Masaryk University
Faculty of Informatics

Accelerating Visualization of Dense Molecular Scenes through Adaptive Occlusion Estimation

Master's Thesis

Matúš Talčík

Brno, Fall 2020

Masaryk University
Faculty of Informatics

Accelerating Visualization of Dense Molecular Scenes through Adaptive Occlusion Estimation

Master's Thesis

Matúš Talčík

Brno, Fall 2020

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Matúš Talčík

Advisor: doc. RNDr. Barbora Kozlíková, Ph.D.

Acknowledgements

I want to thank my supervisor, doc. Barbora Kozlíková, for her perseverance, while reviewing endless iterations of this thesis, and my advisor, Dr. Peter Mindek, for helping me polishing figures to a visually pleasing form.

Abstract

Visual communication of complex phenomena taking place in living organisms is substantial in many areas, namely in education, interdisciplinary exchange of knowledge, or dissemination of scientific findings to a general audience. Communicating molecular processes to a lay audience requires a thorough depiction of the context, which consists of molecular landscapes of immense size. Depicting them in real-time as means for realistic and engaging interactive visualization poses significant challenges to the rendering performance. However, achieving interactive framerates for complex scenes, operating with structures on all scales spanning from atoms up to almost entire cells, still cannot be reached by the existing algorithms. Therefore, we present a novel approach that significantly reduces the GPU workload for large and dense molecular scenes. It allows us to scale up the interactive visualization of molecular scenes consisting of 10^{11} atoms, which is on the scale of an entire bacterial cell. As the models for such large structures are still not available up to their atomistic details, we tested the performance of our approach on synthetic models. Compared to the current state-of-the-art approaches, we achieved the 3x to 7x speedup.

Keywords

rasterization, rendering, visibility, occlusion, molecule

Contents

1 Introduction **1**

2 Related Work **5**
2.1 *Algorithms for Rendering Complex Scenes* 5
2.1.1 Occlusion Culling 5
2.1.2 Billboards 7
2.2 *Modeling and Storytelling* 11
2.3 *Illustrative Rendering of Molecules* 12

3 Algorithm **15**
3.1 *Initial Requirements* 15
3.2 *Overview* 16
3.3 *Stage 1 – Preprocessing* 18
3.4 *Stage 2 – Visibility Computation* 21
3.5 *Stage 3 – Retrieval of Cached Visibility* 25
3.6 *Stage 4 – Final Rendering* 26

4 Results and Discussion **27**

5 Conclusion and Future Work **35**

Bibliography **37**

A List of Electronic Attachments **45**

List of Figures

1.1 Two complex scenes that are forming the main target of our proposed algorithm. Showcased scenes consist of 1000 structures that together have around 14 million molecules and 90 billion atoms. Rendered by our proposed approach, the scenes run in real-time at between 10 and 30 frames per second. *3*

2.1 Example demonstrating the difficulty with automatically generated occluders for molecular data. The occupation of the viewpoint by the molecule (in grey) should be overlapped by occluders (yellow) as much as possible and ideally with a small amount of triangles. The existing automatic solutions are producing many occluders of poor quality of coverage. *8*

3.1 The stages of our method: 1) The preprocessing stage, which is executed during the data loading, sorts molecules of biological structures according to the Hilbert curve mapped onto a sphere. Continuous sorted patches are shown with different colors (some are omitted for clarity). Note that they may overlap as different types of molecules are sorted separately. 2) Visibility computation renders the depth map of a structure from a single viewpoint twice to retrieve visible molecules, which are then cached with our efficient compression based on the Hilbert curve and gap-filling. This stage is performed on-demand in real-time. 3) Retrieval stage retrieves the cached occlusion. 4) Last stage renders the structure. *17*

3.2 Depiction of a single camera viewpoint for dense biological structure. Visible molecules are in blue, occluded ones in gray. Only few molecules, mostly those forming a part of the shell, are visible. *19*

3.3 Mapping of molecules of a biological structure on a unit sphere. The unit sphere is black, molecules are blue. Mapped positions are depicted in green. *20*

3.4 Depiction of visible molecules that can be stored as ranges. Different colors show separate ranges on the Hilbert curve. *20*

3.5 Depiction of how some occluded molecules cause disruption in the Hilbert curve ordering of visible molecules. Grey molecules are occluded in the depicted viewpoint. Yellow and green are visible disjoint clusters in memory. Red molecules form occluded clusters that cause disruption in the Hilbert curve. *22*

3.6 Memory layout of molecules of a biological structure and the associated visibility information. Data associated with each molecule are stored linearly, sorted according to the Hilbert curve. The information which molecules are visible is stored efficiently as ranges of indices pointing to this array. *23*

3.7 Camera viewpoints and snapping of users' camera to the closest viewpoint. Visible molecules from a snapped viewpoint are shown in blue. Yellow is the actual viewpoint that is being snapped to the closest one. *25*

3.8 Post-process effects that are applied to our renders. *26*

4.1 Detail of the benchmark scene containing 1000 models of SARS-CoV-2 [46]. The scene consists of 249, 162, 000 molecules, i.e., 24, 022, 383, 000 atoms in total. *27*

4.2 Percentage of visible molecules at different range limits for our two benchmarking datasets. *29*

4.3 Performance test for camera zoomed-in to a single structure that is covering the entire screen, as depicted in Figure 4.6. *30*

4.4 Benchmark of a synthetic scene with caching the visibility information. Timings across the running time of the application are in blue. The baseline without using our algorithm is in red. The camera is positioned in the middle of the scene. *31*

4.6 A synthetic benchmark scene. Top: Fully zoomed-out view to encompass the whole scene. Middle: Zoomed-out in middle of the scene. Bottom: Single structure. *32*

4.5 Benchmark of a synthetic scene with and without using our algorithm. Timings are across the whole scene, from a completely zoomed-out view capturing the whole scene to zoomed onto a single structure in the middle of the scene. Timings were performed after completing the computation of all visibility information. *33*

1 Introduction

Molecular visualization has been traditionally used in biology and biochemistry to understand and analyze large molecular structures and interactions between them. Historically, the molecular models were assembled manually, and with the advent of computer graphics, first virtual models of individual molecules started to emerge. Nanoscale structures, such as protein molecules consisting of hundreds or thousands of atoms, could not be studied in detail without creating their representative 3D models, as the imaging techniques back then did not offer sufficient resolution.

Even with today's techniques, which allow the scientists to observe nanoscale structures near the atomic resolution, such as x-ray crystallography[1] or cryogenic electron microscopy[2], the data has to be reconstructed into representative 3D models in order to be visualized. Computer graphics remains the only viable means for interactive analysis and visualization of molecular structures at atomic resolution.

Nowadays, the dedicated programmable graphics hardware enables us to visualize structures that are orders of magnitude larger than a single protein. As the computational power of graphics hardware rose, we are now able to interactively visualize structures in so-called mesoscale, i.e., structures of the size of whole virus particles (so-called virions), in their atomic resolution. The ability to display structures of such sizes made it plausible to use molecular visualization to create complex molecular animations and interactive experiences. Such visualizations are invaluable in conveying detailed explanations of molecular processes that consist of large numbers of interacting molecules. The primary purpose of these explanatory visualizations is to be utilized in knowledge dissemination when the goal is to communicate certain aspects of the molecular realm to a general audience. The limited background knowledge of laypersons implies that the context has to be properly visualized together with the main concept.

Biomolecular processes are inherently multi-scale, so they have to be explained on different levels of detail. To convey them in a clear way to the lay audience, they have to be visualized as such. Generally, the processes start as interactions between individual molecules

depending on their atomic configurations, which in turn has many implications on larger, mesoscale molecular structures, such as cell organelles or even the entire cells. Displaying all of the necessary context for such a process would require rendering of an overwhelmingly large number of molecules.

With the currently available methods, the only plausible way to render such complex scenes is to simplify them to a large extent. This can be done, for example, by abstracting molecular details into simpler geometric forms. However, there are several problems associated with such an approach. When the goal is to illustrate the process to a layperson, who does not have any prior knowledge about the conveyed phenomenon or molecular visualization as such, biological structures cannot be arbitrarily simplified to the point which would require any additional background knowledge in order to understand the story. Representations of varying detail within a single story could introduce ambiguities that could lead to miscommunication of the intended message. This is the main motivation for developing methods that are able to display large mesoscale models, which serve for communicating molecular processes, entirely in the atomic resolution.

In general, when creating stories using molecular visualization, it is necessary to consider that the introduced abstraction is a separate visual channel that can unintentionally communicate information that may be misinterpreted by a lay audience. When conveying processes, such as the mechanisms of vaccines, there is a clear need to create the sense of trust in what is being shown. The implied realism inherent to models in atomic resolution might be a decisive aspect in the acceptance of the shown visualization to the lay audience as truth, rather than an uncertain, general concept. With processes, such as the mechanism of a vaccine, this is particularly important, as understanding of them by general audience might be crucial for their acceptance by people and thus influencing the public health.

There already exist methods that are able to interactively display large molecular scenes consisting of up to 15 billion atoms [3, 4, 5]. However, these methods rely on heavy level of detail simplification, which does not scale beyond individual molecules, and there is no general solution that can create simplification of every biological structure. Moreover, even with such an approach, they still cannot reach

the scales necessary for interactive exploration of very large structures, such as bacteria or even cells.

With these requirements and limitations in mind, we propose a method for interactive visualization of large molecular scenes, where the preservation of the atomic resolution is maintained for all visible molecules. Instead of further simplification of biological structure, our method evaluates the potential visibility of their molecules and removes those which will be occluded from all available viewpoints. It does so iteratively during rendering, thus significantly reducing the preprocessing step. In consequence, only the visible molecules are rendered, which presents a significant simplification in dense molecular scenes. Subsequently, we are able to render the individual atoms of the visible molecules at interactive frame rates even for scenes containing more than 10^{11} atoms.

Figure 1.1 demonstrates two scenes that were used for testing the performance of our algorithm. We used the latest knowledge about the SARS-CoV-19 virion structure and synthesized a complex scene consisting of 1000 instances of this virion. The second scene contains synthesized spheres that serve as proxies for the lack of modelled data of biological structures.

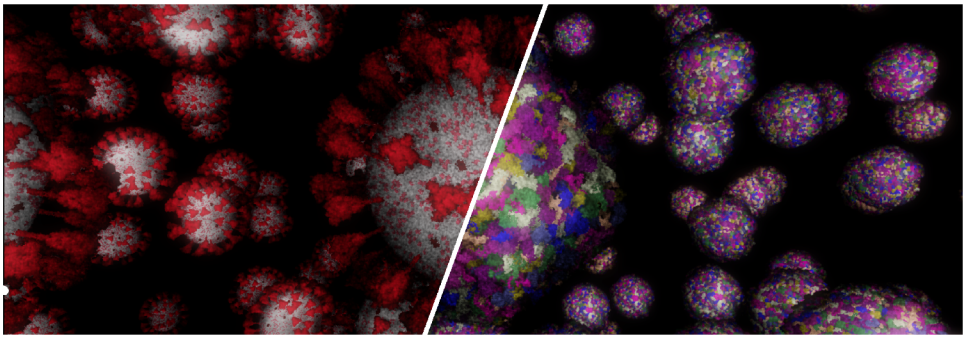


Figure 1.1: Two complex scenes that are forming the main target of our proposed algorithm. Showcased scenes consist of 1000 structures that together have around 14 million molecules and 90 billion atoms. Rendered by our proposed approach, the scenes run in real-time at between 10 and 30 frames per second.

2 Related Work

In this section, we provide the readers with the summary of the research most relevant to our solution. The chapter is divided into three parts. First, we present the closely related existing algorithms for rendering complex scenes, and we explain the insufficiency of these related methods in comparison to our situation. In the second part, we present the tools that are used for creation of the molecular models in the scale we are targeting and how they are used in storytelling. In the third part, we present methods of illustrative rendering that are utilized in molecular visualization for better comprehension and interpretation of a scene.

2.1 Algorithms for Rendering Complex Scenes

There are two main categories of algorithms aiming to improve the rendering time of complex and densely populated scenes, such as those that we are dealing with. Algorithms belonging to the first category try to remove occluded objects from a scene using hierarchical Z-buffer (Hi-Z) [6]. The second category is based on the replacement of objects in the scene by billboards, an approximate representation of the original objects. In the following sections, we explain the details of each approach.

2.1.1 Occlusion Culling

The first category of algorithms is focused on so called culling, i.e removal of objects that are occluded by others in the scene from a certain camera viewpoint. For this purpose, a Hi-Z algorithm was developed, and its variations became the most prevalent methods to cull occluded objects. Hi-Z is a two-pass algorithm. In the first pass, the depth buffer is generated by rendering occluders in place of the actual objects in the scene. Next, the mipmaps for the depth buffer are generated. For each object, based on the projected area of their bounding box in screenspace, a lookup against the mipmapped depth buffer is made to decide whether to render the object or cull it.

The Hi-Z method uses occluders for the efficient generation of a depth map. Occluders should satisfy two conditions: they should be significantly simpler than their original model so that their rendering time is low, and they should occlude as large space as possible so that they occlude the maximum possible number of objects.

Occluders can be different objects made with different methods. One possibility is to stochastically select a subset of a 3D model, in our case a molecule, as an occluder [7]. A more widely used method is to have a proxy for an object that can modeled manually by an artist or created automatically. There are two major methods for automatic creation. They both start by voxelizing an object. First method fills the voxelized space with bounding boxes [8]. A more advanced method presented by Silvennoinen et al. [9] finds several optimal plane cuts across the voxelized space that creates the most occlusion.

There are many variations of Hi-Z and many optimizations to reduce the speed or latency of the method. Typically, the method is implemented on the GPU [10]. On the GPUs, there are two methods to implement the method. First, there are occlusion queries [11], natively supported by the GPUs. Occlusion queries report whether fragments of an object passed the depth test. In addition, on some hardware, they can also report the number of those fragments, which can be used to aggressively cull objects that are almost invisible. Second, the method can be implemented manually through shader storage buffers [12]. If a particular fragment passes the depth test, this information is written to a storage buffer. Both of these approaches introduce some problems. In the case of occlusion queries, there is an overhead in latency when occlusion queries report back to the CPU. This problem can be alleviated in the manual approach as it can directly write draw calls into GPUs memory. However, empty draw calls, created by occluded objects, introduce cost too, and so it is necessary to remove them by compaction [13].

Final optimization for the Hi-Z, when used with polygonal models, is called meshlet culling [14]. The idea is to sort polygons in memory of a mesh such that when they are being processed by the GPU, they create optimal use of a vertex cache. This sorting creates clusters of polygons that are near each other in space, called meshlets. Subsequently, instead of rendering the object in a single draw call, individual

meshlets are culled and rendered, thus creating more fine-grained culling and optimal rendering.

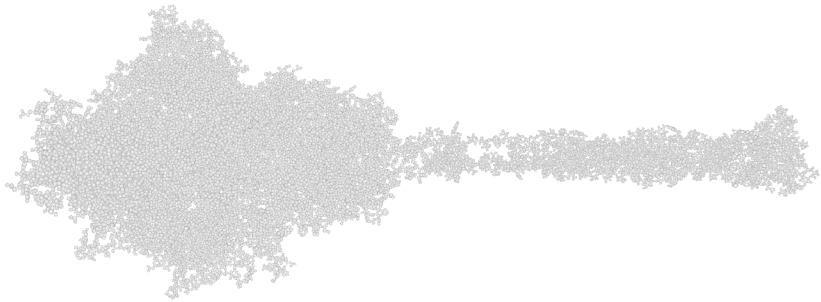
The problems of Hi-Z based methods for our case are two-fold. First, they require the creation of efficient occluders. The Protein Data Bank [15] contains hundreds of thousands of molecules, making it impossible to expect users to create those occluders manually, as it is done for computer games. However, automatic methods fail to generate satisfactory occluders due to the nature of molecular data, as demonstrated in Figure 2.1.

Second, the Hi-Z method needs to process each object at least twice, first to render occluders and then to sample the depth buffer. In-between, it needs to run a reduction operation to remove null draw calls for efficiency [13]. In our scenes containing hundreds of millions of occludees, the computation of such operation would be prohibitively expensive. A similar problem is described in the work of Grottel et al. [16] where they use a grid of cells and thousands of occlusion queries, for every single cell. They can hide the performance hit of occlusion queries on a single object consisting of atoms only, but their solution does not scale up to thousands of objects that we require in our scenes.

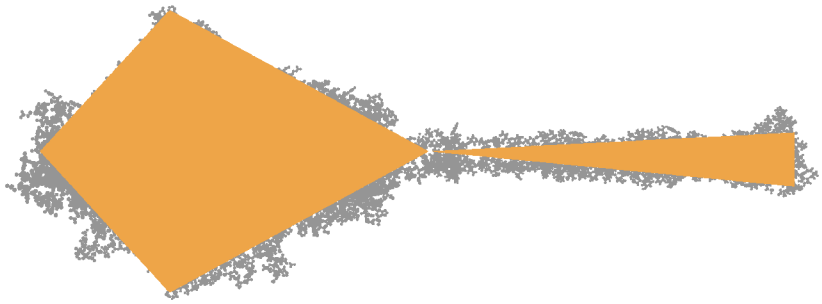
2.1.2 Billboards

The algorithms of the second category replace the objects, in our case the biological structures, with image-based representation. These representations are called impostors or billboards. Billboards can be 2D [17] or 3D [18, 19]. We enumerate several major billboard-based methods in each category.

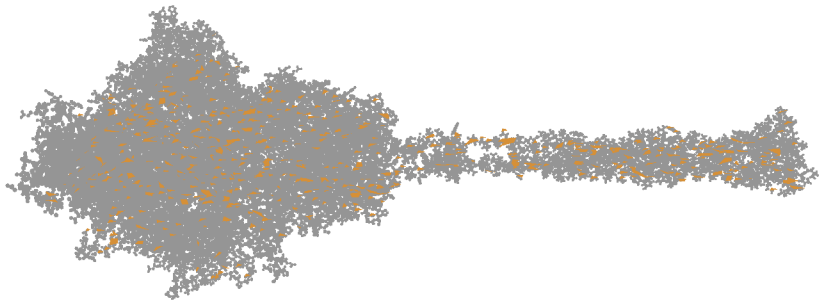
First impostors were planar and were introduced by Maciel et al. [20]. They consist of simple planar geometry, such as a quad. The generation of a planar impostor is not computationally expensive and thus can be done at runtime. Displaying it is also very fast because of its very low geometric complexity. On the other hand, parallax movements are not accounted for, so the impostor would need to be updated frequently. The method of planar impostors is unsuitable for our biological structures due to the lack of parallax movements.



(a) Stick model of the SARS-COV-2 spike protein, consisting of 32,126 atoms.



(b) Example of the desired ideal occluder consisting of only three triangles. Compared to the naive approach when we have one triangle for each atom, this represents a significant reduction.



(c) Real occluder generated by [9]. It consists of many small triangles that are covering only a small subset of the atoms.

Figure 2.1: Example demonstrating the difficulty with automatically generated occluders for molecular data. The occupation of the viewpoint by the molecule (in grey) should be overlapped by occluders (yellow) as much as possible and ideally with a small amount of triangles. The existing automatic solutions are producing many occluders of poor quality of coverage.

The second type of impostors are layered impostors [21, 22]. The basic idea is to use a set of image layers at different distances from the viewer, each representing a certain depth range.

To avoid image gaps between the layers, slightly overlapping depth regions are rendered into each layer. However, no real guarantee is given for avoiding image gaps. This method accounts for parallax movements and has been shown to integrate seamlessly into the scene. The number of parallax errors decreases with an increasing number of layers.

Compared to planar impostors consisting of only one image layer, layered impostors provide a larger valid viewing region. Jakulin [23] shows that this technique can be used for trees with high image quality by alpha blending between different impostor representations. Jeschke et al. [24] further extended layered impostors in several ways: a special spacing of the layers together with a new layer recording method ensures that all parallax movements are properly reproduced, and the impostor is guaranteed to contain all scene parts that might become visible. Layered impostors are much more suitable for biological structures.

The third type of impostors are billboard clouds [17]. The basic principle is to represent an object by a set of alpha-textured planes. In contrast to the planar and layered impostors, the position and orientation of the planes is optimized for approximating the object geometry so that an error tolerance in object space can be met for arbitrary viewpoints. The advantage of this view-independent impostor technique is that the viewer is not restricted to a bounded view cell, but only to a minimum distance to the impostor. In addition, it can be used directly for standard shadow mapping algorithms. Relighting can also be applied to billboard clouds.

Fourth and final types of impostors are called depth images. Depth images contain depth information per-pixel in addition to color for displaying images for new viewpoints. The transformation for projecting samples to a new image is called 3D image warp [25]. The warp can be performed in the forward direction by projecting every sample into the output image. McMillan et al. [26] provide an efficient method for back-to-front ordered warping of a depth image so that visibility is solved correctly without the need of a z-buffer. In contrast, searching the correct representant for an output pixel in the input image is called

backward warping. The latter can be seen as a kind of ray tracing and it is useful if only a few pixels have to be displayed because of the often costly search operations.

The per-pixel depth information allows accurate reproduction of parallax effects and makes it possible to correctly resolve visibility with the surrounding scene. However, occlusion artifacts in the output image may arise during display. In order to reduce them, multiple images can be warped to fill image gaps [27]. Another possibility is to enhance layered impostors with depth images, which contain multiple color and depth values for every pixel position [28].

The second category of billboards makes use of 3D representation. The simplest method involves splatting the geometry into 3D volume and then raycast through it to render the object. Another set of methods uses textures stored as volumes. Slice-based methods [29, 30] utilize a stack of parallel planes as a proxy geometry to sample the volume and render them by blending the slices.

Both of these methods are widely utilized because of their easy implementation and their ability to benefit from hardware trilinear texture interpolation and pre-integration schemes to generate high-quality images at a high framerate [31]. Both slice-based and raycasting approaches have mainly targeted the visualization of a single volume. Integrating multiple volumes that are intersecting with each other or with polygonal models is problematic with these methods.

The state-of-the-art method in 3D billboards is called volumetric billboards [18]. Volumetric billboards are composed of a set of triangular prism-shaped 3D cells with 3D texture coordinates at their vertices and a set of volume data stored into 3D textures, which are mapped into the cells. These cells are then intersected with planes perpendicular with the camera view and renderer back-to-front.

According to Cook et al. [32], none of the 2D billboards preserve enough quality and fully avoid parallax errors for objects that could be seen from all sides, like our biological structures. They state that voxel-based methods exhibit the best quality.

However, Cook et al. [32] note that even for a low-resolution rendering, volumes take around 18MB of memory. This amount of memory would be needed for every single biological structure in our scenes. In addition, the memory would have to be multiplied for every color scheme used in the visualization.

2.2 Modeling and Storytelling

As already stated, large molecular scenes with atomic detail are impossible to image with a microscope due to physical limitations. Because of that, such scenes are created in various dedicated modeling tools. Consequently, these models are then animated or presented in real-time using specialized visualization tools.

Several methods and tools have been developed to assist with the process of designing large mesoscale scenes. Plugins, such as MolecularMaya [33], ePMV 23 [34], or BioBlender [35], assist with the import, modeling, and animation of molecular structures within popular 3D animation packages. These tools expect experts to create scenes and animation manually.

Another category of tools enables non-expert users to create interactive mesoscale models. An example of that is CellPAINT [36], which is a fully manual tool that allows non-expert users to create interactive 2D mesoscale models with an illustrative design imitating hand drawings of David Goodsell [37]. A VR version was also implemented that allowed construction of models in immersive 3D stereoscopic view.

Some tools help automate some or all of the modeling or animation. The GraphiteLifeExplorer [38] streamlines the construction of complex assemblies with proteins and nucleic acids, allowing users to define 3D paths for DNA and other flexible chains, and populate the scene with associated protein structures. Sorger et.al. [39] created a concept of metamorpher. Metamorpher is a set of predefined operations for defining animation states as well as transitions for molecular visualization. They are re-usable templates for compatible data sets.

Le Muzic et al. [40] replaced simulation with omniscient intelligence and passive agents to provide real-time animation of molecular processes. They also introduced a tool called cellVIEW [5] for illustrative and multi-scale rendering. The tool is specifically aimed at visualizing models comprising of billions of atoms. It contains several techniques to allow rendering of such scenes. The main technique is a level of detail scheme that reduces clutter while increasing rendering speed.

The work of Le Muzic et al. [41] introduced equalized cutaway views for better visibility in scientific and educational settings. Sorger et al. [42] introduced transitions that deal with visualization of in-

complete data information in the developmental states of biological mesoscale models. Klein et al. [43] presented the first integrative structural modeling approach of the biological mesoscale. They were able to create interactively, under one second, an entire model of a biological structure given by a recipe. Previously the creation of such models from recipes could take minutes or even hours.

The created, animated, and rendered models communicate stories of processes inside of biological structures. Here we summarise some examples of storytelling use-cases. Klein et al. [44] used fully procedural animation to demonstrate microtubules' behavior. Kouřil et al. [45] created HyperLabels, labels that users can interact with, and combined them with cutaway views to provide the users with insight into the inner structure of the HIV virion. They also created camera transitions linked to the visibility of molecules to make the structure discoverable by the user. Nguyen et al. [46] presented a rule-based tool to construct mesoscale models and demonstrated it by creating a scientifically accurate, atomistic model of SARS-CoV-2 virion.

So far, the largest models for which the interactive frame rates for their visual exploration was reached were capturing a single virus. Therefore, interactive visualizations are nowadays focusing on similar scales. With better algorithms, interactive visualization and storytelling could scale to much larger models, such as those of bacteria or even entire human cells.

2.3 Illustrative Rendering of Molecules

The visualization of large molecular scenes is often crowded, making it difficult for the user to extract underlying information given by the rendering. Many methods were developed to make molecular rendering understandable.

The most important aspect is the coloring of the structures. The simplest method is to assign a unique color to each chemical element. Different units can be separated by color as well, such as chains, functional units, bonds, etc. More sophisticated approaches exist.

Classic approaches use only a single scale coloring scheme. Waldin et al. [47] introduced an adaptive method, called Chameleon, that adjusts the color scheme to distinguish the best visible structural in-

formation on the current scale level. The color scheme changes are interpolated from the previous views. Later, they introduced a system called Cuttlefish [48]. Cuttlefish smoothly redistributes color to maximize color range utilization give the viewing requirements.

There is no observable light at the molecular level. However, Blinn-Phong lighting is typically utilized to give a sense of depth and the ability to recognize individual atoms. Alternatively, toon shading is used to produce artistic rendering resembling the works of David Goodsell [49].

To give the user a sense of depth, ambient occlusion (AO) is typically utilized. AO is an approximation of global illumination. It calculates how exposed each point in a scene is to ambient lighting. This creates darker areas in creases of molecular structures, giving a perception of depth. This effect can be calculated using ray tracing or simulated in screen-space (SSAO). Due to the sheer complexity of large molecular scenes and the need for them to be rendered in real-time, screen-space techniques are mostly used.

Kock et al. [50] made an extension of SSAO to take into account semantic scene organization. Wahle et al. [51] tuned their implementation of SSAO to account for pockets and cavities of user-defined size.

To depict object boundaries, contours are often utilized [52]. The simplest approach to compute contours is to render every object twice: the first time enlarged by the size of desired line width, and in the second time the standard way, and then overlaying the original render creating an outlined render. If contours inside an object are desired as well, to enhance depth perception, differences between depth values can be used.

In order to draw the focus of a user, depth of field (DoF), an effect from photography, can be used to highlight a specific region in a scene [53]. DoF retains the focus area sharp while blurring the rest of the scene. Typically, DoF blurs parts of the image outside of a certain depth range. However, Kosara et al. [54] introduced semantic DoF that instead blurs regions outside of areas of focus. The DoF effect can be computed in screen-space efficiently as well.

3 Algorithm

In this chapter, we present our algorithm in detail. In the first section, we formalize our requirements. In the second section, we give an overview of the algorithm. In the subsequent sections, we go over the implementation details of each stage of the algorithm.

3.1 Initial Requirements

We list the main requirements that were driving the design of our solution. Following the content of typical scenes used for storytelling in biological research and the limitations posed by the target audience, we extracted the following requirements for our algorithm:

R1 Number of molecular structures can range from hundreds to thousands. Therefore, for each structure, a small amount of draw calls must be generated.

R2 Color scheme must be modifiable dynamically in real-time. This ability allows to depict or distinguish the currently best visible structural information[47, 48].

R3 Preservation of the existing molecular level of detail. Previous work used k-means method of reducing number of spheres per molecule based on the distance towards the camera [5]. It has been shown that this type of level of detail reduces visual clutter. Thus, any occlusion method developed should maintain the ability to render LODs of molecules per structure.

R4 Temperature-dependent random walk that captures features of the random diffusive motion that drives mesoscale dynamics [55]. It is typically a small animation that in dense molecular visualizations does not change visibility of particular molecules.

3.2 Overview

Based on the problems and drawbacks of available solutions, we designed an algorithm that enables a real-time rendering of large biological scenes by efficient caching of visible molecules and their instant retrieval.

We can assume that most of the mesoscale biological structures, such as the structure of SARS-CoV-2 virion, have a spherical or oval shape and they consist of densely packed molecules. Based on this assumption, we can expect that most of the molecules of a biological structure are not visible from a single camera viewpoint since they are inside the membrane or occluding each other.

This leads to our idea of caching the visibility information of the molecules of the individual biological structures. The visibility information is cached for a given camera viewpoint. It would be impossible to cache every possible camera viewpoint, so we cache only a subset and always pick the one closest to the user's camera viewpoint. The subset is determined based on the structure and available memory.

Storing the visibility information for each molecule from each viewpoint could take too much memory, making this approach infeasible. That is why we created an approach to store this information efficiently.

To further decrease the memory requirements, we also came up with a lossy compression method that can reduce the memory to an arbitrary limit. The information that we are losing with our compression is visibility of molecules. It means that we mark some molecules as visible when, in reality, they are occluded. We also store the information in preallocated memory with precise locations for each viewpoints so that we can retrieve a cached visibility of a single viewpoint in $O(1)$ time complexity.

With the visible molecules retrieved, we render their atoms as sphere impostors. We also apply several post-process effects, namely ambient occlusion, depth of field, and color grading.

Visualization of the entire pipeline is depicted in Figure 3.1.

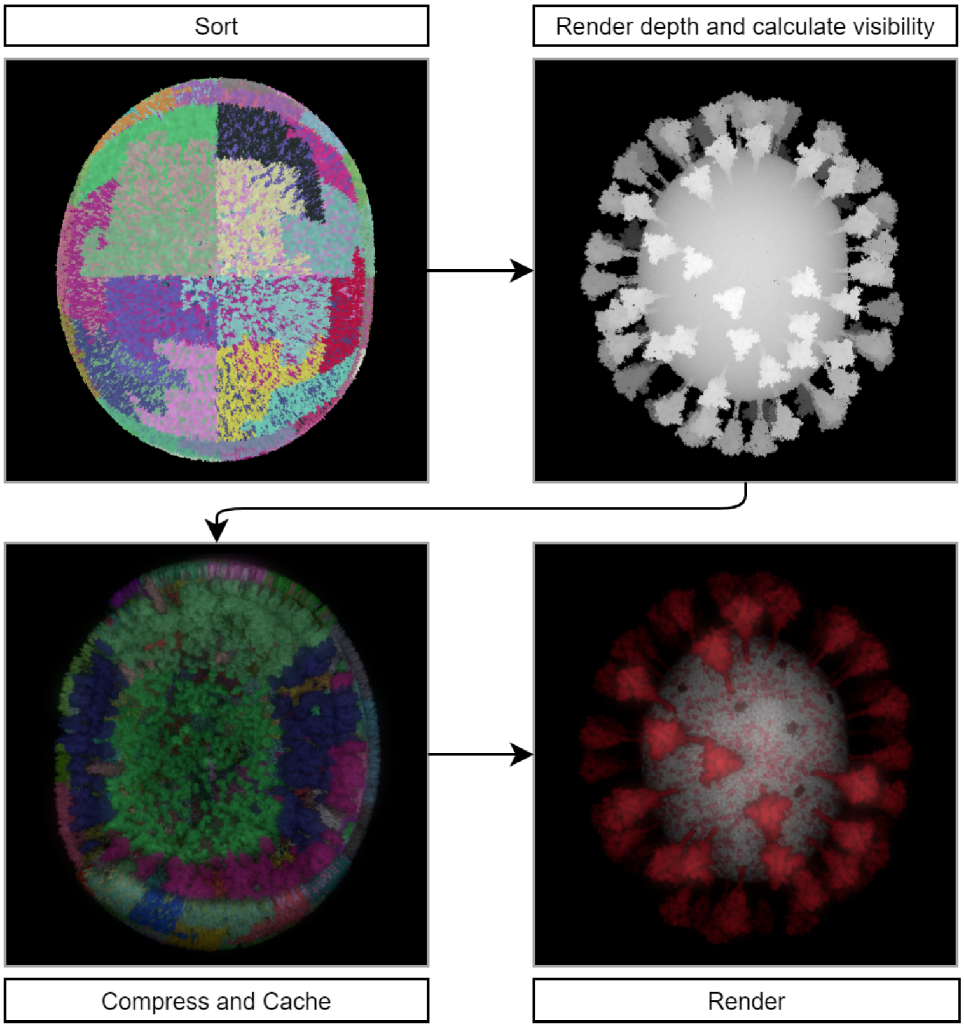


Figure 3.1: The stages of our method: 1) The preprocessing stage, which is executed during the data loading, sorts molecules of biological structures according to the Hilbert curve mapped onto a sphere. Continuous sorted patches are shown with different colors (some are omitted for clarity). Note that they may overlap as different types of molecules are sorted separately. 2) Visibility computation renders the depth map of a structure from a single viewpoint twice to retrieve visible molecules, which are then cached with our efficient compression based on the Hilbert curve and gap-filling. This stage is performed on-demand in real-time. 3) Retrieval stage retrieves the cached occlusion. 4) Last stage renders the structure.

The algorithm consists of the following stages.

- In the first stage, we load a scene into the memory. For each biological structure, we sort their molecules in memory so that in the later stages we can store and subsequently retrieve the information about their visibility from a set of predefined viewpoints. The predefined viewpoints are also precalculated here so that they can have fixed positions in memory and be easily retrievable in the last stage.
- In the second stage, during the rendering of a frame, we compute visibility information for the predefined viewpoints of biological structures selected based on the users' camera viewpoint. We compute only the viewpoints that have not been computed up to the frame. Subsequently, we compress and store this information.
- In the third stage, based on the user's camera viewpoint, we look up the stored visibility information generated in the second stage and render the final scene. The location of stored visibility is retrieved in $O(1)$ time thanks to the knowledge about the precalculated viewpoints' positions in memory from the first stage.
- In the last stage, we render the visible molecules.

3.3 Stage 1 – Preprocessing

The general idea behind our preprocessing stage is to have biological structures prepared in a way that allows us to efficiently store their visibility information in the second stage and retrieve it in the last stage.

First, we sort molecules in memory in such a way that when they are close to each other in space, they are also close to each other in memory. For this purpose, space-filling curves were developed. Among several space-filling curves, the Hilbert curve was shown to be the most effective [56]. For a single camera viewpoint, this allows us to efficiently store visible molecules as they can be represented as a single continuous memory range.

Most of the visible molecules are on the membrane of biological structures. This situation is demonstrated in Figure 3.3. To improve the locality criterion, we use the Hilbert curve mapped onto a sphere as in Purser et al. [57], instead of its spatial variant. Hilbert unit sphere is a unit sphere whose origin is aligned to the centroid of the biological structure. To get the Hilbert position of a molecule on a sphere, we map its local position onto Hilbert unit sphere as depicted in Figure 3.2. We find the position on Hilbert unit sphere by finding intersection of a line, crossing center of the sphere and the center of the molecule, and the sphere.

Figure 3.4 illustrates how the sorted visible molecules can be clustered. We can see that instead of storing visibility information for each molecule separately, we can store an entire cluster as a single range.

Second, as it would be impossible to store all possible views for each biological structure, we have to discretize camera viewpoints as part of this preprocessing stage. We calculate a bounding sphere for the biological structure and distribute viewpoints uniformly at every n degrees along the azimuth and polar angle of every biological structure's bounding sphere.

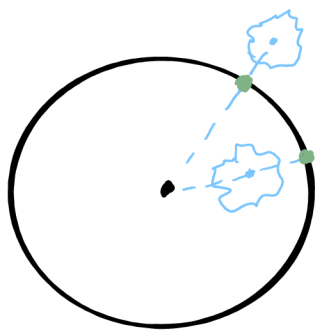


Figure 3.2: Depiction of a single camera viewpoint for dense biological structure. Visible molecules are in blue, occluded ones in gray. Only few molecules, mostly those forming a part of the shell, are visible.

Figure 3.3: Mapping of molecules of a biological structure on a unit sphere. The unit sphere is black, molecules are blue. Mapped positions are depicted in green.

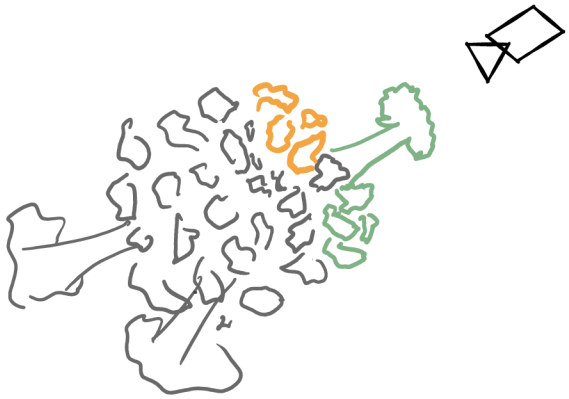


Figure 3.4: Depiction of visible molecules that can be stored as ranges. Different colors show separate ranges on the Hilbert curve.

Large n may cause a so-called popping effect when two discretized viewpoints are too far from each other. Moving a camera between them at the time of the switch from one to another may introduce sudden popping of visible molecules. To overcome this, n must be set to a sufficiently small value. By experimenting with several biological structures, we observed that any angle below five degrees is sufficient to avoid visible popping.

3.4 Stage 2 – Visibility Computation

Within the rendering loop, in each frame, we iterate over each biological structure in the viewing frustum. We pick its predefined camera viewpoint that is closest to the camera position. If the camera viewpoint already contains the visibility information in the cache, this stage is terminated for this biological structure. Otherwise, for each camera viewpoint, we find molecules from the corresponding biological structure that are visible and store this information.

Due to the use of graphics API in our implementation, WebGPU, we did not have access to occlusion queries. For this reason, we opted for a manual approach of obtaining visibility of molecules as described in Section 2.1.1. To obtain visible molecules for a given camera viewpoint, we render a biological structure twice from that viewpoint.

In the first pass, we render a depth buffer. In the second pass, for those fragments of the molecules that passed the depth test, we write a *true* value to a *visibility buffer* that stores the visibility information for each molecule separately. In the next step, we compress the visibility buffer into a *range buffer*, as depicted in Figure 3.6a. We find every interval of consecutive *true* values and store it to the range buffer as the range of indices where the interval starts and ends. This range will be pointing to the *content buffer* containing molecules of the biological structure.

In the compression phase, better results could be reached when we are operating on large ranges. However, due to some overlapping molecules, the Hilbert curve can become disjoint, resulting in a large range buffer. We demonstrate this scenario in Figure 3.5, where the red molecule is occluded by a bigger one in a viewing direction and disrupts the Hilbert curve.

Instead of a single molecule, this problem can be caused by an entire cluster of molecules. We denote these clusters as *gaps*. To remedy the problem of having too many gaps, we designed the following algorithm that removes them.

The algorithm assigns each gap a *cost* that is defined as a number of molecules of the gap multiplied by the number of their atoms. The cost represents how many additional occluded objects we render if we mark them as visible. Subsequently, gaps with the lowest cost are iteratively removed by joining the neighbouring ranges and with

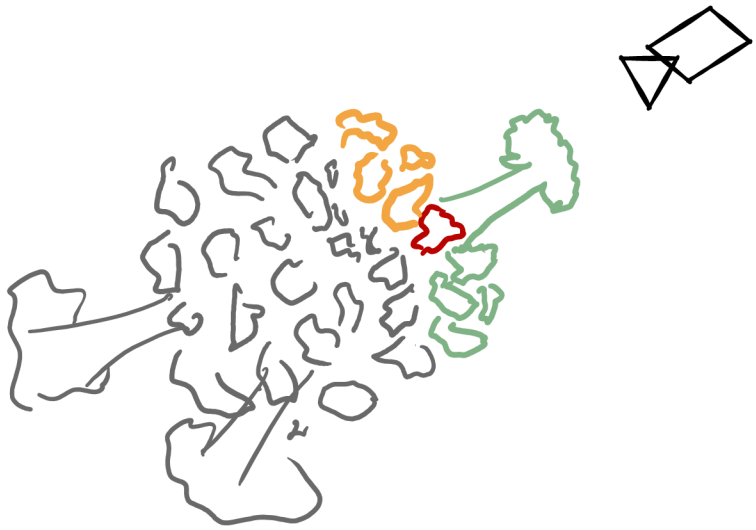
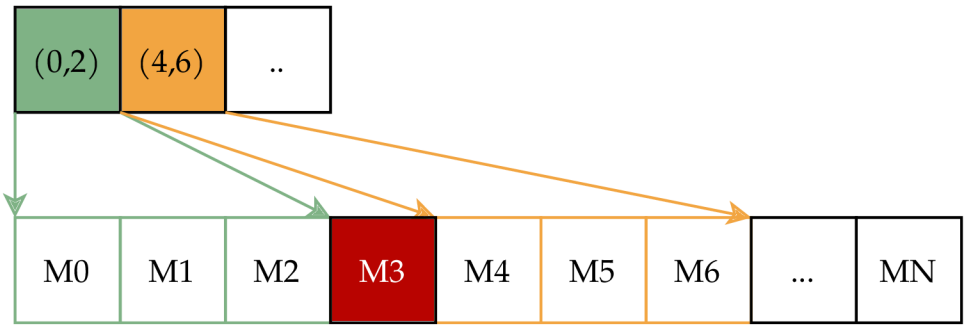


Figure 3.5: Depiction of how some occluded molecules cause disruption in the Hilbert curve ordering of visible molecules. Grey molecules are occluded in the depicted viewpoint. Yellow and green are visible disjoint clusters in memory. Red molecules form occluded clusters that cause disruption in the Hilbert curve.

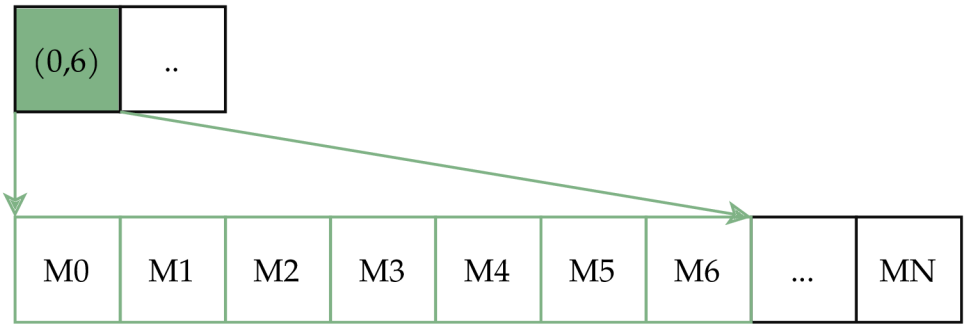
the molecules in the gap. The process is depicted in Figure 3.6a and Figure 3.6b. The gap removal creates a lossy compression for range buffers, exchanging lost information about occluded molecules for the saved memory. We discuss the efficiency of this tradeoff in Section 4.

The described iterative removal procedure stops when a user-defined limit is reached. This limit of the number of ranges is based on the available memory, and on the limit of draw calls that the system can process. Every single range adds eight bytes of memory and an additional draw call. Further discussion regarding the limit can be found in Section 4.

See algorithm 1 for simplified pseudocode of our algorithm. In the pseudocode, we present the main algorithm that finds the gaps as described in this section. The function operates on a single viewpoint of a single biological structure. For further details, you can find the entire source code in the attachment.



(a) The visibility information of scene depicted in Figure 3.5 in memory. One buffer contains ranges that point to clusters of molecules laid consecutively in memory in the second buffer. The molecule buffer is sorted according to the Hilbert curve.



(b) The visibility information of scenes depicted in Figures 3.5 and 3.6a after closing two ranges by incorporating the gap between them.

Figure 3.6: Memory layout of molecules of a biological structure and the associated visibility information. Data associated with each molecule are stored linearly, sorted according to the Hilbert curve. The information which molecules are visible is stored efficiently as ranges of indices pointing to this array.

Algorithm 1: Reduction algorithm.

Input: *ranges* array containing ranges, *ranges_limit* desired limit of ranges

Result: *new_ranges* reduced array of ranges to a limit given by *ranges_limit*

Data: *gaps* array containing found gaps

```
for i ← 0 to ranges.length do
    distance ← ranges[i − 1].end − ranges[i].start ;
    if distance > 0 then
        gaps.add(Gap{start : ranges[i − 1].end, end : ranges[i]});
    end
end
Sort gaps by cost;
while ranges.length > ranges_limit do
    ranges.add(gaps.last()) ;
    gaps.pop() ;
    ranges.length ← ranges.length − 1 ;
end
Sort ranges by start;
start ← ranges[0].start ;
for i ← 0 to ranges.length do
    if range[i + 1].start − range[i].end > 0 then
        new_ranges.add(Range{start : start, end : range[i].end});
        start ← ranges[i + 1].start ;
    end
end
```

3.5 Stage 3 – Retrieval of Cached Visibility

The third stage retrieves the biological structures from their visible viewpoints. With a fixed number of camera viewpoints established in Section 3.3 we first have to snap users' camera view to the closest camera viewpoint, for each biological structure, as depicted in Figure 3.7. Afterwards, we map the spherical position of all selected viewpoints to their indices pointing to range buffers, as outlined in Algorithm 2.

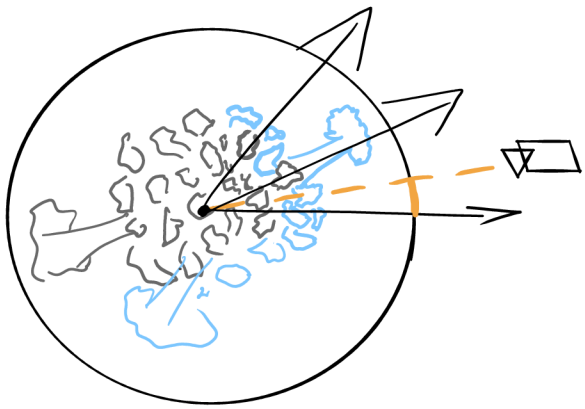


Figure 3.7: Camera viewpoints and snapping of users' camera to the closest viewpoint. Visible molecules from a snapped viewpoint are shown in blue. Yellow is the actual viewpoint that is being snapped to the closest one.

Algorithm 2: Spherical coordinates to index

1 n - number of degrees between two views;

2 $azimuth$, $zenith$ - spherical coordinates of the snapped view;

3 $steps \leftarrow 360/n$;

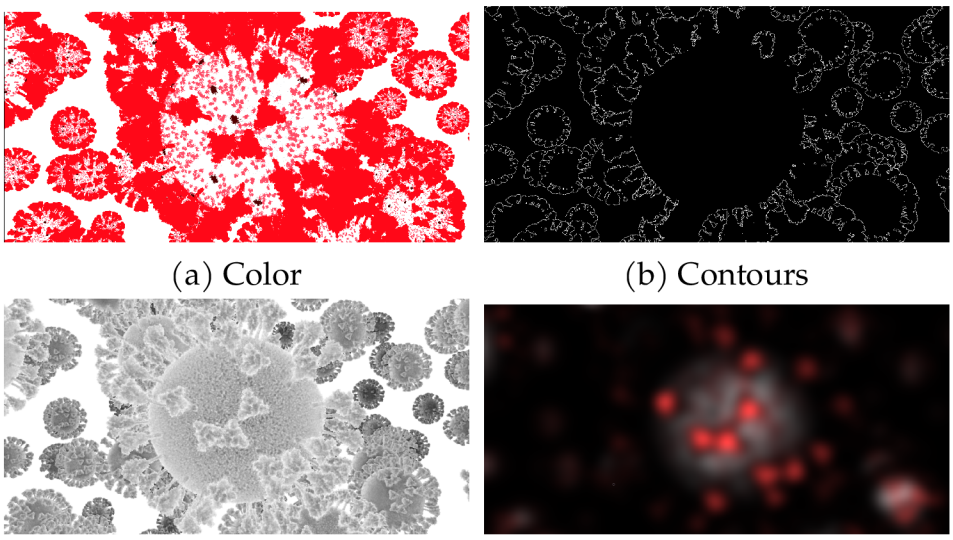
4 $index \leftarrow (azimuth/n) * steps + zenith/n$;

3.6 Stage 4 – Final Rendering

Once the visible molecules are retrieved, we can render them. We render atoms of molecules using sphere impostors, as in cellVIEW [5]. In the case of level of detail, resulting spheres of the k-means reduction algorithm are treated as atoms in terms of rendering.

Our fragment shader outputs color, instance ID, and normals. We created color schemes for our scenes manually. We use instance IDs to create contours of objects. Normals are sent into the SSAO computation. To retain stable framerate, we used SSAO implementation by AMD called Combined Adaptive Compute Ambient Occlusion [58], but ported it to WebGPU API. We mix two SSAO effects, one with small and one with a large radius in world space, to capture visibility at a molecular and at a structural level.

Finally, color, SSAO, and contours are combined. The output is blurred using DoF calculation that focuses on the structure in the center of a view. We also highlight some of the most important molecules by increasing the brightness of their color, blurring it, and then adding the color to the original image. Intermediate results that are combined together are shown in Figure 3.8.



(a) Color (b) Contours

(c) SSAO (d) Bloom

Figure 3.8: Post-process effects that are applied to our renders.

4 Results and Discussion

In this chapter, we summarize the benefits of the proposed algorithm and discuss how we managed to fulfill our initial requirements given the postulated assumptions. The demonstration of the usefulness, scalability, and performance of our approach is showcased on exemplary benchmarking scenes (see Figure 4.1).

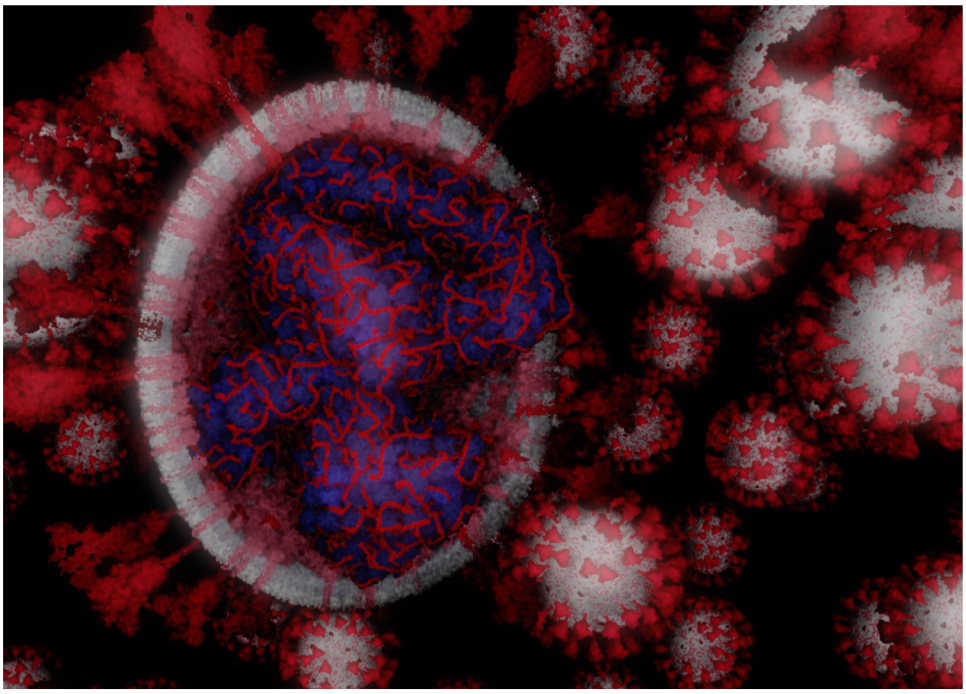


Figure 4.1: Detail of the benchmark scene containing 1000 models of SARS-CoV-2 [46]. The scene consists of $249,162,000$ molecules, i.e., $24,022,383,000$ atoms in total.

The first benefit, which is in line with our initial requirements, is that with our method, we can preserve all the detail down to the atomic level while reaching interactive frame rates even for massive scenes. Moreover, with our approach, we can perform operations that are impossible or cumbersome to reach when using a billboarding method. Among such operations is recoloring every single molecule or even individual atoms at any time. To a certain extent, our method also

supports the simulation of molecules' dynamic movements, which gives the scene a sense of realism, as molecules are naturally undergoing constant movements. We can animate these movements, i.e., support a jiggling of molecules and their atoms that resembles the Brownian motion. However, this can be allowed just to the degree when the translation of molecules does not change the information in the visibility buffer storing which molecules are occluded. From our observation, limiting the movements with respect to this restriction is sufficient for such a temperature-dependent random walk capturing features of the diffusive motion that drives the mesoscale dynamics [55]. Compared to the Hi-Z-based methods, our main benefit lies in the fact that we do not perform any additional computation on the GPU after caching the occlusion results.

To demonstrate the applicability and performance of our method, we focus our evaluation on measuring the gain in the rendering speed for the molecular scenes and the memory consumption. For more fine-grained performance measurement, we also separately tested the effectiveness of our gap removal algorithm.

As it is difficult to obtain molecular models of sizes our method is aiming for, we had to generate a synthesized benchmarking dataset. The basic building block of this dataset is a synthetic biological structure depicted in Figure 4.6, consisting of 80 million atoms. Using this model, we generated a molecular scene depicted in Figure 4.6 with 120 billion atoms.

As it is not meaningful to render molecules in far distance from the camera in atomistic resolution, we employed a level of detail technique described by Le Muzic et al. [40] that is targeting to render very large molecular scenes. We implemented the method in the programming language Rust and graphics API WebGPU. We chose WebGPU for its capability to handle large amounts of draw calls that our method can produce. The testing was performed with the implementation in the WebGPU API to handle large amounts of draw calls that can be produced. All benchmark tests were performed on Intel Core i9-98880H CPU and NVIDIA RTX 2070 Max-Q GPU in FullHD resolution.

First, we looked at whether our gap removal algorithm is effective in lossy compression. To observe its efficiency, we measured what percentage of molecules is marked as visible using different limit variable. The results for two benchmark datasets we used – the synthetic

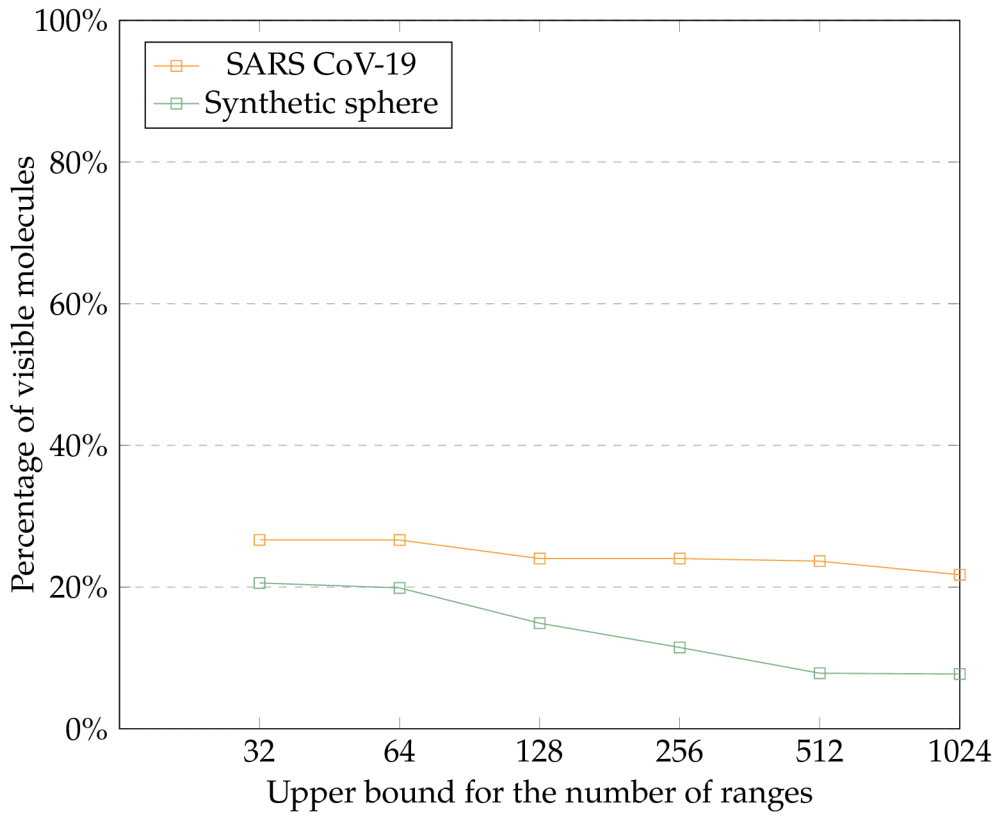


Figure 4.2: Percentage of visible molecules at different range limits for our two benchmarking datasets.

dataset (Figure 4.6) and the SARS-CoV-2 virions (Figure 4.1) – are shown in Figure 4.2. It can be observed that going from 1024 to 32 ranges, which is a reduction factor of 32, the compression marks only a small percentage of molecules as visible. To show how the reduction translates to the actual memory usage of range buffers, the following equation can be used for such calculation:

$$Bytes = Steps^2 * Limit * 8$$

Here $Steps$ equals to $360/N$ and N represents the distance between two camera viewpoints. For example, with a limit of 32 and N set to 5, we use 1.2 MB of memory for a single biological structure.

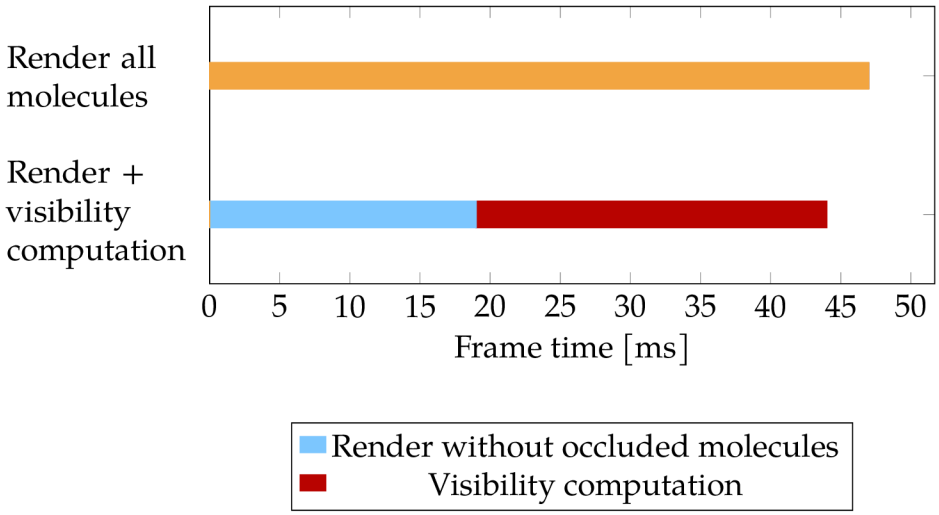


Figure 4.3: Performance test for camera zoomed-in to a single structure that is covering the entire screen, as depicted in Figure 4.6.

Second, we look at the benchmarks of rendering a single biological structure. Timings can be seen in Figure 4.3. As can be observed, for a single structure, there is a small benefit. A significant result is that the structure's visibility computation and rendering are only slightly slower than rendering the structure without occlusion computation.

The last benchmark operates on a synthetic scene shown in Figure 4.6. The scene contains 1,000 synthetic biological structures that together consist of 18 million molecules and 120 billion atoms. For every single frame, we calculate the visibility information for several camera viewpoints that have not been computed yet. Benchmark results can be seen in Figure 4.4. We observed that a short time after the application's start, the framerate stabilizes at a much higher value than without using our technique.

However, few performance drops can be observed that disappear after the next few seconds of running time, when either most or all of the occlusion is already computed and cached. We also show the improvements in rendering time across different zoom levels of the scene in Figure 4.5.

With our method, we can scale molecular visualization even further than just using reduction at molecular level, as in cellVIEW [5]. Our

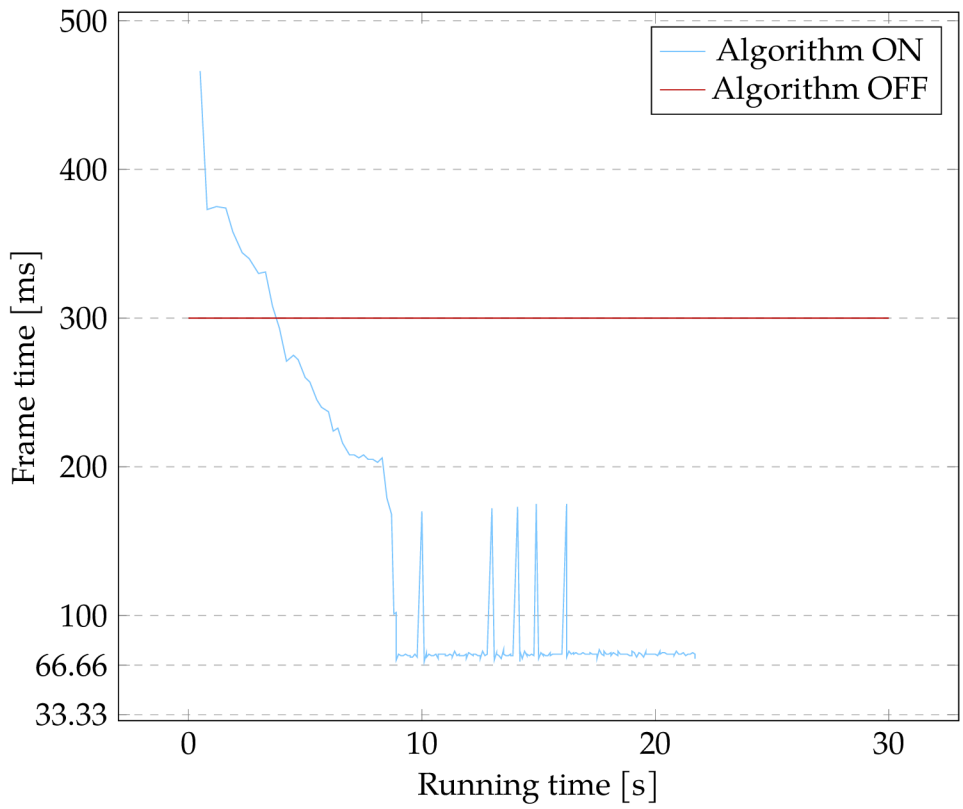


Figure 4.4: Benchmark of a synthetic scene with caching the visibility information. Timings across the running time of the application are in blue. The baseline without using our algorithm is in red. The camera is positioned in the middle of the scene.

limitation is that motion at the molecular level beyond Brownian cannot be applied, and clipping planes cannot use cached visibility.

However, these limitations are still less restricting than those of billboards. In our examples, when user's camera comes close to a single biological structure, we render all of its molecules with no techniques applied. We see this as an acceptable trade-off and a simulation of a real application.

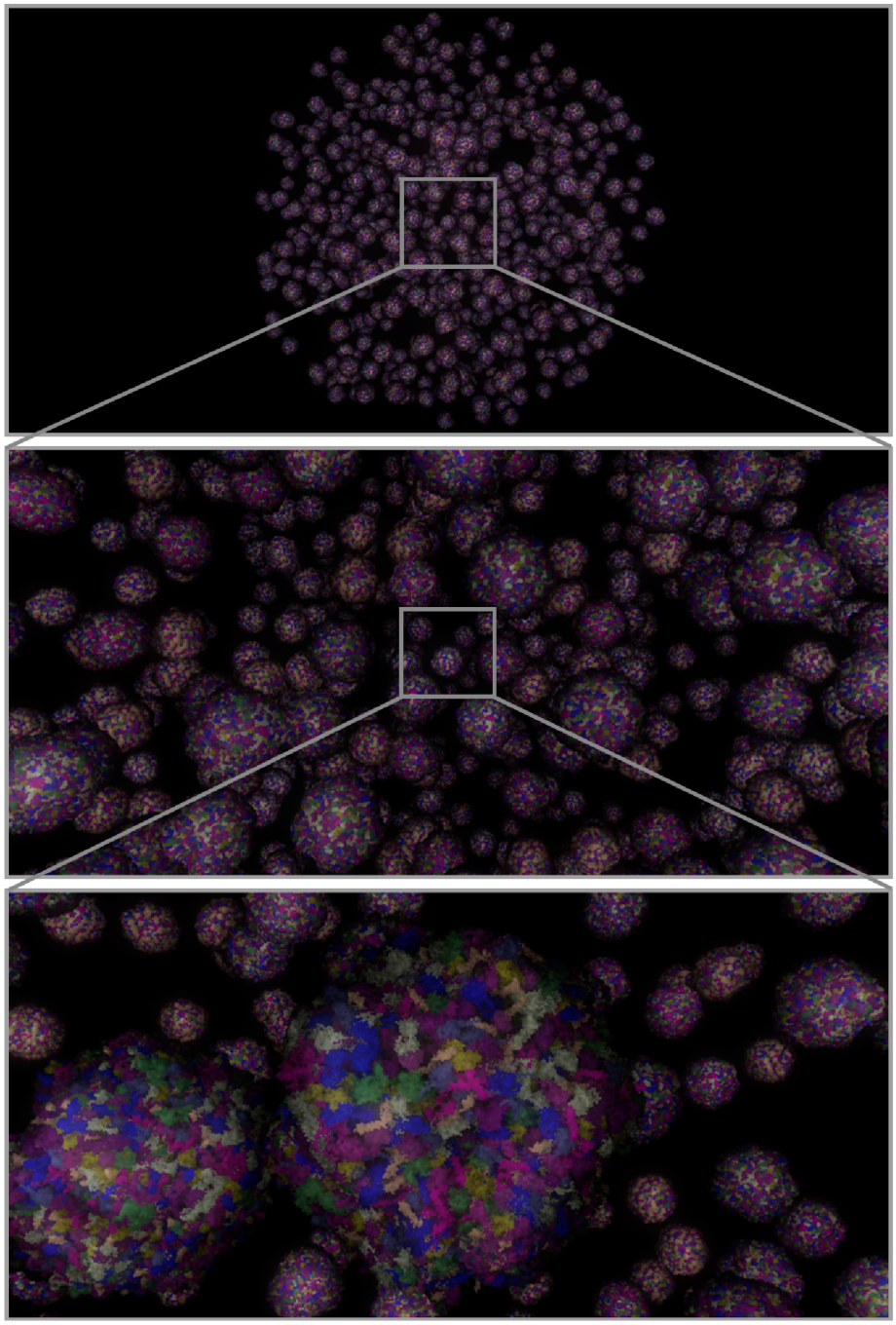


Figure 4.6: A synthetic benchmark scene. Top: Fully zoomed-out view to encompass the whole scene. Middle: Zoomed-out in middle of the scene. Bottom: Single structure.

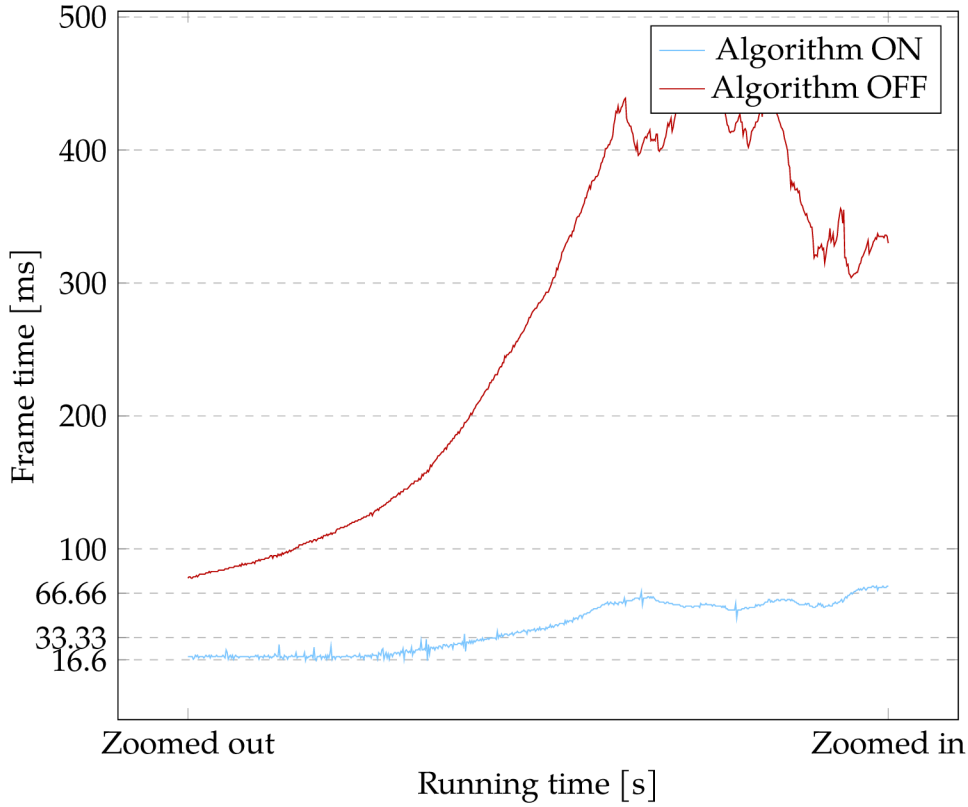


Figure 4.5: Benchmark of a synthetic scene with and without using our algorithm. Timings are across the whole scene, from a completely zoomed-out view capturing the whole scene to zoomed onto a single structure in the middle of the scene. Timings were performed after completing the computation of all visibility information.

5 Conclusion and Future Work

In this thesis, we presented a method to accelerate interactive visualizations of dense molecular scenes that enables us to render scenes where the number of atoms is ten times of magnitude larger than the existing state-of-the-art approaches. We demonstrated the usability of our proposed algorithm by its application on a synthetically generated scene that simulates the potential use case.

Using our method, we successfully reached the reduction of the rendering time by a factor ranging from 3 to 7, depending on the camera's position. Our method provides a competitive alternative to billboard-based methods while still operating with the geometric representations of objects in a scene. This has many advantages, such as the ability to recolor molecules and atoms, the option to use already established simplification level of detail approaches at the molecular level, and to simulate the Brownian motion. All these options are adding the necessary sense of realism and are significantly enhancing the perception and exploration process.

Our conducted benchmarking testing proved that using our approach, we are reaching significant improvements of the rendering process for large molecular scenes. Although the size of the currently available molecular models is not yet reaching those that our approach is able to render, the advancements in generating this biological knowledge and capturing capabilities are rapid and we expect models of such size to be available soon. When the models appear in the future, our approach is ready for their interactive rendering and exploration.

Our method comes with some limitations that we are aiming to overcome in the future. From our observations, at this scale, the rendering performance is limited by the number of viewport culling units. In relation to this, we see two potential future research directions. The first one is to expand the abstraction and simplification beyond the molecular level and thus make our solution more robust and universal with respect to the content of the scene. This may be quite difficult as it is impossible to say whether there can be found an automatic solution to preparing the levels of detail for each biological structure that has to be currently created manually. The second potential future extension is to replace rasterization with ray tracing. With too many

occluded hierarchical structures in a frame, it might be faster to trace a ray with the simultaneous usage of an efficient acceleration structure.

At the time of writing this thesis, a new deep learning method was developed [59] that can reliably predict the folding of protein structures. This will allow the generation of precise models of biological structures. Those models can then be visualized with our algorithm for further study. We see this as a validation of the need for visualization algorithms such as ours.

Bibliography

1. SMYTH, M S; MARTIN, J H J. X-Ray crystallography. *Molecular pathology : MP*. 2000, vol. 53, no. 1, pp. 8–14.
2. CHENG, Yifan; GRIGORIEFF, Nikolaus; PENCZEK, Pawel; WALZ, Thomas. A Primer to Single-Particle Cryo-Electron Microscopy. *Cell*. 2015, vol. 161, pp. 438–449. Available from DOI: `10.1016/j.cell.2015.03.050`.
3. FALK, Martin; KRONE, Michael; ERTL, Thomas. Atomistic Visualization of Mesoscopic Whole-Cell Simulations Using Ray-Casted Instancing. *Computer Graphics Forum*. 2013, vol. 32, no. 8, pp. 195–206. Available from DOI: `https://doi.org/10.1111/cgf.12197`.
4. LINDOW, N.; BAUM, D.; HEGE, H.-C. Interactive Rendering of Materials and Biological Structures on Atomic and Nanoscopic Scale. *Computer Graphics Forum*. 2012, vol. 31, no. 3pt4, pp. 1325–1334. Available from DOI: `https://doi.org/10.1111/j.1467-8659.2012.03128.x`.
5. LE MUZIC, Mathieu; AUTIN, Ludovic; PARULEK, Julius; VIOLA, Ivan. cellVIEW: a Tool for Illustrative and Multi-Scale Rendering of Large Biomolecular Datasets. *Eurographics Workshop on Visual Computing for Biomedicine*. 2015, vol. 2015, pp. 61–70. ISSN 2070-5778. Available from DOI: `10.2312/vcbm.20151209`.
6. GREENE, Ned; KASS, Michael; MILLER, Gavin. Hierarchical Z-Buffer Visibility. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. Anaheim, CA: Association for Computing Machinery, 1993, pp. 231–238. SIGGRAPH '93. ISBN 0897916018. Available from DOI: `10.1145/166117.166147`.
7. ZHANG, Hansong; MANOCHA, Dinesh; HUDSON, Tom; HOFF, Kenneth E. Visibility Culling Using Hierarchical Occlusion Maps. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 77–88. SIGGRAPH '97. ISBN 0897918967. Available from DOI: `10.1145/258734.258781`.

8. NAAJI, KARIM. *AABB Occluders* [http://karim.naaji.fr/blog/2019/15.11.19.html]. Accessed: 2020-09-27.

9. SILVENNOINEN, Ari; SARANSAARI, Hannu; LAINE, Samuli; LEHTINEN, Jaakko. Occluder Simplification Using Planar Sections. *Computer Graphics Forum*. 2014, vol. 33, no. 1, pp. 235–245. Available from DOI: 10.1111/cgf.12271.

10. ANAGNOSTOU, KOSTAS. *EXPERIMENTS IN GPU-BASED OCCLUSION CULLING* [https://interplayoflight.wordpress.com/2017/11/15/experiments-in-gpu-based-occlusion-culling/]. Accessed: 2020-09-27.

11. BITTNER, Jiří; WIMMER, Michael; PIRINGER, Harald; PURGATHOFER, Werner. Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. *Computer Graphics Forum*. 2004, vol. 23, pp. 615–624. Available from DOI: 10.1111/j.1467-8659.2004.00793.x.

12. HILL, Stephen; COLLIN, Daniel. Practical, Dynamic Visibility for Games. In: ENGEL, Wolfgang (ed.). *GPU Pro 2*. A K Peters, 2011, pp. 329–347.

13. HAAR, Ulrich; AALTONEN, Sebastian. *GPU Driven Rendering Pipelines* [http://advances.realtimerendering.com/s2015/aaltonenhaar_siggraph2015_combined_final_footer_220dpi.pdf]. Accessed: 2020-09-27.

14. HAAR, Ulrich; AALTONEN, Sebastian. *GPU-Driven Rendering Pipelines* [Presentation at SIGGRAPH 2015: Advances in Real-Time Rendering in Games]. 2015.

15. BERMAN, Helen M.; WESTBROOK, John; FENG, Zukang; GILLILAND, Gary; BHAT, T. N.; WEISSIG, Helge; SHINDYALOV, Ilya N.; BOURNE, Philip E. The Protein Data Bank. *Nucleic Acids Research*. 2000, vol. 28, no. 1, pp. 235–242. ISSN 0305-1048. Available from DOI: 10.1093/nar/28.1.235.

16. GROTTEL, Sebastian; REINA, Guido; DACHSBACHER, Carsten; ERTL, Thomas. Coherent Culling and Shading for Large Molecular Dynamics Visualization. In: *Proceedings of the 12th Eurographics / IEEE - VGTC Conference on Visualization*. Bordeaux, France: The Eurographs Association & John Wiley & Sons, Ltd., 2010,

pp. 953–962. EuroVis'10. Available from DOI: 10.1111/j.1467-8659.2009.01698.x.

17. DÉCORET, Xavier; DURAND, Frédo; SILLION, François X.; DORSEY, Julie. Billboard clouds for extreme model simplification. *ACM TRANS. GRAPH*. 2003, vol. 22, pp. 689–696.

18. DECAUDIN, Philippe; NEYRET, Fabrice. Volumetric Billboards. *Computer Graphics Forum*. 2009, vol. 28, no. 8, pp. 2079–2089. Available from DOI: https://doi.org/10.1111/j.1467-8659.2009.01354.x.

19. NEYRET, Fabrice. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. 1998, vol. 4, no. 1, pp. 55–70. ISSN 1077-2626. Available from DOI: 10.1109/2945.675652.

20. MACIEL, Paulo W. C.; SHIRLEY, Peter. Visual Navigation of Large Environments Using Textured Clusters. In: *In 1995 Symposium on Interactive 3D Graphics*. 1995, pp. 95–102.

21. MEYER, Alexandre; MEYER, Re; NEYRET, Fabrice. Interactive Volumetric Textures. In: *In Eurographics Rendering Workshop*. Eurographics, Springer Wein. ISBN, 1998, pp. 157–168.

22. SCHAUFLER, Gernot. Per-Object Image Warping with Layered Impostors. In: DRETTAKIS, George; MAX, Nelson (eds.). *Rendering Techniques '98*. Vienna: Springer Vienna, 1998, pp. 145–156. ISBN 978-3-7091-6453-2.

23. JAKULIN, Aleks. Interactive Vegetation Rendering with Slicing and Blending. In: *Eurographics 2000 - Short Presentations*. Eurographics Association, 2000. ISSN 1017-4656. Available from DOI: 10.2312/egs.20001004.

24. JESCHKE, Stefan; WIMMER, Michael; SCHUMAN, Heidrun. Layered Environment-Map Impostors for Arbitrary Scenes. In: *IN PROC. GRAPHICS INTERFACE*. 2002, pp. 1–8.

25. MCMILLAN, J. An Image-Based Approach to Three-Dimensional Computer Graphics. In: 1997.

26. MCMILLAN, L.; BISHOP, Gary. Plenoptic modeling: an image-based rendering system. In: *SIGGRAPH '95*. 1995.

27. RAFFERTY, Matthew M.; ALIAGA, D.; LASTRA, A. 3D image warping in architectural walkthroughs. *Proceedings. IEEE 1998 Virtual Reality Annual International Symposium* (*Cat. No.98CB36180*). 1998, pp. 228–233.

28. SHADE, J.; GORTLER, S.; HE, L.; SZELISKI, R. Layered depth images. In: *SIGGRAPH '98*. 1998.

29. CABRAL, B.; CAM, N.; FORAN, Jim. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: *VVS '94*. 1994.

30. LACROUTE, Philippe; LEVOY, M. Fast volume rendering using a shear-warp factorization of the viewing transformation. In: *SIGGRAPH '94*. 1994.

31. ENGEL, Klaus; ERTL, Thomas. Interactive High-Quality Volume Rendering with Flexible Consumer Graphics Hardware. In: *Eurographics 2002 - STARs*. Eurographics Association, 2002. ISSN 1017-4656. Available from DOI: `10.2312/egst.20021051`.

32. COOK, Robert L.; HALSTEAD, John; PLANCK, Maxwell; RYU, David. Stochastic Simplification of Aggregate Detail. *ACM Transactions on Graphics*. 2007, vol. 26, no. 3, pp. 79–es. ISSN 0730-0301. Available from DOI: `10.1145/1276377.1276476`.

33. *MolecularMaya*. Available also from: `https://clarafi.com/tools/mmaya/`.

34. JOHNSON, Graham T.; AUTIN, Ludovic; GOODSELL, David S.; SANNER, Michel F.; OLSON, Arthur J. ePMV Embeds Molecular Modeling into Professional Animation Software Environments. *Structure*. 2011, vol. 19, no. 3, pp. 293–303. ISSN 0969-2126. Available from DOI: `10.1016/j.str.2010.12.023`. Publisher: Elsevier.

35. *BioBlender*. Available also from: `http://www.bioblender.org/`.

36. GARDNER, A.; AUTIN, L.; BARBARO, B.; OLSON, A. J.; GOODSELL, D. S. CellPAINT: Interactive Illustration of Dynamic Mesoscale Cellular Environments. *IEEE Computer Graphics and Applications*. 2018, vol. 38, no. 6, pp. 51–66. ISSN 1558-1756. Available from DOI: `10.1109/MCG.2018.2877076`.

37. GOODSELL, David S. Inside a living cell. *Trends in Biochemical Sciences*. 1991, vol. 16, no. 6, pp. 203–206. ISSN 0968-0004. Available from DOI: https://doi.org/10.1016/0968-0004(91)90083-8.

38. HORNUS, Samuel; LÉVY, Bruno; LARIVIÈRE, Damien; FOURMENTIN, Eric. Easy DNA modeling and more with GraphiteLifeExplorer. *PloS one*. 2013, vol. 8, no. 1, pp. e53609–e53609. ISSN 1932-6203. Available from DOI: 10.1371/journal.pone.0053609. Edition: 2013/01/07 Publisher: Public Library of Science.

39. SORGER, Johannes; MINDEK, Peter; RAUTEK, Peter; GRÖLLER, Meister Eduard; JOHNSON, Graham; VIOLA, Ivan. Metamorphers: Storytelling Templates For Illustrative Animated Transitions in Molecular Visualization. In: *Proceedings of the Spring Conference on Computer Graphics 2017*. Mikulov, Czech Republic, 2017, pp. 27–36. SCCG '17. Available also from: https://www.cg.tuwien.ac.at/research/publications/2017/sorger_2017_metamorphers/.

40. LE MUZIC, M.; PARULEK, J.; STAVRUM, A. K.; VIOLA, I. Illustrative Visualization of Molecular Reactions Using Omniscient Intelligence and Passive Agents. In: Swansea, Wales, United Kingdom: Eurographics Association, 2014, pp. 141–150. EuroVis '14.

41. MUZIC, Mathieu Le; MINDEK, Peter; SORGER, Johannes; AUTIN, Ludovic; GOODSELL, David; VIOLA, Ivan. Visibility Equalizer: Cutaway Visualization of Mesoscopic Biological Models. *Computer Graphics Forum*. 2016, vol. 35, no. 3. Available also from: https://www.cg.tuwien.ac.at/research/publications/2016/lemuzic-mindek-2016-viseq/.

42. SORGER, Johannes; MINDEK, Peter; KLEIN, Tobias; JOHNSON, Graham; VIOLA, Ivan. Illustrative Transitions in Molecular Visualization via Forward and Inverse Abstraction Transform. In: S. BRUCKNER, B. Preim; VILANOVA, A. (eds.). *Eurographics Workshop on Visual Computing for Biology and Medicine* (*VCBM*). Bergen, 2016, pp. 21–30. Available also from: https://www.cg.tuwien.ac.at/research/publications/2016/sorger-2016-fowardabstraction/.

43. KLEIN, Tobias; AUTIN, Ludovic; KOZLÍKOVÁ, Barbora; GOODSELL, David; OLSON, Arthur; GRÖLLER, Meister Eduard; VIOLA, Ivan. Instant Construction and Visualization of Crowded Biological Environments. *IEEE Transactions on Visualization and Computer Graphics*. 2018, vol. 24, no. 1. Available from DOI: 10.1109/TVCG.2017.2744258.

44. KLEIN, T.; VIOLA, I.; GRÖLLER, E.; MINDEK, P. Multi-Scale Procedural Animations of Microtubule Dynamics Based on Measured Data. *IEEE Transactions on Visualization and Computer Graphics*. 2020, vol. 26, no. 1, pp. 622–632.

45. KOUŘIL, D.; ISENBERG, T.; KOZLÍKOVÁ, B.; MEYER, M.; GROELLER, E.; VIOLA, I. HyperLabels: Browsing of Dense and Hierarchical Molecular 3D Models. *IEEE Transactions on Visualization and Computer Graphics*. 2020, pp. 1–1. Available from DOI: 10.1109/TVCG.2020.2975583.

46. NGUYEN, Ngan et al. *Modeling in the Time of COVID-19: Statistical and Rule-based Mesoscale Models*. 2020. Available from arXiv: 2005.01804 [q-bio.QM].

47. WALDIN, Nicholas; MUZIC, Mathieu Le; WALDNER, Manuela; GRÖLLER, Eduard; GOODSELL, David; LUDOVIC, Autin; VIOLA, Ivan. Chameleon - Dynamic Color Mapping for Multi-Scale Structural Biology Models. In: BRUCKNER, Stefan; PREIM, Bernhard; VILANOVA, Anna; HAUSER, Helwig; HENNEMUTH, Anja; LUNDERVOLD, Arvid (eds.). *Eurographics Workshop on Visual Computing for Biology and Medicine*. The Eurographics Association, 2016. ISBN 978-3-03868-010-9. ISSN 2070-5786. Available from DOI: 10.2312/vcbm.20161266.

48. WALDIN, N.; WALDNER, M.; LE MUZIC, M.; GRÖLLER, E.; GOODSELL, D. S.; AUTIN, L.; OLSON, A. J.; VIOLA, I. Cuttlefish: Color Mapping for Dynamic Multi-Scale Visualizations. *Computer Graphics Forum*. 2019, vol. 38, no. 6, pp. 150–164. Available from DOI: 10.1111/cgf.13611.

49. GOODSELL, David. *The Machinery of Life*. 2009. Available from DOI: 10.1007/978-0-387-84925-6.

50. KOCH, Thomas Bernhard; KOUŘIL, David; KLEIN, Tobias; MINDEK, Peter; VIOLA, Ivan. Semantic Screen-Space Occlusion for Multiscale Molecular Visualization. *Eurographics Workshop on Visual Computing for Biology and Medicine*. 2018. Available also from: `https://www.cg.tuwien.ac.at/research/publications/2018/koch_bernhard_2018-1/`.

51. WAHLE, M.; WRIGGERS, W. Multi-scale Visualization of Molecular Architecture Using Real-Time Ambient Occlusion in Sculptor. *PLoS Computational Biology*. 2015, vol. 11. Available from DOI: `10.1371/journal.pcbi.1004516`.

52. ZWAN, Matthew; LUEKS, Wouter; BEKKER, Henk; ISENBERG, Tobias. Illustrative Molecular Visualization with Continuous Abstraction. *Comput. Graph. Forum*. 2011, vol. 30, pp. 683–690. Available from DOI: `10.1111/j.1467-8659.2011.01917.x`.

53. FALK, Martin; KRONE, Michael; ERTL, Thomas. Atomistic Visualization of Mesoscopic Whole-Cell Simulations Using Ray-Casted Instancing. *Computer Graphics Forum*. 2013, vol. 32, pp. 195–206. Available from DOI: `10.1111/cgf.12197`.

54. KOSARA, Robert; MIKSCH, Silvia; HAUSER, Helwig. Semantic Depth of Field. In: *Proceedings of the IEEE Symposium on Information Visualization 2001* (*INFOVIS'01*). USA: IEEE Computer Society, 2001, p. 97. INFOVIS '01. ISBN 0769513425.

55. MUZIC, M. L.; WALDNER, M.; PARULEK, J.; VIOLA, I. Illustrative Timelapse: A technique for illustrative visualization of particle-based simulations. *2015 IEEE Pacific Visualization Symposium* (*PacificVis*). 2015, pp. 247–254.

56. MOON, B.; JAGADISH, H. V.; FALOUTSOS, C.; SALTZ, J. H. Analysis of the clustering properties of the Hilbert space-filling curve. *IEEE Transactions on Knowledge and Data Engineering*. 2001, vol. 13, no. 1, pp. 124–141. Available from DOI: `10.1109/69.908985`.

57. PURSER, Robert James. Construction of a Hilbert curve on the sphere with an isometric parameterization of area. 2009. Available also from: `https://repository.library.noaa.gov/view/noaa/11402`.

58. AMD. *Combined Adaptive Compute Ambient Occlusion* [https://gpuopen.com/fidelityfx-cacao/]. Accessed: 2020-12-28.

59. SENIOR, Andrew W. et al. Improved protein structure prediction using potentials from deep learning. *Nature*. 2020, vol. 577, no. 7792, pp. 706–710. ISSN 1476-4687. Available from DOI: 10.1038/s41586-019-1923-7.

A List of Electronic Attachments

Along with the thesis, the attachments are included in attachments.zip file consisting of:

- `data/` data of biological structures used in generated outputs in the thesis
- `kmeans/` source code of library for reducing molecules using k-means algorithm
- `lod_creator/` source code of application that creates level of detail for molecules
- `sphere_creator/` source code of application that generates our synthetic spheres
- `master_thesis/` main project folder with source code of applications that were used to generate outputs in this thesis. See README.md for instructions of compiling and running the code.