# WebMGA 2.0
# WebGL Molecular Graphics Application for the Interactive Rendering of Convex-Body Fluids

Supervisor:
Prof. Guido Germano
Faculty of Engineering
Department of Computer Science

University College London

April 2023

# Chapter 1

# Introduction

WebMGA 2.0 is a WebGL-based real-time graphic application that specialises in the rendering of coarse-grained models of molecular fluids. Molecular fluid visualisation often relies on non-spherical convex rigid bodies as elementary units, which are not readily available in conventional molecular graphics software. WebMGA 2.0 fills this gap by providing accurate and interactive rendering of large ensembles of prolated and elongated convex bodies.

WebMGA 2.0 is a refinement of WebMGA 1.0, which evolved from QMGA, a liquid crystal graphic software published in 2008 written in C++. Due to significant portions of its codebase becoming depreciated, the installation process of QMGA is extremely tedious. As a solution, WebMGA 1.0 was developed to inherit QMGA's ability to provide accurate rendering of liquid crystals. However, WebMGA 1.0 lacks some features from QMGA and has significantly lower performance. WebMGA 2.0 aims to complete QMGA's feature set and improve the performance, at the same time provide a intuitive user interface. WebMGA 2.0 is implemented in JavaScript and the WebGL-derived library Three.js.

WebMGA 2.0 extends the capabilities of WebMGA 1.0 by introducing new features such as periodic boundary conditions, real-time rendering of molecular movement motions videos and slicing. These features are particularly useful in the field of liquid crystal research as providing methods for in-depth observations and analyses of molecular systems, which are crucial for gaining a comprehensive understanding of their structure and dynamics.

Given that typical sizes of molecule systems in liquid crystal research field are often large, performance is the key of the usability of WebMGA 2.0. WebMGA 2.0 has achieved notable improvements in performance

by optimising its rendering procedures and methods. WebMGA 2.0 is capable of efficiently rendering large-scale systems of molecules containing $50^4$ particles, with a satisfactory average frame rate of 20.0, which fulfils most requirements in this field.

WebMGA 2.0 is deployed at https://hiiamyue.github.io/WebMGA-2/. For the best results, a google Chrome browser is recommended.

One of the challenges encountered during the development of WebMGA 2.0 is the need to navigate through the code base of WebMGA 1.0 which lacks comments for most functions.

## 1.1 Aims and goals

• Install both WebMGA 1.0 and QMGA. Compare and analyse the functionalities between two and report missing features as well as areas of improvements.

• Create lists of functional and non functional requirements for WebMGA 2.0.

• Research and learn Three.js rendering fundamentals, basic knowledge of computer simulations of liquid crystal and scene optimisation techniques.

• Implement required functionalities and deliver a bug free web app of WebMGA.

• Improve the performance of WebMGA to meet the standard for rendering large system.

• Deliver a tool that is helpful in research of the liquid crystal field.

• Produce several videos demos for Video functionalities with different setting e.g slicing applied.

• Produce clear documentation and commented codebase.

• Conduct testing on the new version of WebMGA and provide critical analysis on the result.

• Provide suggestion for future work.

# Contents

# Chapter 2

# Requirements

## 2.1 Evaluation of WebMGA 1.0 and QMGA

Currently, QMGA stands as the sole visualisation tool for liquid crystals research since its publication in 2008. However, accessing it from a modern computer is proven to be challenging. The primary aim of WebMGA 2.0 is to offer researchers in this field the same capabilities as QMGA while simultaneously providing a better user experience.

WebMGA 1.0 successfully transformed fundamentals of QMGA into a modern web based application. In particular, here are some advantages of WebMGA 1.0:

**Rendering accuracy:** WebMGA 1.0 accurately recreated the molecular shapes available in QMGA into JavaScript, such as ellipsoids, spherocylinders, spheroplatelets and cut spheres. Additional functionalities are provided to allow modification of the geometries of these shapes. It inherited QMGA's advanced colour coding scheme which calculates the colour of each particle based on its orientation. This feature enables users to easily observe the overall phase order at a glance. Overall, WebMGA 1.0 is able to achieve a high level of rendering accuracy in geometries, orientation and colouring of a molecule, producing same simulation output as QMGA.

**User interface and system design:** QMGA interface primarily relies on horizontal navigation bars, which contain several unlabelled buttons that may not be intuitive for users. WebMGA 1.0 has a brand new user interface design that incorporates a sub-menu to separate different features in parallel with the navigation bar. All the functionalities are well defined and labelled with similar featured grouped and placed in a logical position. WebMGA 1.0's codebase is designed with a well-structured model-view-controller architec-

ture. This design pattern enables the separation of concerns, which simplifies the maintenance and modification process for future developers. QMGA is implemented in a single 5000+ line file, making it sophisticated for future development.

WebMGA 2.0 should ensure that improvements in other areas made do not compromise the existing strengths of WebMGA 1.0.

WebMGA 1.0 has been noted to have some limitations, making it an incomplete solution. Here are several areas where WebMGA 1.0 needs improvement to match the level of capability offered by QMGA and to provide a comprehensive and efficient visualisation tool:

**Incomplete feature sets:** WebMGA 1.0 lacks support for key features such as periodic boundary conditions and video rendering, at the same time It exhibits limitations in functionalities such as slicing and customisation of model configurations. These features are valuable tools that enhance QMGA's capacity to provide additional insights of liquid systems for researchers.

WebMGA 2.0 should prioritise the implementation of these features.

**User interface deficiency:** WebMGA 1.0 implements a intuitive and user-friendly interface. However, the system has been reported to contain frequent bugs that can cause it to halt or malfunction. To ensure the effectiveness and reliability of WebMGA 2.0, it is crucial to provide a bug-free version that is thoroughly tested and verified before release.

**Rendering speed:** According to the performance test results from WebMGA 1.0's development thesis (Figure 10.6), WebMGA 1.0 is able to render a system of a maximum size of $10^3$ with satisfactory FPS range between 10-20 [6]. This level of rendering speed is not sufficient to meet the performance needs for rendering liquid crystal systems, especially considering that a relatively small system can consist of over $10^4$ objects.

In order for WebMGA 2.0 to be practically and useful in this filed, it is necessary for WebMGA 2.0 to optimise the rendering speed. An ideal performance benchmark for WebMGA 2.0 would be to achieve a similar level of performance as QMGA, which is capable of smoothly rendering large-scale systems of size $10^5$ [3].

## 2.2 Functionality requirements

Functional requirements can be divided into two parts: requirements for adding new features and requirements for removing existing bugs.

### 2.2.1 New features

Following a comprehensive evaluation of the features offered by QMGA and WebMGA 1.0, a detailed set of functional requirements has been developed for the new features that were not available in WebMGA 1.0 such as periodic boundary conditions(PBC). These requirements are presented in the following Table 2.1.

| | ID | Detail | Priority | Achieved |
|---|---|---|---|---|
| **PBC** | 1.1 | Perform test on whether selected model is bounded inside unit box | Must | **Y** |
| | 1.2 | Apply periodic boundary conditions and fold model molecules into unit box | Must | **Y** |
| | 1.3 | Deselect the fold toggle to display original unfolded model | Must | **Y** |
| | 1.4 | Disable folding for already folded molecules and provided insight | Must | **Y** |
| **Video** | 2.1 | Allow uploading and parsing multiple file inputs | Must | **Y** |
| | 2.2 | Render the positions and orientations of each configuration accurately | Must | **Y** |
| | 2.3 | Generate and download smooth video animations for user input configurations | Must | **Y** |
| | 2.4 | Real time video rendering on screen | Could | **Y** |
| | 2.5 | Allow user to apply additional functionalities to video e.g apply slicing or PBC | Mush | **Y** |
| | 2.6 | User specified video frame rate | Could | **Y** |
| | 2.7 | User specified download file name | Could | **Y** |
| **IO** | 3.1 | Allow user to specify image resolution for download | Must | **Y** |
| **Culling** | 4.1 | Apply backface culling to improve performance | Must | **Y** |
| | 4.2 | Apply frustum culling to improve performance | Must | **Alternative** |
| | 4.3 | Apply occlusion culling to improve performance | Must | **Alternative** |

Table 2.1: Functional requirements for new features.

### 2.2.2 Refining the existing features

During an examination of the existing functionalities in WebMGA 1.0, errors were identified in various parts of the program, with most errors occurred in the user interface. A summary of these errors and its priority has been compiled in the following table.

| | ID | Detail | Priority | Fixed |
|---|---|---|---|---|
| **User interface** | 5.1 | Numerical input box produces NaN input causing program to crash | High | **Y** |
| | 5.2 | View state at side menu does not update after switched model | High | **Y** |
| | 5.3 | Background colour is inconsistent with some model has back background and some has white | Medium | **Y** |
| | 5.4 | Front end layout not responsive to change in screen size | Medium | **N** |
| **Model** | 6.1 | Memory leak caused by not disposing old meshes and geometries after switching model | High | **Y** |
| | 6.2 | Memory leak caused by assigning new cameras every time user switches model | High | **Y** |
| **Slicing** | 7.1 | Sliced molecules were rendered hollow | High | **Y** |
| | 7.2 | Slicing is enabled constantly in background, larger molecule always not shown completely | medium | **Y** |
| **Unit Box** | 8.1 | Unit box of previous model is generated when switching model | High | **Y** |
| | 8.2 | Wrong size of unit Box is generated for unfolded molecules | High | **Y** |

Table 2.2: Functional requirements for remove existing bugs.

## 2.3 Non-functional requirements

WebMGA 2.0 is primarily designed for educators and researchers working in the field of liquid crystal research and it is essential to meets their needs. Given that many users will likely be running the WebMGA on their personal laptops which has limited GPU and CPU power, it is crucial that the performance is optimised to ensure fast loading times and smooth operation.

The following table provides non functional requirements for WebMGA 2.0.

| **Requirement** | **Achieved** |
|---|---|
| Load every model in the library in less than 0.25 seconds | Y |
| Ability to render a system with $10^5$ molecules with satisfactory FPS | Y |
| Smooth real time rendering of visualisation videos of a system size of $10^4$ at at least 6 fps | Y |
| Reasonable time spend under 30 minutes to produce visualisation video consists of 1000 configuration files | Y |
| Easy navigation from main page to each functionalities | Y |
| Clear and intuitively labelled user interface | Y |

Table 2.3: Non-functional requirements.

Use case for researchers for new features of WebMGA 2.0 is shown below:

| **Section** | **Use case** |
|---|---|
| Slicing | Specify clipping plane location to view the inside of a system |
| | Toggle clipping plane helpers |
| PBC | Toggle to show the unit box |
| | Toggle to fold a molecule |
| | Toggle to unfold a molecule |
| Video | Toggle to visualise a set of molecular motion configurations that is generated in molecular dynamics program |
| | Set desired light, camera position, model configurations |
| | Apply periodic boundary conditions |
| | Input file names |
| | Set desired FPS |

Table 2.4: Use case of side menu for researchers.

# Chapter 3

# Scene optimisation

## 3.1 Rendering pipeline optimisation

### 3.1.1 Geometry instancing

It is a well known fact that for a scene rendered consisting of a vast quantity of small objects, the processing time on the CPU for the draw call is greater than the amount of time the GPU takes to actually draw the mesh, so the performance is entirely CPU-bounded [5]. The GPU is constantly starved waiting for new instructions from the CPU. This is proved by Wloka [12] in 2003 that in a extreme case where only two triangles are sent per draw call, the maximum number of triangles that can be rendered is only $10^5$, which falls short by a factor of 1500 compared to the maximum throughput possible with a 1GHz Intel Pentium III processor.

Simulation of liquid crystal systems consists of rendering a system with a large number of objects with the same geometry and material but are arranged in various positions, colours and orientations.

WebMGA1.0's current rendering pipeline requires sending one draw call per number of objects in the system to the GPU. This can be as much as $10^5$ draw calls per frame. As discussed before, this method creates an excessive flow on the CPU while GPU is starved.

The core concept of optimising this problem in computer graphic and simulation is to reduce the number of draw calls and increase the amount of triangles sent in one call, so that GPU is fully utilised. Ideally when rendering is required, a GPU should not be waiting at all and should always have instructions to work on. One approach to achieve this is through geometry instancing [1].

WebMGA 2.0 implements geometry instancing using InstanceMesh class [11] from Three.js. InstanceMesh takes the base geometry of a set and the number of instance required, then sends a single draw call to the

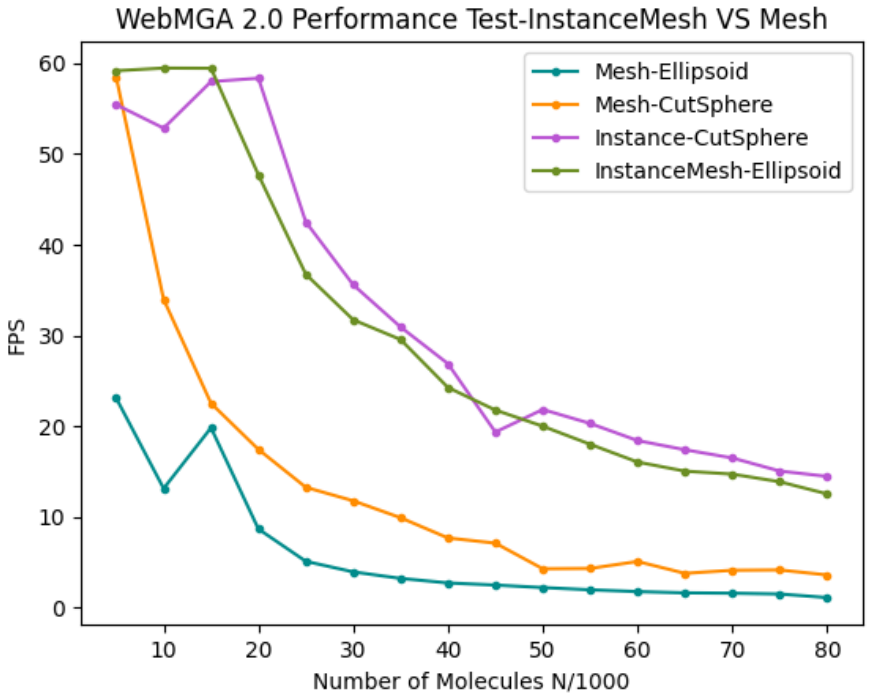


Figure 3.1: WebMGA2.0 performance test

GPU where it draws the instance multiple times. This methods reduces the unnecessary CPU runtime cost and maximise the GPU usage.

The challenge is to understand the colour coding scheme and orientation calculating methods used by WebMGA 1.0 and apply them into a world transforming matrix to assign correct value to each instances. Here is a interesting shape generated accidentally in this process: Figure 10.7.

Figure 3.1 is a performance test conducted on a machine of Apple M1 chip,a 8-core CPU with 4 performance and 4 efficiency core and a 8-core GPU. As we can see, geometry instancing significantly improved the performance of WebMGA 2.0. With the use of ordinary mesh, WebMGA 2.0 can render a system of maximum 20,000 ellipsoids or cut-spheres, with a satisfactory frame rate of 10 to 18.

### 3.1.2 Culling

Culling is a optimising strategy that selectively removing objects or portions of objects from rendering. QMGA implements several culling techniques to aid its rendering efficiency.

**Backface culling:** Backface culling is a technique to discard the object surface that has its normal pointing away from the viewpoint, thereby reducing the number of triangles rendered. In Three.js, the option to render either front, back, or both faces can be specified during creation of mesh materials. WebMGA2.0 has backface culling deployed as a default by only render the front side of a model, except when slicing functionality is enabled.

**Frustum culling:** Frustum culling discards rendering of the object that are outside the camera's field of view. Implementation of this method requires comparing bounding box of each particles to the camera view. Frustum culling is available for ordinary mesh objects, however it is not readily possible with InstanceMesh since the position and orientation of a single instance is assigned after the instance is created.

**Occlusion culling:** Occlusion culling discards an object that is occluded by other objects and is not visible from the view point.[2] Similar to frustum culling, the computation of occlusion culling also requires knowing the exact location of bounding box of objects. The most commonly used method to implement occlusion culling, namely occlusion queries from WebGL 2.0, involves assessing whether a bounding box is occluded by requesting a GPU-based visibility test on triangles. While this approach can reduce redundant rendering of occluded objects, it may leads to an increase in draw calls and introduces additional overheads.

Due to the time constrains and the fact that the benefits of occlusion culling may not be as significant in scenes that are already optimised for instance placementas it introduces additional overheads of implementing occlusion culling, WebMGA 2.0 does not implement frustum nor occlusion culling. However with InstanceMesh and backface culling, WebMGA 2.0 reaches its performance goal.

**Possible algorithms for future development** Since the size of a bounding box is known, it is possible to segments bounding box into several small unit boxes then perform frustum culling and occlusion culling on those boxes in stead. If a box is culled, remove all the positions that lie within the box and discards rendering of such instance.

## 3.2 Application and memory optimisation

To optimise the performance of WebMGA 2.0, it is crucial to thoroughly understand the application level workflows of its functionality and prevent unnecessary change of state. In particular, an analysis of WebMGA 1.0's algorithm for rendering a new sample configuration Algorithm 1 has revealed several errors that must be addressed to improve overall performance.

**Algorithm 1** WebMGA 1.0 model rendering algorithm

```
if loadSample() then
    RemoveAllMeshFromScene();
    for Set in Molecule.sets do
        Set.generateGeometry();
        for Object in Set do
            Object.generateOrientationAndColors();
            Object.generateMesh();
        end for
        if this.state! == NaN then
            View.SetState();
            RenderScene();
        else
            View.SetDefaultState()
            RenderScene();
        end if
    end for
    RenderScene();
end if
```

**Redundant rendering call** RenderScene() function is called multiple times during rendering by functions View.setCamera under View.set.state, which is unnecessary. Redundant function calls occurred in various parts of the WebMGA 1.0's codebase. In addition to analysing workflow, one helpful technique used is to add a line of code that prints alert messages, such as console.log (), inside all critical functions in the pipeline. This helps in determining whether a unintended function is being called multiple times.

WebMGA 2.0 successfully minimised redundancy by removing all irrelevant function calls executed during runtime.

**Memory leaks** As shown in Algorithm 1 WebMGA 1.0 removes all meshes from the scene before rendering new systems, but none of them are disposed. As a result, geometry and material information of these meshes are continually stored in the memory even though they will never be used again.

Not disposing meshes properly before rendering new systems can result in a serious memory leak issue, especially when the target molecule system is of a large size. When creating a video for a system with over 2,000 particles using WebMGA 1.0, rendering of a single frame requires an average of 85 MB memory usage. As the memory usage increases each time a frame is rendered, it will eventually leads the program to halt due to the insufficient usable memory.

WebMGA 2.0 added functions:
$mesh.material.dispose()$,$mesh.geometry.dispose()$
and $mesh.dispose()$ to completely dispose an outdated mesh [10] and free up its memory location. Currently, WebMGA 2.0 maintains a average memory usage of only 42.5MB while rendering a video for 2000+ sized system. The memory usage monitoring is enabled through Chrome developer mode's memory inspect tool.

As a result, WebMGA 2.0 is capable of rendering a system of size $10^4$ with an average fps of 13.136 after CPU and memory optimisation, without InstanceMesh. This is two times faster than its predecessor see Table 8.1 for more detail.

# Chapter 4

# Slicing

When a molecule system is densely packed, it is difficult to observe the inner side. Nevertheless, QMGA utilises the slicing functionality to enable the examination of the inner dynamics and features of a system. QMGA developed slicing where it considers the centre of a object and once a clipping plane bypass its centre, the whole object is removed. After discussions with Professor Germano, an agreement was reached that the slicing functionality in WebMGA 2.0 would only remove a portion of an object. In future work, an option to remove whole objects with slicing could be included as an additional feature for users who require it.

WebMGA 1.0 implemented the slicing functionality using Three.js default clipping planes with slicers, which discard pixels of the assigned meshes that have negative distances to the clipping planes. A limitation of this implementation is that sliced objects appears to be hollow: Figure 4.1. WebMGA 2.0 aims to overcome the limitation by rendering solid looking sliced surfaces, providing a more comprehensive visual representation of molecular systems.

## 4.1 Development

### 4.1.1 Identification of issues

The initial step is determining the cause of the hollows. After conducting research on the Three.js documentation and examining the related code, it has become clear that the reason for the hollow sliced surface in WebMGA 1.0 is due to the default backface culling which only renders the front sides of meshes and omits the back faces. When an object is sliced, it creates an opening and the interior of the mesh is visible. These interior surfaces are supposed to be rendered with backside of a material but with backface culling applied, the interior surface is not rendered so causing the hollows.

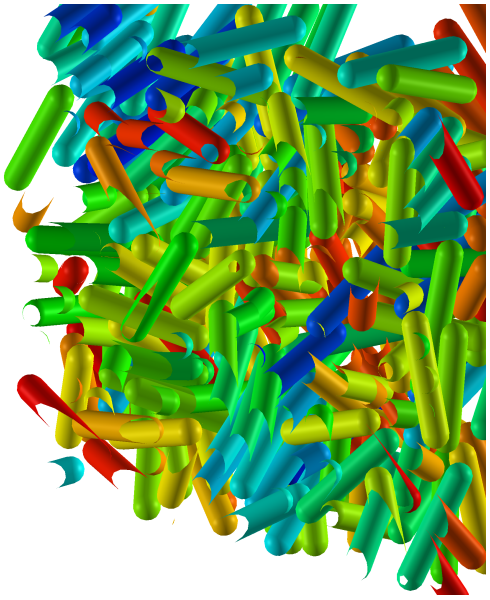

Figure 4.1: WebMGA 1.0 fragmented slicing section of SC4 nematic molecule.

### 4.1.2 Methods

At present, there are no readily available methods for generating solid-like sliced surfaces. However, two possible work-around methods are identified: constructive solid geometry and simulating with the back side method.

The constructive solid geometry approach is to dynamically generate a cover plane that covers the opening of the sliced mesh. To implement this method, every pixels in the scene will be checked against a stencil test to generate geometries for new cover planes to be rendered every time the slicer moves. Since the slicer is updated every 0.1 steps, if a user move slicer from range 50 to 30, such operation will send 200 draw calls to GPU per objects in the molecule system. It is a fact that communicating with GPU can be very expensive, so this method reduces overall performance.

The backside method simulates appearance of the sliced surface of the mesh on its back side. It only sends one draw call to the GPU to render all backsides as one InstanceMesh object.

### 4.1.3 Implementation

Considering both methods produce similar results and backsides method executes in a faster rendering speed, WebMGA 2.0 has addressed this issue by implementing backside method. The front sides of meshes are

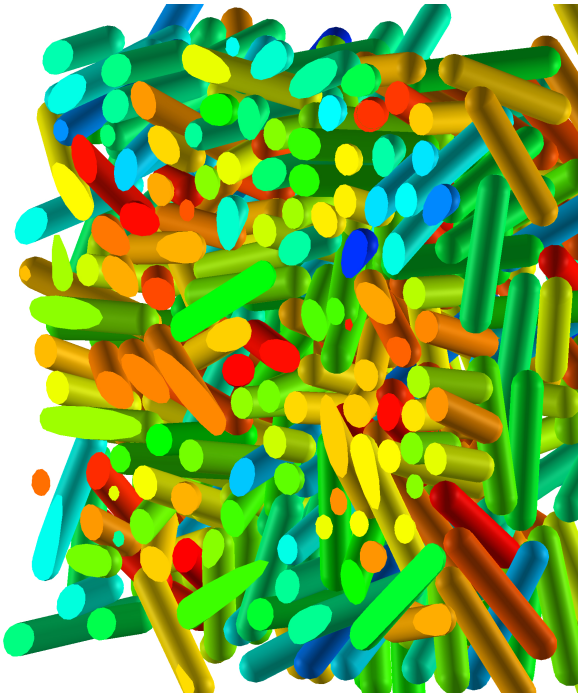

Figure 4.2: WebMGA 2.0 Improved slicing section unfolded SC4 nematic molecule.

rendered with MeshPhongMaterial — a material for shiny surfaces with specular highlights. Rendering back sides with this material cannot replicate the appearance of a flat surface when a object is sliced open, as the reflection of light presents it as the inside of the object instead of a level surface. Instead, Three.MeshBasicMaterial is chosen to render backsides of objects. This material is not affected by light — it does not reflect nor refract light, which allows the back side of the mesh to simulate a flat surface even though it is actually the interior of the object. In addition, since the back sides of meshes are not visible from usual views, they are not rendered until user enables slicing functionality. This is to improve the performance by enabling back face culling and reducing unnecessary rendering.

WebMGA 2.0 has achieved a realistic simulation of solid sliced molecules using this approach. Figure 4.2 is a slicing example.

Due to the nature of the clipping planes being assigned to meshes when meshes are generated, slicing is always enabled in its default range in the background. When implementing periodic boundary conditions, the clipping plane sometimes remove a part of the large unfolded molecules by accident. WebMGA 2.0 improved upon this by doubling the range of slicers and clipping plane for which most of unfolded molecules lie within.

## 4.2 Performance

To test the performance of the slicing feature, comparisons were conducted on a wider range of molecules with different shapes. For instance, the O5 nemantic (Figure 4.4) is visualised using ellipsoid shapes, while the SC4 nematic (Figure 4.2 )is visualised using spherocylinders.

There exists systems that comprises more than one type of molecule,

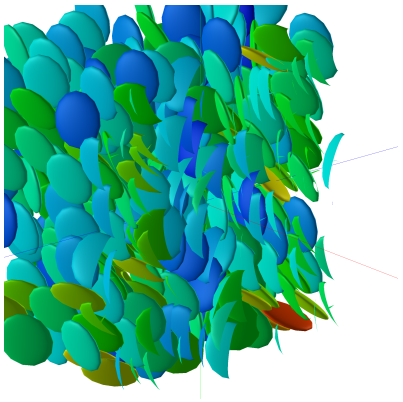

Figure 4.3: WebMGA 1.0 slicing of O5 Nematic.

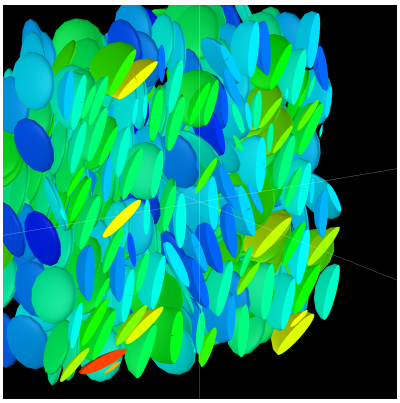

Figure 4.4: WebMGA 2.0 slicing of O5 Nematic.

and it is crucial to verify that slicing is functional for all molecules. A biaxial crystal comprises three sets of different molecules. Below shows the slicing of WebMGA 2.0 applied on a biaxial crystal and confirmed to work on all three sets of molecules.

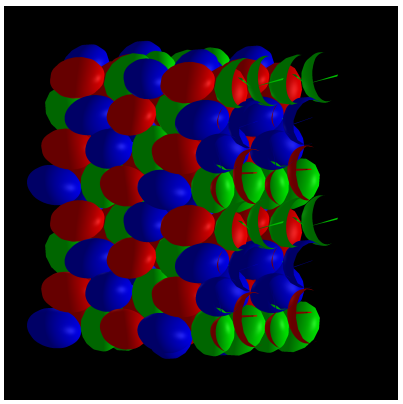

Figure 4.5: WebMGA 1.0 - slicing of biaxial crystal.

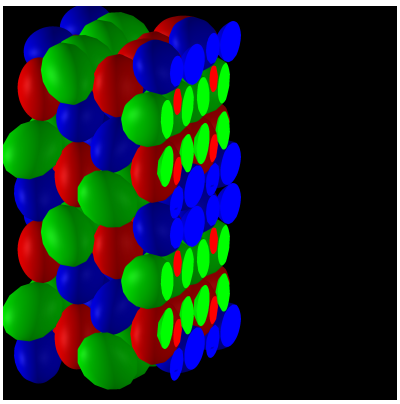

Figure 4.6: WebMGA 2.0 - slicing of biaxial crystal.

Overall, the slicing functionality guarantees good results on all of the model shapes and with multiple molecule sets.

# Chapter 5

# Periodic boundary conditions

## 5.1 Background

In a molecular dynamics simulation, when simulating a very small system it is possible that the cohesive forces between the molecules are strong enough to hold the system together throughout the simulation. Alternatively, molecules can be confined in a container, preventing them from drifting apart. This method however introduces surface effects to bulk liquids due to the fact that large fraction of molecules lie on surface of a container and experiences different force than those in bulks. This is typically considered an undesired effect.

Periodic boundary condition is the important technique in a molecular dynamics simulation to remove surface effect and simulate bulk liquids. For short-ranged interactions and away from phase transitions, PBC have

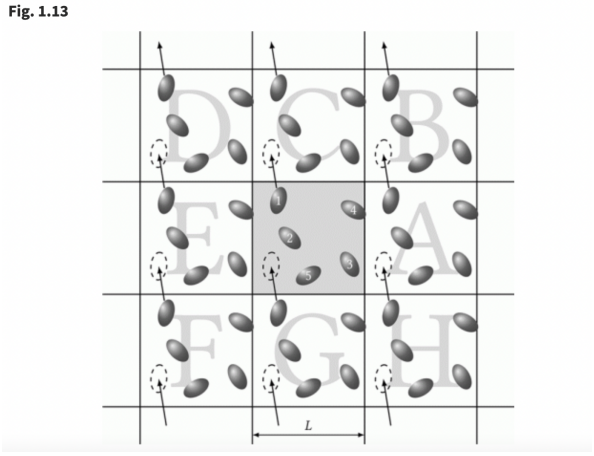


Figure 5.1: 2D PBC example [4]

little effect on the thermodynamic properties and structures of fluids.

The technique of periodic boundary conditions involves replicating a cubic simulation box infinitely in all directions to create an infinite lattice. During the simulation, the movement of a molecule within the central box is mirrored by the movement of its periodic images in the neighbouring boxes. Consequently, when a molecule exits the central box through one face, a periodic image of it enters through the opposite face. The boundary of the central box is not physically restricted by walls, and there are no surface molecules at the boundary.

Periodic boundary conditions is one of the key features of QMGA, allowing display of the molecules bounded insides its unit box, without surface effect. WebMGA 2.0 will focus on simulating the dynamics of a liquid system with periodic boundary applied within a single unit box.

## 5.2 Development

### 5.2.1 Re-implement unit box methods

In the previous version of WebMGA, the size of unit box of each molecules is generated dynamically in the program by calculating the bound of its input geometries, using the Three.js library function

*mesh.geometry.computeBoundingBox()*.

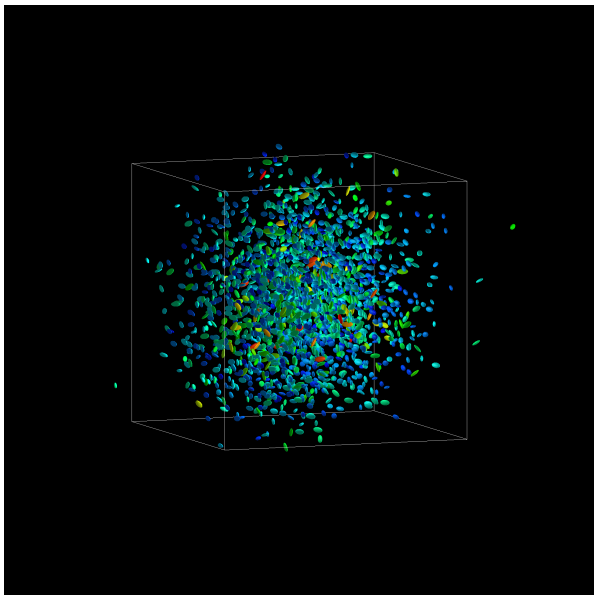

Figure 5.2: WebMGA 1.0 - incorrect unit box

Such method is valid only if assuming the input molecule is always in a folded state. When rendering a unfolded molecule, this method errors and produce a box larger than unit box as Figure 5.2 shows. Since implementation of periodic boundary conditions relies on the size of the unit box, it is crucial to have the correct information.

WebMGA 2.0 addresses this issue by enabling users to specify the size of the unit box in configuration files and generate unit box accordingly. Figure 5.5 is a example of correct unit box size for a unfolded system. Typically, the size of the unit box is obtained on the first line of configurations from molecular dynamics programs such as LAMMPS. As a result, this change should not pose a significant burden on users.

Here is the new configuration format with *unitBox* in addition.

```
"model": {
        "sets": [
            {
                "name": "Set A",
                "orientationType": "q",
                "unitBox":[
                    [1,1,1]
                ],
                "positions": [
                    [0,0,0]
                ],
                "orientations": [
                   [0,0,0]
                ]
            }
        ]
    }
```

### 5.2.2 Pseudo configurations of unfolded molecules

Periodic boundary condition involves folding a molecule into its bounding box. Testing of this functionality requires configurations of a unfolded system, which is not readily available in WebMGA 1.0. To obtain such configurations, one can either generate a fresh set using molecular dynamics programs or generate Pseudo unfolded system according to a folded one. Considering the time cost to master a complex program like LAMMPS[7], the later method is chosen. Pseudo unfolded positions are generated by adding a random portion(1 to -1) of length of the unit box:

```
this.positions =
    this.positions(Math.random() * (2) -1)*
    this.unitBox.length
```

Pseudo samples may cause unwanted overlaps of the particles after periodic boundary applies. Nevertheless, It is sufficient to give a brief visual idea of this physic concept for student and educators, who may not to have a configuration. Pseudo unfolded samples of SC4 nematic and E3 chiral nematic are added to the sample Library for their use.

### 5.2.3 Implementation

When user toggles the 'Fold' option, WebMGA-2 performs a test checking the positions of all objects in the target molecule against its unit box position. If there exists some object which its position lies outside the unit box range, the molecule is said to be unfolded and vise-versa.

If a molecule system is unfolded, we can then apply periodic boundary condition to it. WebMGA 2.0 apples minimum image convention while calculating a new positions for a particle that moved outside of unit box. The follow code is used to wrap the coordinates of a molecule back into the central box by subtracting an integer number of box lengths from its position [4]:

```
this.positions = this.positions-this.unitBox.length *
Math.round(this.positions/this.unitBox.length);
```

Periodic boundary conditions play a crucial role in simulating the Brownian motion of molecules and generating videos. To allow application of PBC to video, the fold state is checked when a system is initially generated. If a setting of apply PBC in View class is set to true and the molecule is unfolded, PBC is automatically applied to it before rendering to screen.

WebMGA 2.0 achieved accurate calculation and rendering of PBC applied liquid system.

## 5.3 Performance

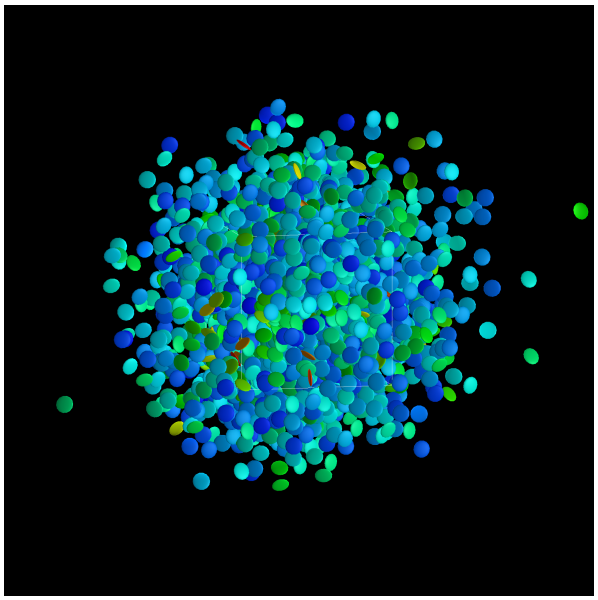

Figure 5.3: Unfolded soft discotic ellipsoids.

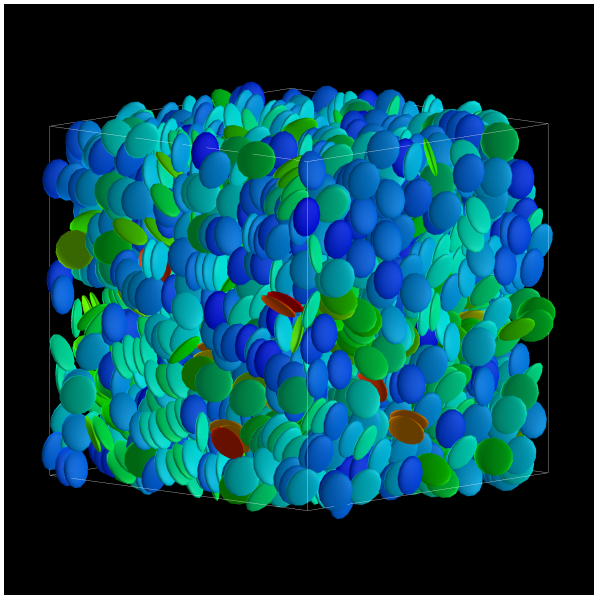

Figure 5.4: Folded soft discotic ellipsoids.

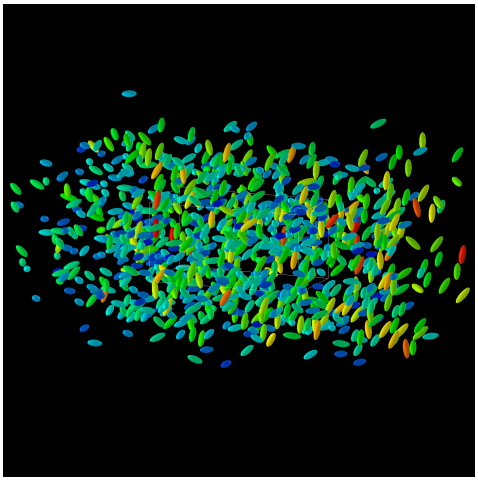

Figure 5.5: Dummy unfolded E3 chiral nematic.

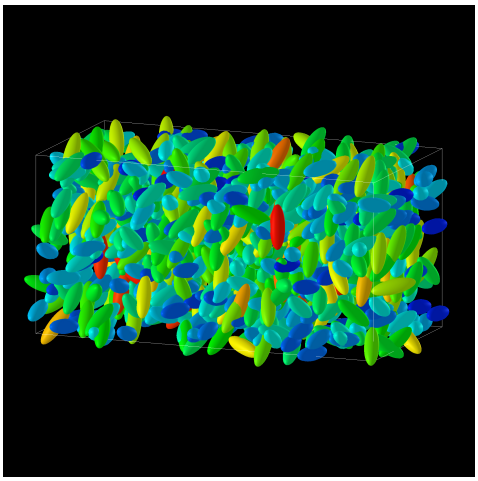

Figure 5.6: Folded dummy E3 chiral nematic.

A unfolded configuration of soft discotic ellipsoids interacting with the Gay-Berne potential with elongation parameter $\kappa = 0.2$ was generated from GBMEGA later in the process. Application of periodic boundary conditions has been tested on both dummy unfolded configurations and unfolded configurations obtained from GBMEGA, which is a parallel simulator for molecular dynamics.

Furthermore, a video visualisation of Brownian motion with PBC applied is generated to show case the successful application of PBC :https://www.youtube.com/watch?v=F225-2x-ybg.

# Chapter 6

# Video

Brownian motion is the stochastic motion of particles caused by random collisions with other particles in gas or liquid medium. The ability to visualise Brownian motion can be highly advantageous for researchers and educators.

## 6.1 Development

WebMGA 2.0 generates the animation of molecule motions by loading files sequentially and render them one after the other to create motion picture. When dealing with large and complex liquid system, the load and render times become long and resulting in a stagnant animation. In QMGA, saving all frames to disk was possible, but the conversion of frames to a video file required help from external programs. WebMGA 2.0 has the ability to create a non-stagnant video, regardless of the actual time required to render a single frame. In addition, an option to alternate frame rate to adjust the speed of the video is includes in WebMGA 2.0, provides users with greater flexibility and control over the rendering process.

The video functionality significantly increase the importance of WebMGA in the field as it aids the visualisation of complex molecular dynamics such as Brownian motion, which are often difficult to grasp through static images or data alone. Here is a link to a sample video created by WebMGA 2.0 simulating Brownian motion of a system with 2000 molecules: https://www.youtube.com/watch?v=zX8BXLG4GvA.

As a graphic visualisation application, WebMGA 2.0 is not designed to generate configurations for any molecular motions. These configurations are expected to be generated from molecular dynamics programs such as LAMMPS [7]. Note that input files are expected to be formatted according to WebMGA configuration style.

One of the challenges encountered during the development of video functionality is to work around the existing codebase, as it needs to interact with many other components such as functions to generate new models. Another key part is the rendering speed. WebMGA 2.0 aims to achieve real-time rendering of video seamlessly on screen and reduce the time required to generate and download high quality visualisation videos. Below is a detailed implementation plan.

### 6.1.1 Load files

The first step is to allow users to load multiple files. The previous upload button is designed to load a single configuration and is located at the top of main menu. To utilise usability, a new load function is implemented allowing multiple file uploads and its located at Video side menu providing easy access. Loaded configurations will be stored in the system allowing repeated generation of video from same files. Unclicking the load button will free the files from memory resource and allow loading new inputs.

While it can be convenient to include a pre-built Brownian motion configuration library for demonstration purposes, it should be noted that the sample used to generate below videos consist of 10,000 JavaScript files. Including them in program leads to a decrease in the overall execution speed of the program, resulting in performance issues and lags. Alternatively, a link is provided to access these samples.

### 6.1.2 Create animation

After files are loaded, each models will be rendered on the canvas in sequence, providing a vivid visualisation of molecular motions. This is allow by creating a animation loop utilising modern browser technologies such as requestAnimationFrame. The loading and rendering of models take up most of the time and computational power. During the initial development stage where no optimisation was applied, the program was able to render the configurations correctly, but it encountered a memory overflow issue after rendering only 102 files. At the same time its rendering speed for a system of 2000 molecules was 0.20 fps, far slower than desire speed of 20 fps. To address these issues, cares were taken to dispose all objects from previous frame to free the memory and several scene optimisation strategy was adapted to improve the fps. For more details see Chapter 3.

```
RealTimeVideo(i,samples,max_iter,capturer,vidState,filename){
    if(i ===0){
        capturer.start();
        capturer.capture(this.model.renderer.domElement);
    }
    if(i<max_iter){
        this.functions[1].bind(this)(samples[i],i,vidState);
        capturer.capture( this.model.renderer.domElement )
        console.log('running animation',i)
        if(this.state.video === true ){
            requestAnimationFrame( ()=> this.RealTimeVideo(i+1,samples,max_iter,capturer,vidState,filename));
        };
    }
    if (i === max_iter){
            capturer.stop();
            capturer.save(function( blob ) {
                console.log(blob);
                var url = URL.createObjectURL(blob);
                var link = document.createElement('a');
                link.href = url;
                console.log(filename)
                link.download = filename + '.webm';
                document.body.appendChild(link);
                link.click();
                document.body.removeChild(link);
            });
            View.state.reference.video =false;
            View.state.reference.setVideoState = false;

    }}
```

Figure 6.1: Snapshot of animation loop.

### 6.1.3 Set environment

WebMGA 2.0 provides numerous customise options for video generation, including environmental settings such as periodic boundary conditions, changing model shapes, altering light positions and colours. These features provide users with more flexibility to create unique and informative visualisations of their simulations in different shaped molecules and settings. User can also specify speed and file name for the video. This functionality is implemented utilising the View settings. When the program displays the first model on the screen, users can preview the environment settings and apply them to the model. If an environment is set, the View object that stores all the settings is passed along to the next model to be rendered, ensuring that the video maintains a coherent set of settings throughout.

Modifying several settings can slow down the rendering speed because of the current rendering pipeline which a model is generated first, and the view settings are applied to it afterwards. If modifications are made to the model shapes or PBC, for example, this can result in the model being rendered twice, once for the initial generation and once again with the modified settings. Solutions this issue requires adapting a more efficient application rendering hierarchy. This could involve a fundamental overhaul of the program so is not carried out. However the usage of InstanceMesh has reduced the impact of this redundancy.

### 6.1.4 Video production

A few options are available to allow capturing and downloading of the animation rendered to canvas . The first option is to use the a generic API that simply records a media stream created in HTML such as MediaStream recording API [9].

Another option is to utilise the ccapture.js library [8] which is designed to capture canvas-based animations at a fixed frame rate. WebMGA 2.0 implements this library because of its extended functionalities. Ccapture.js instead of record the entire duration of the rendering video, It captures the canvas after each frame is rendered. This is enabled in the animation call. After all frames are recorded, it generates a video with user specified fps value.

## 6.2 Performance

Overall the performance of video functionality meets the needs for rendering a typical liquid crystal system. After applying optimisation strategy, WebMGA 2.0 achieved a desirable fps of 20-25 throughout the rendering of 10,000 frames for a system consists of 2000 objects. This allows seamless real time viewing of the video for most system sizes. To produce a video for such system, an average rendering time is 21 minutes.

### 6.2.1 Video presentations

The configuration files used to produce video samples are soft discotic ellipsoids interacting with the Gay-Berne potential with elongation parameter $\kappa = 0.2$ generated from GBMEGA. Theses are provided by Professor Germano.
Here are sample videos produced by WebMGA 2.0 for such system:
Visualisation of Brownian motion in a system without periodic boundary condition:
https://www.youtube.com/watch?v=zX8BXLG4GvA.
Visualisation of Brownian motion in a system with periodic boundary condition applied:
https://www.youtube.com/watch?v=F225-2x-ybg.
Visualisation of Brownian motion in a system with applying a different model shape(spheroplatelet).
https://www.youtube.com/watch?v=P5v1AHtxpm8.

# Chapter 7

# Polishing and debugging

Addressing inconsistencies and improving usability are critical factors for enhancing the professional usability of WebMGA 2.0 in scientific contexts. The most challenging part of this part of implementation is to navigate thought the large and complex codebase and investigate the cause of issues.

## 7.1 Numerical input box

The numerical input box serves as a vital component of the front-end interface as it facilitate the user input of configurations. However it inherits an error: If a user manually types into the input box instead of using the provided steps, an error occurs causing the input box to malfunction and the model to be removed from the canvas. Looking into the implementation of numerical input box, it has been identified that this issues is been caused by updating model while configuration input is NaN.
**Solution:** A value validation is performed before updating model and if an input received from front end appears to be NaN or any other invalid form, this input will be set to zero. This solution prevents model disappearing from screen.

## 7.2 Side-menu malfunctions

When user loads a new model, the side menu fails to update and shows data of last model.
**Solution**: Function to automatically close and update the sidebar is added when loading new model.

## 7.3 Inconsistent background colour

Upon inspection of the model library, it was observed that certain models had a white background while others had a black background, leading to inconsistencies in the appearance of the models.
**Solution**: An option to toggle dark background is added to the side menu. The default is set to having dark background and users can manually switch to white.

# Chapter 8

# Performance

## 8.1 Rendering speed

One of the key strength of QMGA is its ability to seamlessly render a system up to $10^5$ molecules. Although Java script is known to be slower than C++, WebMGA 2.0 made significant improvement in its rendering methods and application design to increase the rendering speed, to the same level of QMGA.

Performance test is conducted by adding random particles to the scene and rendering it while auto-rotating. The mean fps values for each level of the test are used as indicator. Both performance tests for WebMGA 2.0 and 1.0 are conducted on a machine of Apple M1 chip, a 8-core CPU with 4 performance and 4 efficiency core and a 8-core GPU. To simulate a possible scenario where the user's computer may have other tasks running while running this program, tests were performed with multiple browser pages open in the background.

### 8.1.1 Comparison to QMGA

Below is a comparison of the rendering speed of QMGA, WebMGA 1.0 and WebMGA 2.0(both mesh rendering and InstanceMesh rendering methods). Both WebMGA 1.0 and 2.0 are tested with random ellipsoids models with size of 1,1,2 and material of MeshPhongMaterial. QMGA testing results for vertex buffer objects while occlusion culling applied are extracted from its original thesis.

The result shows that WebMGA 2.0 using the mesh rendering method shows some improvement compared to WebMGA 1.0, but it still falls short in performance compared to QMGA.

WebMGA 2.0, using the instance mesh rendering method however, achieved even slightly better performance than QMGA for system of size between 5,000 and 50,000. For larger values of N between 70,000 and

| N/1000 | WebMGA 2.0–Mesh | WebMGA 2.0–IM | QMGA VBO+OC | WebMGA 1.0 |
|---|---|---|---|---|
| 5 | 23.081 | 59.145 | 50.300 | 18.754 |
| 10 | 13.136 | 59.435 | 38.000 | 9.050 |
| 20 | 8.673 | 47.674 | 27.400 | 1.428 |
| 50 | 1.980 | 19.995 | 17.400 | 0.026 |
| 70 | 1.123 | 13.877 | 14.200 | N/A |
| 100 | 1.169 | 9.943 | 11.800 | N/A |
| 140 | 1.093 | 7.357 | 9.300 | N/A |

Table 8.1: Frames per second for a system of $N$ discotic ellipsoids rendered in WebMGA 2.0 QMGA (Figure 10.5) with vertex buffer objects and occlusion culling and WebMGA 1.0.

140,000, WebMGA 2.0 falls slightly behind QMGA, with a difference of only roughly 2 fps. This outcome is highly satisfactory as it demonstrates a significant improvement from WebMGA 1.0. In practice, WebMGA 2.0 loads every model in the library instantly, while WebMGA 1.0 takes over 2 seconds to load a larger system such as E3 chiral nemantic.

A more detailed comparison between rendering with Instance mesh and mesh methods can be found at Figure 3.1.

### 8.1.2 Shapes

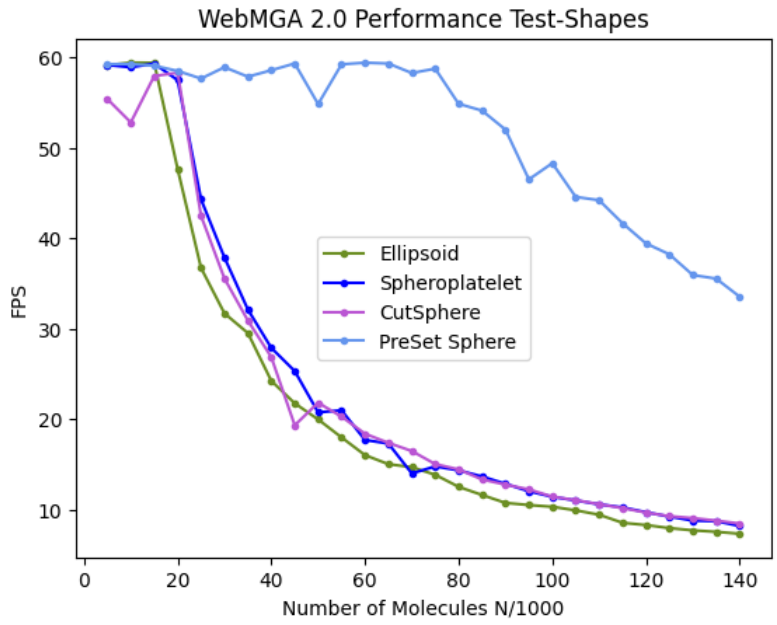


Figure 8.1: WebMGA 2.0 performance test for each models.

Tests are carried out for each models and the results show that the rendering speeds for ellipsoid, spherocylinder, and spheroplatelet are in a similar range. This suggests that no specific model significantly impacts the rendering performance.

Note that although rendering with pre-set shapes in Three.js such as sphere may achieve better results, the majority of simulations of liquid crystal system use shapes of ellipsoid spherocylinder, spheroplatelet or cut sphere. Therefore, most weight should be given to those commonly used shapes when evaluating the performance.

The performance gaps between QMGA shapes and pre-set shapes suggest that the complex geometry of QMGA shapes results in longer rendering times. To further improve the rendering speed of WebMGA 2.0 in the future, it is necessary to investigate and optimise the geometry calculation process. One potential solution could be to use a combination of Three.js pre-set shapes with constructive solid geometry method to represent desired shapes, instead of calculating the geometry.

According to Battistini (2021) [6], the different levels of details and materials shows no significant impact on the rendering speed (Figure 10.4).

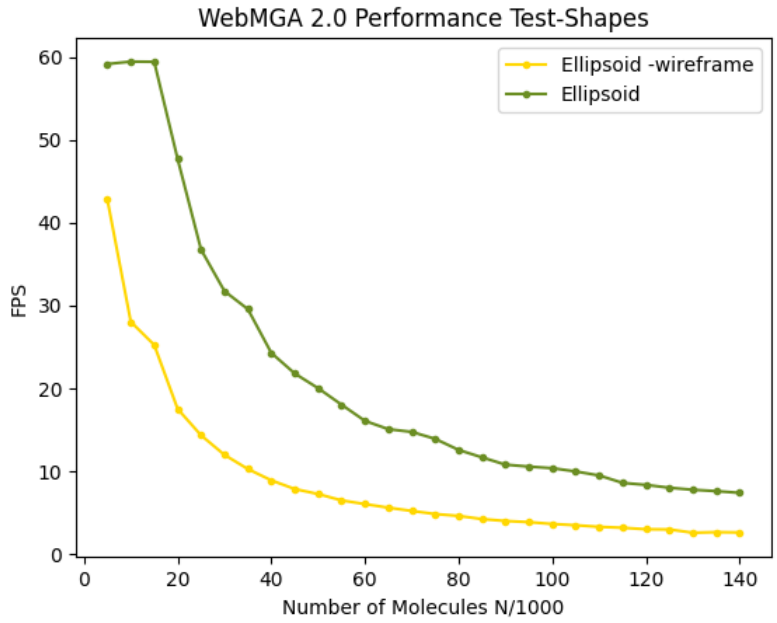


Figure 8.2: WebMGA 2.0 performance test on wireframe.

However, as shown in Figure 8.2 the rendering of wire-frame objects is slower. Upon closer inspection, it is evident that the rendering speed of wire-frames is twice that of the normal model. This issue has not

been addressed yet due to time constraints, but the cause of it has been identified : when the user toggles the wire-frame option, the program first generates the normal model and then the wire-frame so the process is ran twice. The same problem also happens to any other user modifications through the View class, such as changing to a different model. This is an inherited bug and should be addressed in the future to provide smoother rendering of user specified models.

## 8.2 Rendering accuracy

WebMGA 2.0 introduced Instance mesh rendering method which requires a different way of assigning positions, orientations and colouring to each molecules. WebMGA 2.0 applied correct transformation of its colouring mapping to instance meshes and maintained high rendering accuracy.

To test the rendering accuracy, comparisons are made with QMGA and WebMGA 1.0.

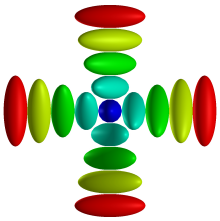

Figure 8.3: WebMGA 2.0 - dummy configuration.

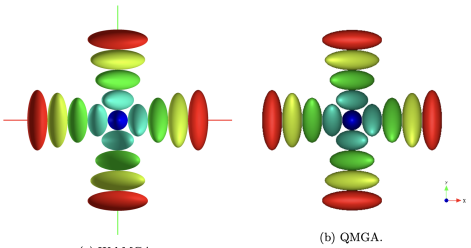


Figure 8.4: Comparison of dummy configuration [6].

# Chapter 9

# Evaluation and conclusion

## 9.1 Achievements

Overall, WebMGA 2.0 provides a reliable tool for visualising molecular systems in liquid crystal research. Its rendering accuracy in terms of molecule positioning, coloration and orientation is highly precise, making it a suitable tool for professional applications. Furthermore, WebMGA 2.0 operates at impressive rendering speeds, providing seamless rendering of systems up to $10^5$ with succulent FPS. Providing additional robust visualising functionalities including periodic boundary conditions, slicing and video visualisation, WebMGA 2.0 enables a deeper level of insight into the behaviour and structure of molecular systems. With bug free user interface, WebMGA 2.0 brings amazing user experience.

The primary goal of this project was to complete the feature set of QMGA that has not been delivered in WebMGA 1.0. This goal has been successfully meet and all of the missing functionalities have been implemented and thoroughly tested in WebMGA 2.0.

## 9.2 Critical evaluation

WebMGA 2.0 has been developed to a high standard, but there is still room for improvement, particularly in the areas of rendering accuracy and usability.

### 9.2.1 Future work

- **Integrate frustum culling and occlusion culling with InstanceMesh** Since InstanceMesh improve run time by only sending one draw call, both frustum culling and occlusion culling should be implemented sending as less draw calls as possible. One way of

implementing this is to assign each molecules into a few area and discard all molecules in one area if not seen on screen.

- **Optimise view class** Currently modifying model goes through the view class and any modification causes model to be rendered twice. To improve the performance, this redundancy should be removed.
- **Converting input file format** WebMGA requires a specific input formats which is different from the common output format by molecular dynamics programs like LAMMPS. Coveting between 2 formats can be tedious for non technical researchers, therefore it would be beneficial to allow user to input standard configuration files and converted them inside the program.
- **Responsive user interface** The layouts of side menu and navigation bar should be responsive to the change of screen size. This can be improved by applying bootstrap framework to the front end.
- **Options to pause, replay and stop Video** Currently, video functionality is designed to create and download video, not for real time viewing. Operations for real time viewing such as pausing and replaying can be added once WebMGA achieves higher rendering speed.

# Bibliography


[1] Francesco Carucci. *GPU Gems 2*. 2005, pp. 47–67. URL: https://developer.nvidia.com/gpugems/gpugems2/part-i-geometric-complexity/chapter-3-inside-geometry-instancing.

[2] Dean Sekulic Croteam. *GPU Gems 2,Efficient Occlusion Culling*. 2005, pp. 47–67. URL: https://developer.nvidia.com/gpugems/gpugems/part-v-performance-and-practicalities/chapter-29-efficient-occlusion-culling.

[3] ‡ Adrian T. Gabriel † Timm Meyer and Guido Germano*. "Molecular Graphics of Convex Body Fluids". In: *J. Chem. Theory Comput* 4 (2008), pp. 468–476. DOI: https://doi.org/10.1021/ct700192z.

[4] Dominic J. Tildesley Michael P. Allen. *Computer Simulation of Liquids (2nd edn)*. Oxford University Press, 2017. ISBN: 9780191841439. DOI: https://doi.org/10.1093/oso/9780198803195.001.0001.

[5] Tomas Akenine-Möller et al. *Real-Time Rendering 4th Edition*. Boca Raton, FL, USA: A K Peters/CRC Press, 2018, p. 1200. ISBN: 978-1-13862-700-0.

[6] Eduardo Battistini. "WebMGA - WebGL Molecular Graphics Application for the Interactive Rendering of Coarse-Grained Liquid Crystal Models". In: (2021). URL: http://students.cs.ucl.ac.uk/2019/group3/WebMGA/diss.pdf.

[7] A. P. Thompson et al. "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales". In: *Comp. Phys. Comm.* 271 (2022), p. 108171. DOI: 10.1016/j.cpc.2021.108171.

[8] *CCapture library*. URL: https://github.com/spite/ccapture.js/#contributors.

[9] *MediaStream API*. URL: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream_Recording_API.

[10] Three.js. *How to dispose of objects*. URL: https://threejs.org/docs/#manual/en/introduction/How-to-dispose-of-objects.

[11] Three.js. *InstancedMesh*. URL: https://threejs.org/docs/#api/en/objects/InstancedMesh.

[12] Matthias Wloka. *Batch, Batch, Batch: What Does It Really Mean?* URL: Game%20Developers%20Conference,%20Mar.%202003..

# Chapter 10

# Appendix

## 10.1 More performance tests for QMGA and WebMGA 1.0

***Table 1.*** Frames per Second for a System of *N* Discotic Ellipsoids in a Columnar Phase, Using Display Lists (DL) or Vertex Buffer Objects (VBO) without or with Occlusion Culling (OC)[a]

| *N*/1000 | DL | VBO | DL+OC | VBO+OC |
|---|---|---|---|---|
| 1 | 485.2 | 585.7 | 75.0 | 79.6 |
| 5 | 95.6 | 108.7 | 49.9 | 50.3 |
| 10 | 48.8 | 54.8 | 37.4 | 38.0 |
| 20 | 24.7 | 27.6 | 26.8 | 27.4 |
| 50 | 9.9 | 11.1 | 16.3 | 17.4 |
| 70 | 7.1 | 8.0 | 14.0 | 14.2 |
| 100 | 4.9 | 5.4 | 11.5 | 11.8 |
| 140 | 3.5 | 3.6 | 9.1 | 9.3 |

[a] Errors are below 5%.

Figure 10.1: QMGA Performance Test [3] .

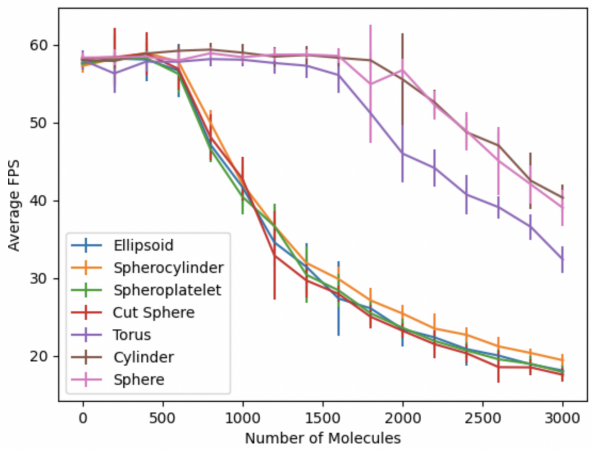


Figure 10.2: WebMGA 1.0 Performance Test with LOD level 3 and Phong Material [6].

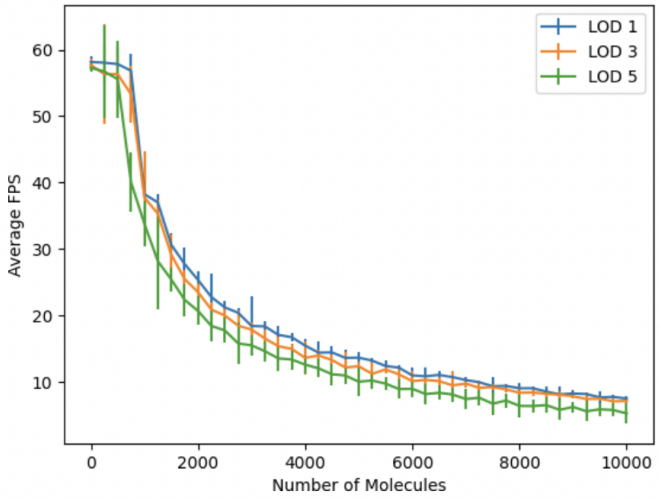


Figure 16: LOD test results, using ellipsoid geometry and Phong material.

Figure 10.3: WebMGA 1.0-LOD test [6].

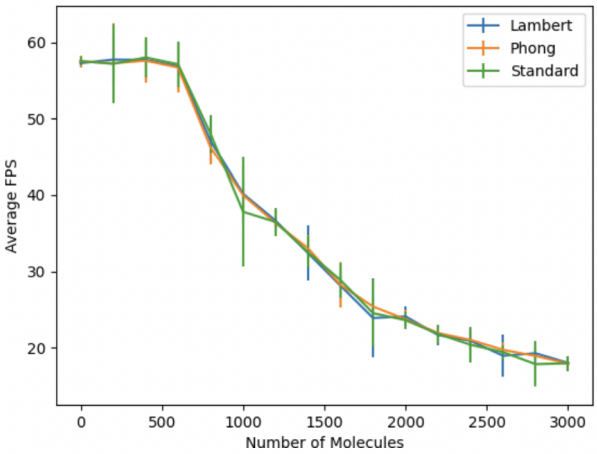


Figure 17: Material performance test results, using ellipsoid geometry and LOD 3.

Figure 10.4: WebMGA 1.0-material test [6].

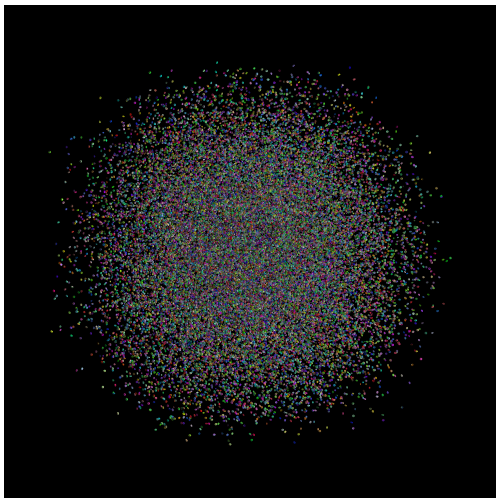

Figure 10.5: a snapshot of performance test when N = 60,000.

## 10.2 Manual

```
 _        __      __     __   ------------
| |      / /--   / /-   /   |/   /  ----/    |
| | /| / /  - \/  -- \/  /|-/ / /  --/ /| |
| |/ |/ /   --/ /-/ / /   / / /-/ / --- |
|--/|--/\---/--·---/-/   /-/\----/-/   |-|
```

by Eduardo Battistini **and** Yue He

---

USER MANUAL

---

Welcome to WebMGA.

Please see the 'About' section of the program to learn more about the purpose of WebMGA **and** the configurations included **in** the library.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

LICENSE

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Copyright 2023. Carlos Eduardo Battistini Parra **and** Yue He.

Permission **is** hereby granted, free of charge, to **any** person obtaining a copy of this software **and** associated documentation files (the "Software"), to deal **in** the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, **and/or** sell copies of the Software, **and** to permit persons to whom the Software **is** furnished to do so, subject to the following conditions:

The above copyright notice **and** this permission notice shall be included **in all** copies **or** substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS␣IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

CITING WEBMGA

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If you use pictures produced with WebMGA **in** a scientific publication, please cite with a text like:

The pictures were produced with WebMGA [1], an evolution of QMGA based on WebGL [2].

[1] Yue He **and** Eduardo Battistini, WebMGA, a WebGL Molecular Graphics Application **for** the Interactive Rendering of Coarse-Grained Liquid Crystal Models, BSc Thesis **in** Computer Science, UCL, 2021,2023 https://astromarx.github.io/WebMGA, https://hiiamyue.github.io/WebMGA-2/

[2] Adrian T. Gabriel, Timm Meyer, Guido Germano, "Molecular graphics of convex-body fluids", Journal of Chemical Theory and Computation 4, 468-476, 2008, DOI 10.1021/ct700192z, http://qmga.sourceforce.net.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                        CONFIGURATION FILE FORMAT
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

To upload a custom configuration, you must generate a JSON file
containing positions, orientations, and sets of molecules in the
following format and upload it.

{
    "model": {
        "sets": [
            {
                "name": "Set A",
                "orientationType": "v",
                "unitBox":[
                    [0,0,0]
                ],
                "positions": [
                    [0,0,0]
                ],
                "orientations": [
                    [0,1,0]
                ]
            }
        ]
    }
}


For information on the JSON format, please see:
https://www.json.org/

If your configuration has more than one molecule type, restrict each
set of molecules to a different object in the "sets" list.

You may name sets as you please.

You must identify which format you are specifying the molecule
orientations with for the "orientationType" with one of the following
identifiers:

        v - Unit vector
        q - Quaternion
        a - Axis angles
        e - Euler angles

Each molecule should have a corresponding position and orientation
list the lists "positions" and "orientations".

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                        ORIENTATION SPECIFICATION
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

Unit Vector: [x,y,z]
Quaternion : [w,x,y,z]
Euler angles: [x,y,z]
Axis angles: [axis_x, axis_y, axis_z, angle]
```

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

.WEBMGA FILES

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

If you save a configuration, the provided model data will be saved in JSON format along with a "view" object, which contains all the viewing parameters at the time the configuration was saved. You may change the parameters manually in the saved file or re-upload this type of file to recreate a model with specified viewing settings.

To see a sample, please select a configuration from the Library and click 'Save'.

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

SUB-MENU OVERVIEW

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

The menu in the header contains information about WebMGA, Loading, Saving, Exporting, and Uploading, as well as the Library of preset configurations and Level of Detail setting.

By increasing the level of detail, more detailed images will be produced at the cost of poorer performance.

In the Sidebar, you will find the following sub-menus:

Model:

For picking a shape for each of the sets in the configurations and their corresponding parameters. Also includes options of colouring manually or by the director and displaying shapes as wire-frames.

Ambient:

For specifying ambient light and background colours.

Lighting:

For specifying positions and colours of 'point' and 'directional' lights. Toggling 'helpers' will display figures that will make positioning lights easier.

Slicing:

For clipping the system in X, Y, or Z dimensions. Also includes 'helpers'.

Reference:

For including axes (which may be multi-coloured), and a bounding box. The size and colour of the axes may be specified. Also toggle fold to fold model into its bounding box.

Video:

For generating video visualisation of molecular motion.
Toggle load to load configuration files at once.
Select desired environmental set ups then toggle set environment to apply to video.
Also choose frame rate and file name before creating video.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                                    CONTACT
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

For help, queries, suggestions or anything else, please contact
zceeceb@ucl.ac.uk, zcabyhe@ucl.ac.uk

================================================================================
                              THANK YOU FOR USING WEBMGA!
================================================================================
```

## 10.3 Takeouts

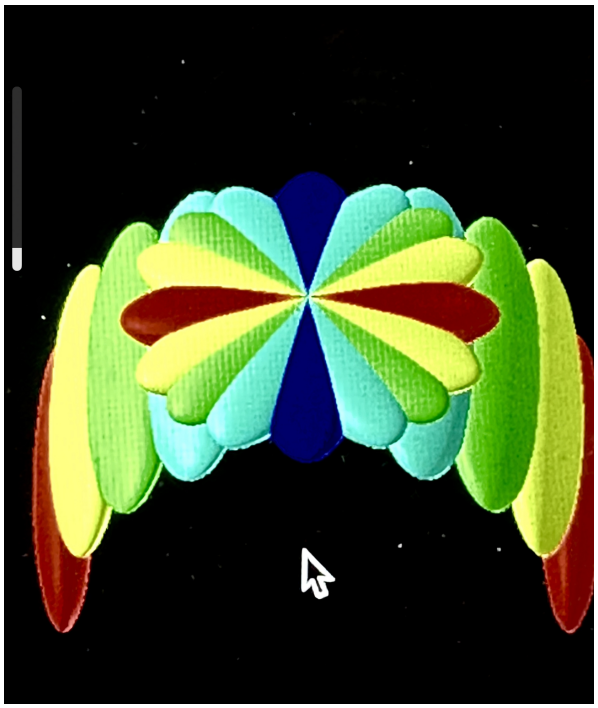

Figure 10.6: Elegant flower shape generated while debugging.

Figure 10.7: Eastern egg shape.