264(3):585–602, 1996.

[HSS$^+$05] Markus Hadwiger, Christian Sigg, Henning Scharsach, Khatja Bühler, and Markus Gross. Real-time ray-casting and advanced shading of discrete isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[HV09] P. Hermosilla and P.P. Vázquez. Single pass gpu stylized edges. In *Proceedings of IV Iberoamerican Symposium in Computer Graphics (SIACG 2009)*, 2009.

[HV10] P. Hermosilla and P.P. Vázquez. Npr techniques using the geometry shader. *GPU Pro*, pages 149–166, 2010.

[JD08] Pere-Pau Vázquez José Díaz, Héctor Yela. Vicinity occlusion maps: Enhanced depth perception of volumetric models. In *Computer Graphics International (CGI)*, pages 56–63, 2008.

[Jol14] Ian Jolliffe. *Principal Component Analysis.* John Wiley & Sons, Ltd, 2014.

[K.52] Linderstrøm-Lang K. Proteins and enzymes. *Lane Medical Lectures*, VI, 1952.

[Kaj86] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, August 1986.

[Kaj09] V. Kajalin. Screen-space ambient occlusion. *ShaderX7*, pages 413–424, 2009.

[KBE08] Michael Krone, Katrin Bidmon, and Thomas Ertl. GPU-based Visualisation of Protein Secondary Structure. In Ik Soo Lim and Wen Tang, editors, *Theory and Practice of Computer Graphics.* The Eurographics Association, 2008.

[KBE09] M. Krone, K. Bidmon, and T. Ertl. Interactive visualization of molecular surface dynamics. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1391–1398, Nov 2009.

[KBW96] Reto Koradi, Martin Billeter, and Kurt Wüthrich. Molmol: A program for display and analysis of macromolecular structures. *Journal of Molecular Graphics*, 14(1):51 – 55, 1996.

[KC13] Nickolay A Khazanov and Heather A Carlson. Exploring the composition of protein-ligand binding sites on a large scale. *PLoS Computational Biology*, 9(11):e1003321, 2013.

[KDE10] Michael Krone, Carsten Dachsbacher, and Thomas Ertl. Parallel computation and interactive visualization of time-varying solvent excluded surfaces, 2010.

[Kek66] Aug. Kekuié. Untersuchungen über aromatische verbindungen ueber die constitution der aromatischen verbindungen. i. ueber die constitution der aromatischen verbindungen. *Justus Liebigs Annalen der Chemie*, 137(2):129–196, 1866.

[KGE11] M. Krone, S. Grottel, and T. Ertl. Parallel contour-buildup algorithm for the molecular surface. In *Biological Data Visualization (BioVis), 2011 IEEE Symposium on*, pages 17–22, Oct 2011.

[KKL$^+$16] Barbora Kozlíková, Michael Krone, Norbert Lindow, Martin Falk, Marc Baaden, Daniel Baum, Ivan Viola, Julius Parulek, and

Hans-Christian Hege. Visualization of molecular structure: State of the art revisited. *Computer Graphics Forum*, 2016. (to appear).

[Kol65] L. W. Koltun. Space filling atomic units and connectors for molecular models, February 23 1965. US Patent 3,170,246.

[Lan02] Hayden Landis. Production-ready global illumination, 2002.

[LBH12] Norbert Lindow, Daniel Baum, and Hans-Christian Hege. Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale. *Computer Graphics Forum*, 31(3pt4):1325–1334, 2012.

[LBH14] N. Lindow, D. Baum, and H.-C. Hege. Ligand excluded surface: A new type of molecular surface. *Visualization and Computer Graphics, IEEE Transactions on*, 20(12):2486–2495, Dec 2014.

[LBPcH10] Norbert Lindow, Daniel Baum, Steffen Prohaska, and Hans christian Hege. Accelerated visualization of dynamic molecular surfaces, 2010.

[LC87] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM SIGGRAPH Computer Graphics and Interactive Techniques*, volume 21, pages 163–169, 1987.

[LKEP14] Kai Lawonn, Michael Krone, Thomas Ertl, and Bernhard Preim. Line integral convolution for real-time illustration of molecular surface shape and salient regions. *Computer Graphics Forum*, 33(3):181–190, 2014.

[LMPSV14] Mathieu Le Muzic, Julius Parulek, Anne Kristin Stavrum, and Ivan Viola. Illustrative visualization of molecular reactions using omniscient intelligence and passive agents. *Computer Graphics Forum*, 33(3):141–150, 2014.

[LR71] B. Lee and F.M. Richards. The interpretation of protein structures: Estimation of static accessibility. *Journal of Molecular Biology*, 55(3):379 – IN4, 1971.

[LS11] Roman A Laskowski and Mark B Swindells. Ligplot+: multiple ligand–protein interaction diagrams for drug discovery. *Journal of chemical information and modeling*, 51(10):2778–2786, 2011.

[McG10] Morgan McGuire. Ambient occlusion volumes. In *Proceedings of High Performance Graphics 2010*, June 2010.

[MGB$^+$05] John Moreland, Apostol Gramada, Oleksandr Buzko, Qing Zhang, and Philip Bourne. The molecular biology toolkit (mbt): a modular platform for developing molecular visualization applications. *BMC Bioinformatics*, 6(1):21, 2005.

[MH04] Morgan McGuire and John F. Hughes. Hardware-determined feature edges. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 35–147. ACM Press, 2004.

[MI08] Koichi Momma and Fujio Izumi. VESTA: a three-dimensional visualization system for electronic and structural analysis. *Journal of Applied Crystallography*, 41(3):653–658, 2008.

[MI11] Koichi Momma and Fujio Izumi. VESTA 3 for three-dimensional visualization of crystal, volumetric and morphology data. *Journal of Applied Crystallography*, 44(6):1272–1276, 2011.

[Mit07] M. Mittring. Finding next gen-cryengine 2. In SIGGRAPH '07: ACM SIGGRAPH 2007 courses, pages 97-121, 2007.

[MOBH11] Morgan McGuire, Brian Osman, Michael Bukowski, and Padraic Hennessy. The alchemy screen-space ambient obscurance algorithm. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, pages 25–32, New York, NY, USA, 2011. ACM.

[MSG13] Armin Madadkar-Sobhani and Victor Guallar. Pele web server: atomistic study of biomolecular systems at your fingertips. *Nucleic Acids Research*, 41(W1):W322–W328, 2013.

[OBC04] Alexey Onufriev, Donald Bashford, and David A Case. Exploring protein native states and large-scale conformational changes with a modified generalized born model. *Proteins: Structure, Function, and Bioinformatics*, 55(2):383–394, 2004.

[Pau45] L. Pauling. The nature of the chemical bond. 1945.

[Per05] James A. Perkins. A history of molecular representation part one: 1800 to the 1960s. *Journal of Biocommunication*, 31(1), 2005.

[PG04] M. Pharr and S. Green. Ambient occlusion. *GPU Gems*, pages 667–692, 2004.

[PGH$^+$04] Eric F. Pettersen, Thomas D. Goddard, Conrad C. Huang, Gregory S. Couch, Daniel M. Greenblatt, Elaine C. Meng, and Thomas E. Ferrin. Ucsf chimera-a visualization system for exploratory research and analysis. *Journal of Computational Chemistry*, 25(13):1605–1612, 2004.

[PHWF01] Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 581–, New York, NY, USA, 2001. ACM.

[PMP10] G. Papaioannou, M. L. Menexi, and C. Papadopoulos. Real-time volume-based ambient occlusion. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):752–762, Sept 2010.

[PV12] Julius Parulek and Ivan Viola. Implicit representation of molecular surfaces. In *IEEE Pacific Visualization Symposium*, pages 217–224, 2012.

[PVV02] Alessandro Pedretti, Luigi Villa, and Giulio Vistoli. Vega: a versatile program to convert, handle and visualize molecular structure on windows-based {PCs}. *Journal of Molecular Graphics and Modelling*, 21(1):47 – 49, 2002.

[Ric77] F. M. Richards. Areas, Volumes, Packing, and Protein Structure. *Annual Review of Biophysics and Bioengineering*, 6(1):151–176, 1977.

[Ric81] Jane S. Richardson. The anatomy and taxonomy of protein structure. In John T. Edsall C.B. Anfinsen and Frederic M. Richards, editors, *Advances in Protein Chemistry*, volume 34 of *Advances in Protein Chemistry*, pages 167 – 339. Academic Press, 1981.

[RK03] Fernando Randima and Mark J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, 2003.

[RUC$^+$13] Jane B Reece, Lisa A Urry, Michael L Cain, Steven A Wasserman, Peter V Minorsky, and Robert B Jackson. *Campbell biology*. Pearson Higher Ed, 10 edition, 2013.

[SA07] Perumaal Shanmugam and Okan Arikan. Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 73–80, New York, NY, USA, 2007. ACM.

[SAMG14] Alper Sarikaya, Danielle Albers, Julie Mitchell, and Michael Gleicher. Visualizing validation of protein surface classifiers. *Computer Graphics Forum*, 33(3):171–180, 2014.

[SBK98] Michael Schaefer, Christian Bartels, and Martin Karplus. Solution conformations and thermodynamics of structured peptides: molecular dynamics simulation with an implicit solvation model. *Journal of molecular biology*, 284(3):835–848, 1998.

[Sch16] LLC Schrödinger. Schrödinger release 2016-1: Maestro version 10.5. http://gts.sourceforge.net/, 2016.

[SEJ+05] Rajendra Kumar Singh, AS Ethayathulla, Talat Jabeen, Sujata Sharma, Punit Kaur, and Tej P Singh. Aspirin induces its anti-inflammatory effects through its specific binding to phospholipase a2: Crystal structure of the complex formed between phospholipase a2 and aspirin at 1.9 å resolution. *Journal of drug targeting*, 13(2):113–119, 2005.

[SGG15] Joachim Staib, Sebastian Grottel, and Stefan Gumhold. Visualization of particle-based data with transparency and ambient occlusion. *Computer Graphics Forum*, 34(3):151–160, 2015.

[SHL+12] Nadine Schneider, Sally Hindle, Gudrun Lange, Robert Klein, Jürgen Albrecht, Hans Briem, Kristin Beyer, Holger Claußen, Marcus Gastreich, Christian Lemmen, et al. Substantial improvements in large-scale redocking and screening using the novel hyde scoring function. *Journal of computer-aided molecular design*, 26(6):701–723, 2012.

[SKR+14] K. Scharnowski, M. Krone, G. Reina, T. Kulschewski, J. Pleiss, and T. Ertl. Comparative visualization of molecular surfaces using deformable models. *Computer Graphics Forum*, 33(3):191–200, 2014.

[SMR06] Katrin Stierand, Patrick C Maaß, and Matthias Rarey. Molecular complexes at a glance: automated generation of two-dimensional complex diagrams. *Bioinformatics*, 22(14):1710–1716, 2006.

[SOS96] M. F. Sanner, A. J. Olson, and J. C. Spehner. Reduced surface: an efficient way to compute molecular surfaces. *Biopolymers*, 38(3):305–320, Mar 1996.

[SSH$^+$15] Sebastian Salentin, Sven Schreiber, V Joachim Haupt, Melissa F Adasme, and Michael Schroeder. Plip: fully automated protein–ligand interaction profiler. *Nucleic acids research*, 43(W1):W443–W447, 2015.

[SSZK04] Mirko Sattler, Ralf Sarlette, Gabriel Zachmann, and Reinhard Klein. Hardware-accelerated ambient occlusion computation. In *Proceedings of the Vision, Modeling, and Visualization Conference 2004 (VMV 2004), Stanford, California, USA, November 16-18, 2004*, pages 331–338, 2004.

[ST90] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '90, pages 197–206, New York, NY, USA, 1990. ACM.

[Sun16] Marc Sunet. *Ambient Occlusion on Mobile: An Empirical Comparison.* Universitat Politècnica de Catalunya, 2016.

[SVGR16] R. Skånberg, P. Vázquez, V. Guallar, and T. Ropinski. Real-time molecular visualization supporting diffuse interreflections and ambient occlusion. *IEEE transactions on visualization and computer graphics*, 22(1):718–727, 01 2016.

[SWBG06] Christian Sigg, Tim Weyrich, Mario Botsch, and Markus Gross. Gpu-based ray-casting of quadratic surfaces. In *Proceedings of the 3rd Eurographics / IEEE VGTC Conference on Point-Based Graphics*, SPBG'06, pages 59–65, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[TA96] M. Totrov and R. Abagyan. The contour-buildup algorithm to calculate the analytical molecular surface. *J. Struct. Biol.*, 116(1):138–143, 1996.

[TCM06] Marco Tarini, Paolo Cignoni, and Claudio Montani. Ambient occlusion and edge cueing for enhancing real time molecular visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1237–1244, September 2006.

[VBJ$^+$94] Amitabh Varshney, Frederick P. Brooks, Jr., Jr. William, and William V. Wright. Linearly scalable computation of smooth molecular surfaces. In *In IEEE Computer Graphics and Applications Vol 14*, page pp., 1994.

[VGB$^+$05] Ivan Viola, Meister Eduard Gr"oller, Katja B"uhler, Markus Hadwiger, Bernhard Preim, David Ebert, Mario Costa Sousa, and Don Stredney. Ieee visualization tutorial on illustrative visualization, 2005.

[VV10] D. Voet and J.G. Voet. *Biochemistry, 4th Edition*. John Wiley & Sons, 2010.

[WB11] Manuel Wahle and Stefan Birmanns. Gpu-accelerated visualization of protein dynamics in ribbon mode. In *Visualization and Data Analysis*, volume 7868, 2011.

[WC53] J. D. Watson and F. H. C. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[Web09] Joseph R Weber. Proteinshader: illustrative rendering of macromolecules. *BMC structural biology*, 9(1):19, 2009.

[WL05] Gerhard Wolber and Thierry Langer. Ligandscout: 3-d pharmacophores derived from protein-bound ligands and their use as virtual screening filters. *Journal of chemical information and modeling*, 45(1):160–169, 2005.

[WLL$^+$99] J.Michael Word, Simon C. Lovell, Thomas H. LaBean, Hope C. Taylor, Michael E. Zalis, Brent K. Presley, Jane S. Richardson, and David C. Richardson. Visualizing and quantifying molecular goodness-of-fit: small-probe contact dots with explicit hydrogen atoms. *Journal of Molecular Biology*, 285(4):1711–1733, 1999.

[XZ09] Dong Xu and Yang Zhang. Generating triangulated macromolecular surfaces by euclidean distance transform. *PLOS ONE*, 4(12):e8140, 2009.

[Yu09] Zeyun Yu. A list-based method for fast generation of molecular surfaces. In *Int. Conf. of the IEEE Engineering in Medicine and Biology Society*, volume 31, pages 5909–5912, 2009.

[ZIK98] S. Zhukov, A. Iones, and G. Kronin. An ambient light illumination model. In George Drettakis and Nelson Max, editors, *Rendering Techniques '98*, Eurographics, pages 45–55. Springer Vienna, 1998.

[ZSK09] Matus Zamborsky, Tibor Szabo, and Barbora Kozlikova. Dynamic visualization of protein secondary structures. In *Proceedings of the 13th Central European Seminar on Computer Graphics (CESCG)*, pages 147–152, 2009.

Molecular dynamics simulations are computer simulations of the physical movements of atoms and molecules, and the interactions between them. In the particular cases we focus on (pharmaceutical drug design and enzymatic catalysis), molecular dynamics simulations predict the binding mode and binding affinity of a small molecule (the drug) with a biomolecule. Providing new tools and techniques to visualize and to interact with these simulations is crucial in understanding them. Throughout the development of this thesis, we have proposed new techniques to improve both the rendering quality and speed of different molecular representation models. Moreover, we developed a novel system to analyze docking simulations and the underlying interactions between the atom groups.