WebMGA

WebGL Molecular Graphics Application for the Interactive Rendering of Coarse-Grained Liquid Crystal Models

Eduardo Battistini*

Computer Science BSc, University College London
Supervisor: Dr. Guido Germano

April 20, 2021

*This report is submitted as part requirement for the Computer Science BSc at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

The WebGL Molecular Graphics Application, or WebMGA, is a web-based visualisation tool for coarse-grained molecular models that utilises prolated and elongated convex bodies as the elementary units of simulation. Given the prevalence of said geometries in the modelling of liquid crystal systems and the lack of available visualisation platforms suitable for this niche, WebMGA provides a unique, out-of-the-box solution for researchers and educators to generate, stylise, and interact with three-dimensional renders of molecular simulations.

WebMGA is written in Javascript, and implements the state of the art, WebGL-based graphics library Threejs for rendering images and the rSuite Reactjs library to provide a sleek user interface that is intuitively compartmentalised and easy to learn.

WebMGA is an evolution of QMGA, an OpenGL and Qt3 based application written in C++ that filled this gap in molecular graphics in 2008. The necessity to iterate QMGA was justified by the inaccessibility of the original software. Significant portions of its code had become deprecated and, as such, its installation procedure on a modern machine was laborious and highly complex, leading to the loss of its user base.

The first stage of this project was to discover the issues in QMGA's codebase and report their respective solutions, in order to maintain its usability. After this task was completed, WebMGA was built to maintain the excellent utility and performance of its predecessor, while improving its architecture and user experience.

WebMGA is deployed at `astromarx.github.io/WebMGA`, enabling users to access it on any browser, while QMGA must be manually compiled and installed, and is only supported on Linux.

The code is distributed as open source.

Contents

1 Introduction

When QMGA [1] was released in 2008, it filled a gap in the field of molecular graphics by providing a flexible platform for visualising systems with non-spherical, convex bodies as simplified representations of mesogenic molecules [2]. At the time, most chemical visualisation applications utilised spheres and rods for representing molecules. For fields such as liquid crystals, the primary focus of QMGA, this is inadequate. As a result, QMGA successfully serviced this computational chemistry niche and garnered 62 citations on Google Scholar.

From its release, QMGA required manual installation on a Linux OS. This alienated the subset of its possible users with limited programming experience in this domain. In fact, the most requested feature extensions on SourceForge [3], the site where the code is hosted, have been a simpler installation procedure and porting the application to other platforms, most notably macOS. As time went on, several of the libraries and frameworks utilised in QMGA evolved, making its installation even more challenging. This led to the inevitable loss of potential users and many installation assistance requests that continue to this day.

In effect, the purpose of this project was to develop QMGA's next iteration. Since its first version provided an efficient and effective visualisation framework, the objective was to update the existing code base to make it accessible to more users than ever before. Given the vast progression of graphics APIs and front-end frameworks since its release, there was also ample room for improvement of the original software. After some essential code maintenance, different avenues through which to move QMGA forward were explored.

The result of this project is WebMGA, a web application that replicates most of QMGA's functionalities, despite having few similarities in code. The project was created using JavaScript and React [4], as well as the Threejs [5] and rSuite [6] libraries. A strong result has been achieved that effectively recreates visualisations virtually identical to those in QMGA. Furthermore, the program includes a new library feature containing configurations provided by Prof. M. P. Allen, in hopes of providing utility to students and educators who may wish to study liquid crystals without generating their own configurations.

1.1 Aims & Goals

- Install the existing version of QMGA and identify the issues that were creating installation difficulties for its users. From these findings, report

the most effective solution for these issues and encourage an update to the existing code base.

- Research different alternatives for updating QMGA and consider each one's feasibility and advantages in the context of modern users.
- Conduct a review to identify changes in the requirements of molecular visualisation programs, which may have evolved since the original release of QMGA.
- Identify a list of concrete requirements for the next iteration of QMGA.
- Deliver a working version of the next iteration of QMGA, including as many of its functionalities as possible given the time constraints.
- Develop code with a highly readable and modular architecture to encourage future additions from users and programmers, whilst ensuring that the delivered functionality is well tested and robust.
- Improve upon the user experience and utility provided by QMGA if possible, while maintaining the essential characteristics that made it an effective tool.
- Include a library feature with multiple molecular visualisations accompanied by scientific explanations, to provide value to educators and students interested in liquid crystals.
- Create straightforward documentation ensuring the resulting platform is easy to learn.
- Evaluate the resulting software, and provide insight into how it could be improved and what additional functionality it could benefit from.

2 Background

2.1 Essential Geometries for Modelling Liquid Crystals

Organic matter sometimes transitions through numerous distinct phases when transforming from liquid to solid. A mesophase is a state of matter which shares some characteristics of both liquids and solids. As the phase of a substance transitions from one state to the other, crystal-like structures are sometimes formed [7].

The distinguishing properties of a liquid crystal are the rotational symmetry and maintained translational ability of its underlying molecules, which allow substances to flow as liquids. Several different types of liquid crystals exist and are primarily characterised by the types of symmetry their molecules retain [8]. The two main liquid crystal mesophases are smectic and nematic. A brief explanation of each phase is provided by Figure 1.

According to Allen and Tildesley, elongated or prolated convex bodies lend themselves well to simulating liquid crystal systems [9]. For this reason, utilising ellipsoids and spherocylinders as a basic building block for simulations of granular systems was a natural choice for QMGA.

The underlying molecules in liquid crystals may be highly complex, and are often composed of hundreds of atoms. For instance, for nematogen quinquaphenyl, modelled with 30 interaction sites based on each carbon atom, the interaction

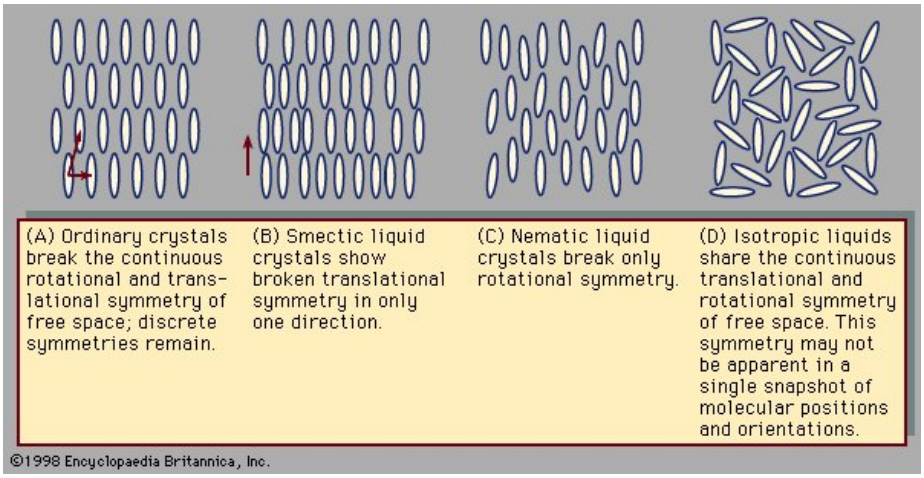
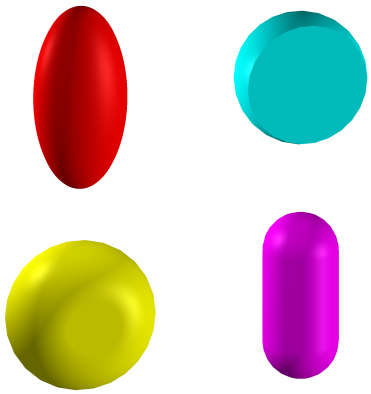
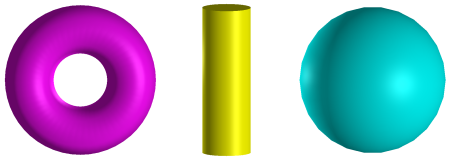


Figure 1: Overview of liquid crystal phases [10].

between a pair of molecules would require $30 \times 30 = 900$ site-site interactions. By representing it as an ellipsoid, only a single molecule-molecule interaction would be needed [9]. While this is a significant simplification, it allows for the computational modelling of hundreds of thousands of molecules, which is useful when the focus is a system, as opposed to individual molecules. As evidenced by the large number of publications utilising ellipsoid geometry for modelling liquid crystals, it is clear that it continues to be a sensible choice of elementary unit for modelling purposes [11, 12, 13].



(a) An ellipsoid (red), cut sphere (cyan), spheroplatelet (yellow), and a spherocylinder (pink).



(b) A torus, cylinder, and sphere.

Figure 2: Set of shapes supported in WebMGA.

Spherocylinders are often used alongside ellipsoids for modelling uniaxial molecules [14, 15]. Formally, a spherocylinder is the Minkowski sum of a sphere and a line segment [16]. An oblate spherocylinder, defined as a "spheroplatelet" in QMGA, is the Minkowski sum of a sphere and a circular platelet. However, it should be noted that the term "spheroplatelet" is sometimes used to refer to the Minkowski sum of a sphere and platelets of varying shapes, such as a rectangle, as is done by Taylor [17]. Given the prevalence of these shapes in the aforementioned studies of liquid crystals, they must be included as fundamental building blocks in the next iteration of QMGA.

Finally, the torus' appearance in studies of densest packings, such as those by Gabbrielli, Jiao, and Torquato [18] and Entov and Verbitsky [19], should be considered in the scope of this project.

2.2 Overview of Existing Software

At this moment, there is no visualising software for the domain of liquid crystals that aims to display large scale granular systems other than QMGA. As such, researchers often create their own visualisation code or utilise general purpose visualisers, as was done by Copar, Porenta, and Zumer [20] with POV Ray [21].

Plato [22] is a notable visualisation library, built with POV Ray, that has been used for liquid crystals in the past. However, the primary limitation of such approaches is their lack of interactivity with their produced images, as their viewing options cannot be adjusted in real time. As such, all their viewing parameters, such as camera zoom and position, need to be specified before rendering. It was precisely this issue that QMGA aimed to address.

JMOL [23] and VMD [24] are general purpose chemical visualisers that can simulate large scale molecular systems and have significant user bases. However, they lack the ability to utilise ellipsoids and other geometries especially relevant to liquid crystals as molecular models in their implementations.

There is a visualising application developed at UCL, LCView [25], which is indeed domain specific to liquid crystals, but it focuses on director fields around singular molecules, as opposed to granular systems, which is the area of interest of QMGA.

2.3 Maintaining QMGA

Perhaps the most time-consuming obstacle encountered during this project was successfully compiling the original version of QMGA on a modern operating system. Several approaches were tried, including the manual installation of outdated versions of OpenGL [26] and Qt 3.3 [27], the libraries used in QMGA, and even the use of deprecated C++ compilers.

After several weeks of tinkering, it was found that the `<unistd.h>` library was not being imported properly in some files, and that the syntax used to define arrays was outdated in several parts of the code. Additionally, an issue was found in the Makefile that was generated by the `make` command with help from the UCL Helpdesk. The issue was that the relevant graphics libraries were not being correctly linked in the compilation procedure.

After these issues were addressed, a brief report was produced by Dr. Germano and provided to QMGA's original developer, Dr. Adrian Gabriel, to be included in the installation instructions. The report is included in Appendix 8.2.

3 Requirements

3.1 Critical Evaluation of QMGA

Without a doubt, QMGA has been a useful tool for the visualisation of coarse-grained models of liquid crystals. The original piece of software includes a wide array of functionalities and delivers them robustly. The two characteristics that defined QMGA's success are discussed below:

- **Utility**. QMGA includes a rich set of features that were carefully selected to benefit liquid crystal researchers. For instance, its slicing and simplified stick rendering options enable users to boost performance and observe the inner sections of systems in order to identify supramolecular structures [2]. It also presents a sophisticated colouring algorithm that provides significant benefit for observing systems' phase orders. Additionally, the program accepts several different input file formats and renders videos of the movement within a system, given the velocities of each molecule.
- **Performance**. QMGA was optimised extremely well. Despite the hardware limitations of its time, it was able to render configurations with 20,000 molecules achieving frame rates very close to smooth motion[1]. Techniques such as back-face and occlusion culling and level of detail (LOD) adjustment were important drivers of its performance. Furthermore, the application was developed using pure OpenGL, as opposed to contemporary graphics libraries like Coin3D [28], which allowed for domain specific optimizations to be implemented in low-level graphics rendering code.

It was considered essential for the next iteration of QMGA to keep these principles at the core of its implementation. Even so, a few areas of improvement were identified and considered for the next iteration.

- **User-friendliness**. The design of the front-end of QMGA could be significantly improved in order to enhance intuitivity and ease of learning. Its implementation places the majority of QMGA's functionalities on stacked, horizontal navigation bars. Several of the functionalities are accessed via buttons that do not have labels, and the large variety of buttons appearing adjacently makes the interface difficult to digest. Even

[1] 26.8 FPS.

for buttons with labels, their placement is often out of context and their purpose is generally hard to discern.

Also, the window for specifying model parameters[2] suggests that all models have the same parameters, which is not the case. Although the application is indeed usable, especially for a sophisticated user base, it is clear that the user experience has room for improvement.

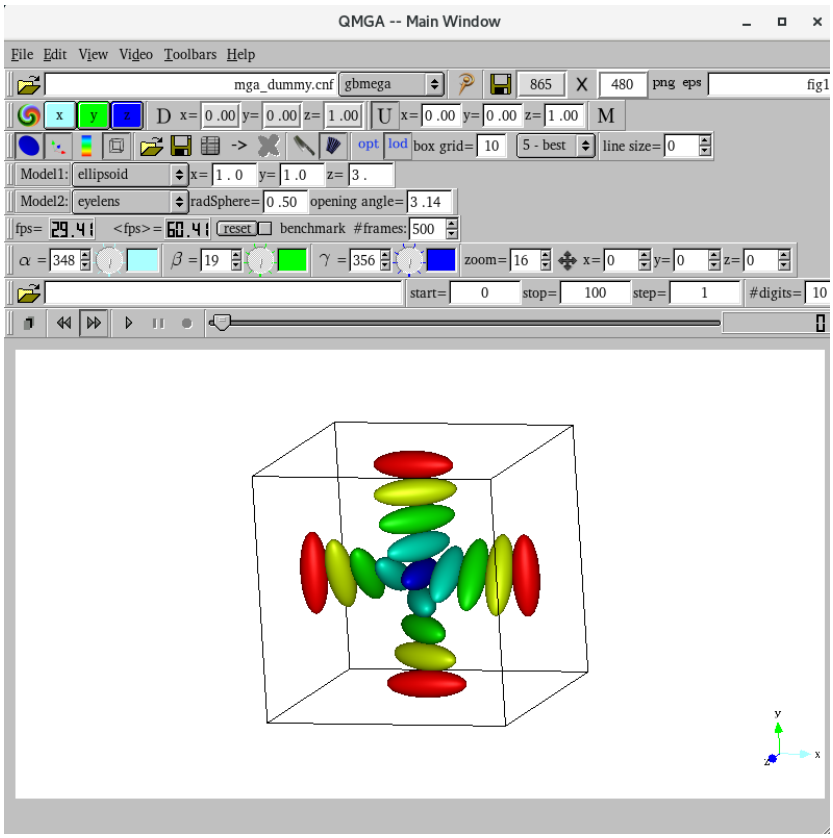


Figure 3: QMGA's user interface.

- **Architecture**. The system architecture of QMGA is another aspect which could be revamped, as it may have contributed to the lack of development from third party users. All of QMGA's rendering code is amalgamated into

[2] A screenshot of the parameter input window and a full screen view of QMGA are included in Appendix 8.1.

a single 5000+ line file[3], and all the visualisation option controls are in the `mga_tools.cpp` file. The front-end of the program is implemented through many files of various file types, such as `.ui` and `.includes.dec.h`. All of these files reside in the main directory of the program, along with the rendering code. The front-end files are meant to be edited within the Qt Designer, and their placement may be confusing for programmers who are not familiar with the Qt environment. Although the code is highly legible, it could be more modular and organised to benefit future developers.

3.2 Non-functional Requirements & Users

The fundamental requirement for WebMGA is that it must enable experts in liquid crystals to visualise different molecular systems effectively. For a list of use cases for researchers, see Table 2. WebMGA should also aim to maintain the performance and utility of QMGA and expand it wherever possible.

As users are likely to be adept in programming to some degree, it is assumed that they will be able to generate input files to feed into WebMGA to visualise their systems. Despite their presumed technical ability, a strong focus should be placed on user experience to ensure WebMGA becomes their preferred visualisation methodology. Finally, the application's architecture must be designed in a way that lends itself well to future expansion and incentives further collaboration.

Apart from researchers, educators and students who are interested in discussing and learning about liquid crystals should find utility in WebMGA. For users of this type, it is unlikely they would ever create configurations beyond the classic liquid crystal phases they are likely to discuss in a classroom setting. For this reason, pre-configured samples of visualisations of nematic, discotic, and smectic liquid crystal phases are included. These were provided by Prof. M. P. Allen[4], alongside a brief study into each of these phase's characteristics. For a list of use cases for educators and students, see Table 1.

[3] `renderer.cpp`

[4] The original configurations were provided in QMGA's format and converted into WebMGA's format.

Table 1: Use cases for educators, students, and researchers.

Program Section	**Use Cases**
Rendered Image	
General Menu	Modify viewing angles, zoom, and look at points using a mouse
Library	Access a variety of visualisations of different liquid crystal mesophases
About	Learn about WebMGA and the provided liquid crystal configurations through the provided documentation
Export	Download the created visualisations with the currently specified viewing settings for educational or research purposes
Autorotate	Automatically rotate the configuration to measure system performance or to analyse configurations from all angles

3.3 Functional Requirements

See Tables 3 and 4 for a list of WebMGA's functional requirements.

Table 2: Use cases for researchers.

Program Section	**Use Cases**
General Menu	
About: Manual	Learn the file format for uploading configurations to WebMGA
Upload	Upload new configurations in `.json` format or previously saved visualisations in `.webmga` format
Save	Download current configuration and view options in `.webmga` format
Visualisation Menu	
Model	Select which molecule set to configure Select shape geometry for the selected set Input geometry parameters Toggle wireframe display Toggle colouring by director Specify user-defined colour
Camera	Select projection type Specify camera position Specify camera zoom Input look at point
Ambient	Specify ambient light colour and intensity Specify background colour
Lighting	Toggle lights Toggle light helpers Specify light colour and intensity Specify light position
Slicing	Input or specify clipping plane locations on all axes Toggle helpers for each clipping plane pair
Reference	Toggle bounding shape Specify bounding shape type Toggle axes Toggle multi-colouring for axes Toggle grid Specify axes and grid size Specify axes and grid colour

Table 3: Functional requirements (1-4).

ID	Description	Priority	Completed
1	***Models***		
1.1	Render elements with correct positions and orientations	M	Y
1.2	Support quaternions and unit vectors for specifying orientations	M	Y
1.3	Render multiple sets of elements with different specifiable characteristics	M	Y
1.4	Specify each sets' elements' shape	M	Y
1.5	Specify each shapes' parameters	M	Y
1.6	Usable interface for specifying Model data	M	Y
1.7	Accept additional orientation formats (Euler and axis angles)	S	Y
1.8	Include support for systems with periodic boundary conditions	S	N
1.9	Display each shape as a wireframe	S	Y
1.10	Colour each shape by the mesophase director	S	Y
1.11	Colour each shape with a user defined colour	C	Y
1.12	Specify each sets' materials' shininess	C	N
2	***Shapes***		
2.1	Ellipsoid	M	Y
2.2	Spherocylinder	M	Y
2.3	Spheroplatelet	M	Y
2.4	Cut sphere	S	Y
2.5	Eye lens	C	N
2.6	Lens	C	N
2.7	Cylinder	C	Y
2.8	Torus	C	Y
3	***Camera***		
3.1	Mouse controls for specifying position, zoom, and look at point	M	Y
3.2	Usable interface for manually specifying camera state	M	Y
3.3	Toggle between orthographic and perspective projections	S	Y
3.4	Integrate manual camera setting system with mouse controls	S	Y
3.5	Toggle autorotation	C	Y
3.6	Specify autorotation speed	C	N
4	***Lighting***		
4.1	Specify ambient light colour and intensity	M	Y
4.2	Specify point light colour, intensity, and position	M	Y
4.3	Specify directional light colour, intensity, and position	M	Y
4.4	Usable interface for specifying light states	M	Y
4.5	Toggle point light and directional light	S	Y
4.6	Specify background colour	S	Y
4.7	Light helpers	C	Y

Table 4: Functional requirements (5-10).

ID	Description	Priority	Completed
5	***Slicing***		
5.1	Slicing in X, Y, and Z from negative and positive directions	S	Y
5.2	Ensure entire molecules are removed when they are sliced	S	N
5.3	Usable interface for specifying slicing state	S	Y
5.4	Show helpers for each slicing direction	S	Y
5.5	Invert sliced area	S	N
6	***Reference***		
6.1	Display bounding box	S	Y
6.2	Display axes	S	Y
6.3	Display grid	S	Y
6.4	Specify grid size	S	Y
6.5	Usable interface for specifying reference elements	S	Y
6.6	Change colour of reference elements	C	Y
6.7	Display each axis in a different colour	C	Y
7	***UI***		
7.1	Responsive, intuitive menu system	M	Y
7.2	Hotkeys for primary functions	C	N
8	***Performance***		
8.1	Change level of detail of entire image dynamically	M	Y
8.2	Measure and display frames per second	M	Y
8.3	Automated performance measurement test	C	Y
8.4	Usable interface for specifying performance test paramaters	C	N
9	***IO***		
9.1	Read file and regenerate element sets (Model)	M	Y
9.2	Read file and regenerate display options (View)	M	Y
9.3	Write model information and view state in JSON format	M	Y
9.4	Export image with specified dimensions	M	Y
9.5	Export image with specified resolution	S	N
9.6	Informative error handling system	S	Y
9.7	Preloaded samples for users to experiment with	S	Y
9.8	Ability to load in multiple configurations at a time	C	N
10	***Information***		
10.1	Manual outlining expected file format	M	Y
10.2	Information about WebMGA	S	Y
10.3	Library of liquid crystal configurations	S	Y
10.4	Explanations for provided configurations	S	Y

4 Design

4.1 Alternatives

Three different approaches for iterating QMGA were considered.

4.1.1 Updating Libraries & Porting

Alternative 1: Updating the Qt framework and OpenGL calls to their current versions and porting the resulting software to Windows and macOS.

This approach was initially preferred, as it would preserve the work done on QMGA and enable past users to quickly return to it. However, as advised by QMGA's original developer, Dr. Gabriel, the porting process would be very time consuming and tedious, and may provide little academic value. In hopes of trying to advance QMGA while finding significant learning opportunities, this plan was abandoned.

4.1.2 Emscripten and WASM

Alternative 2: Embedding the rendering code into a web application utilising Emscripten [29], a WebAssembly (WASM) compiler, and building a new front-end for the application from scratch.

Emscripten is a renowned C++ to WASM converter that has been used to port reasonably large graphics codebases onto the web, such as the BananaBread project [30]. WASM is known to have better performance than plain JS in some cases and could potentially lend itself well to this project.

The primary issue with this approach is that Emscripten natively supports OpenGL ES [31], which provides only a subset of the functionality of OpenGL, excluding costly, yet essential functions, such as `glVertexfv()` and `glLightfv()`. It is possible to convert standard OpenGL to WASM utilising Emscripten, but the general consensus among developers who have attempted this is to opt for alternatives, as this approach often instills bugs that are impossible to resolve without advanced, low-level knowledge of WASM. This would likely have been made worse by the fact that QMGA utilises a deprecated version of OpenGL.

This could have been remedied by updating the function calls in the original QMGA to their latest versions, but this would involve significant hurdles. Firstly, the functionalities that are lost by utilising OpenGL ES are quite advanced and would take significant time to implement. Furthermore, this would require refactoring of almost all of QMGA's rendering code. As such, replicating

fundamental features, such as displaying a single molecule, would take a very significant time investment.

To make matters more difficult, interfacing between the back-end of an application that was written for the Qt architecture would set a significant constraint on how the generated WASM must interact with the new front-end of the application, further complicating the development process.

4.1.3 WebGL

Alternative 3: Rebuilding the program as a web application from scratch, utilising WebGL [32], an embedded systems version of OpenGL meant for the web.

WebGL is almost identical to OpenGL ES, so taking a pure JS approach to developing WebMGA still presented the difficulty of replacing functionalities like `glVertexfv()`. As such, different libraries that could fill this gap, such as Threejs and Babylonjs [33] were considered. The former was chosen due to its excellent documentation.

This approach provided the most flexibility for system architecture design and would allow for restructuring the rendering codebase, in order to improve its organisation. Furthermore, it allowed for seamless integration with Reactjs for developing the user interface of the application.

An additional advantage of this approach was the plethora of tools available for developing web applications with Javascript, such as NPM [34]. This package manager keeps track of all external libraries and their relevant versions in the code base and provides them to the deployed application, essentially making the application future proof and ensuring the goal of maintaining accessibility to a wide array of users is met for many years.

Given this approach's flexibility and its relatively high likelihood of producing a usable result, it was chosen.

4.2 Architecture

The Model-View-Controller design pattern was utilised for WebMGA. An overview of its implementation is provided. See Figure 4 for a diagrammatic representation of the design pattern.

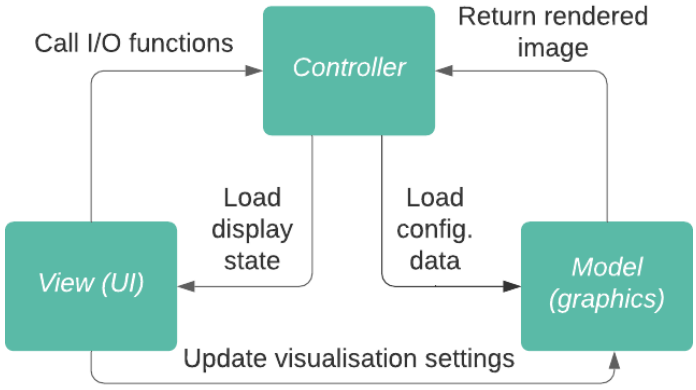


Figure 4: Simplified diagram of WebMGA's MVC design pattern.

4.2.1 Model

In WebMGA, the Model implements essential Threejs graphics components, such as the WebGLRenderer object that produces visualisations. Any change that is effected via the user interface of the program calls a function within the Model.

Furthermore, the Model contains Set objects. Each Set holds a collection of Element objects, which represent molecules, and all of their relevant information, such as their positions, orientations, parameters, and their set-specific viewing options, such as colouring and displaying as a wireframe. Having multiple Sets is what enables the program to display different types of molecules and enables the visualisation of more complex systems. Additional objects such as the camera, clipping planes, bounding shapes and lights are all stored within the Model and utilised within its remit.

4.2.2 View

The View object in the program implements and controls the user interface and stores a state object that contains every aspect of the visualisation options. Every time this state is changed by the user, the View calls one of the Model's functions to reflect that change in the corresponding object in the Model. This is achieved by passing a reference of the Model to the View during its initialisation.

The View is subdivided by the menus it is composed of, the GeneralMenu[5]

[5]The GeneralMenu is placed in the header.

and the VisualisationMenu[6]. The GeneralMenu gives access to functionalities pertaining to file I/O, information, performance, and the library, while the VisualisationMenu contains a set of sub-menus used to interact with different parts of the Model and specify its visualisation parameters.

4.2.3 Controller

The Controller contains the Model and View objects, and mounts the rendering output from the Model and the UI from the View onto the web application's HTML. It contains I/O logic, as it is the main entry point into the program, as well as the performance measurement functionalities.

4.2.4 UML Class Diagram

See Figure 5 for WebMGA's UML class diagram.

[6] The VisualisationMenu is placed in the sidebar.

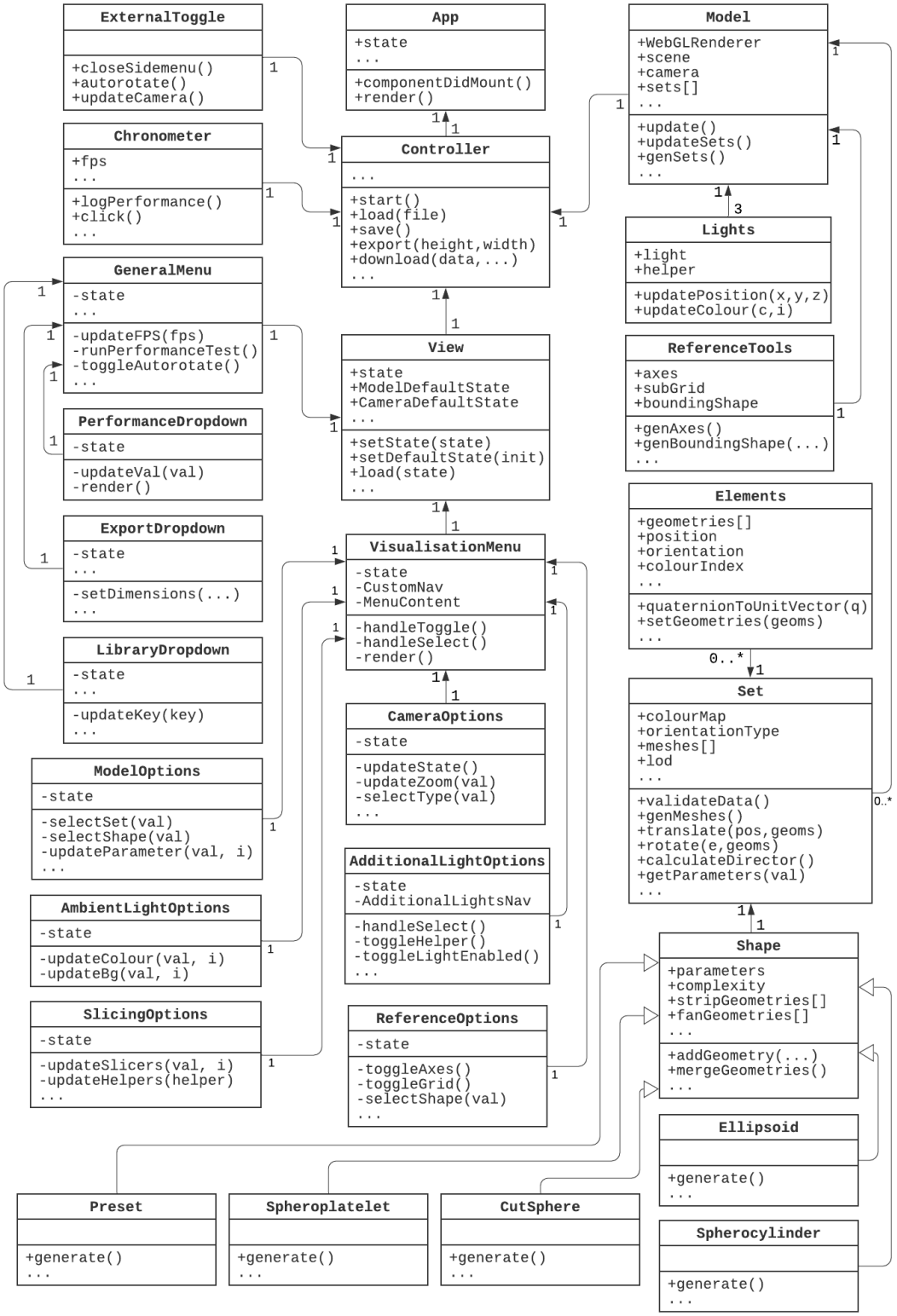


Figure 5: WebMGA's UML class diagram.

5 Development

The development process followed a strongly test-driven approach. It was divided into the sections outlined below. After the completion of each section, new functionalities were manually unit tested and previously included features were re-tested to ensure they were not broken.

5.1 Groundwork

The first goal was to develop the basic building blocks of the application in order to establish an architecture that would simplify the development of future functionalities. At this stage, shell Model and View classes were created and the integration of the necessary libraries was tested.

Then, the code for rendering shapes from the original QMGA was translated into Javascript, adding optimisations where possible. Once the output from the renderer was mounted onto the HTML and the OrbitControls package [35], which is used for moving the camera around the visualisation, was successfully integrated, ellipsoids, spherocylinders, spheroplatelets, and cut spheres[7] were successfully rendered and tested from different viewing angles.

To render a shape, the position of each of its vertices is calculated, and these are grouped into a BufferGeometry object. Using this object and a Material object, which determines the shape's texture, a Mesh object can be created. Each molecule seen in a rendered image is a Mesh that is added to the Scene object. The Scene also contains a Camera object that determines what image is produced based on its position, zoom, and projection type. For an overview of these components, see the Threejs documentation [5].

A rudimentary file reading and positioning system was created to read in coordinates and orientations. The purpose of this system was to produce a visible output through which to test whether the elementary rotation and placement methods were functioning appropriately. This allowed for the development of Set objects, which held information for groups of molecules and reduced overheads in their generation. The rotation system accepts input orientations in quaternions, unit vectors, axis angles, and Euler angles.

[7] See Figure 2a.

5.2 Visualisation Features

The following stage of development involved splitting each of the desired visualisation functionalities into cohesive groups and developing their front-end interaction components and logical implementations simultaneously. The approach used resembled the Agile methodology, as each feature set's development resembled a 'sprint'.

A brief overview of each set of features is provided.

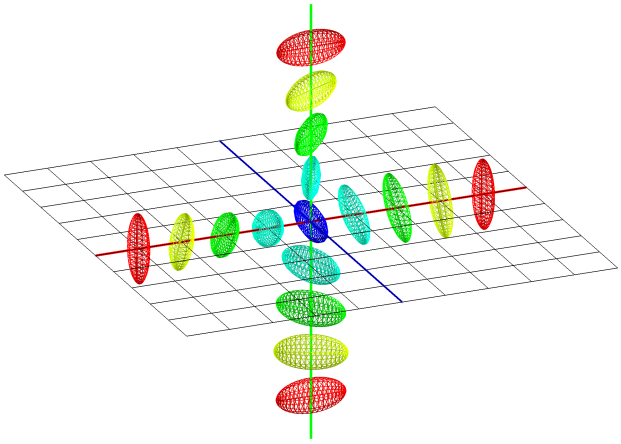


Figure 6: A WebMGA visualisation showing colouring by director, wireframe display, multi-colour axes, and a grid.

5.2.1 Model

The main objectives of the Model functionalities are to enable users to directly specify the attributes and display options of each Set. The respective sub-menu allows for the selection of which Set the user wishes to edit and the specification of its shape, parameters, and colouring. The option to view molecules as wireframes is also given.

The algorithm this section employs to colour molecules by the director is of particular interest. The director of a mesophase is the eigenvector of the order tensor matrix $\mathbf{Q}$ that corresponds to the largest eigenvalue,

$$\mathbf{Q} = \frac{3}{2N} \sum_{i=1}^{N} \hat{\mathrm{e}}_i \otimes \hat{\mathrm{e}}_i - \frac{1}{2} \mathbf{E}$$

where $\mathbf{E}$ is the unit matrix [2].

By calculating the scalar product of a molecule's orientation vector with the director and mapping its value to a colour palette, each molecule's orientation in a system becomes easily discernible.

5.2.2 Camera

The camera options enable users to manually specify the camera's position, zoom and look at point, as well as toggling between orthographic and perspective projections. The camera's data is updated automatically whenever its attributes are updated by the OrbitControls.

In an orthographic projection, the scale of each molecule is not distorted despite its distance from the camera. This is useful for communicating molecule dimensions unambiguously. However, the perspective projection produces images more familiar to humans, as their depth is easily distinguishable. This option may be preferable for discussing a configuration in a live setting, instead of in print. Having both allows for users to gain a richer understanding of their configurations.

Special care was taken to ensure that each projection is adjusted correctly with the toggling of the side menu, exporting, and changing the size of the viewing window.

5.2.3 Ambient & Lighting

The lighting options were divided amongst two sub-menus, due to the large number of parameter inputs that are necessary for specifying colours. As is the case for every colour specification in WebMGA, three sliders are used to enumerate RGB values.

As such, users are given the ability to specify the colour for ambient lighting and the background in the Ambient sub-menu. In the Lighting sub-menu, colouring and positioning options are given for a directional and a point light. Helpers are included for both light types to aid users in visualising the position of each light in their configurations, as shown in Figure 7. These lights are

essential for three dimensional renders, as they are what give a sense of depth to the produced images. Utilising both at the same time achieves this result most effectively.

Should a user desire to produce a two dimensional render, they may disable both of these and achieve their desired result, as is shown in Figure 7.

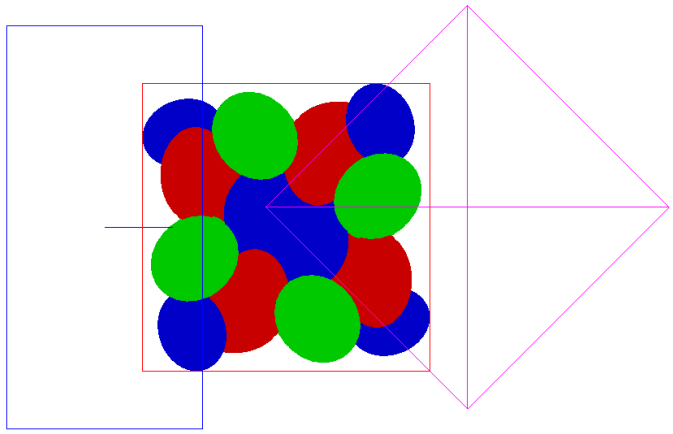


Figure 7: A WebMGA visualisation in 2D, showing a bounding box and lighting helpers.

5.2.4 Slicing

The slicing feature is useful for discerning suprastructures within a large system, as it enables viewing inside of a configuration. For its implementation, two clipping planes are added on each of the system's edges. Via the Slicing submenu, the user may move each of these planes so as to clip everything that resides outside the bounding box they create. Additionally, helper planes are included which may be toggled to help users visualise the slicing box, and know approximately how much of the system they are viewing, as shown in Figure 8.

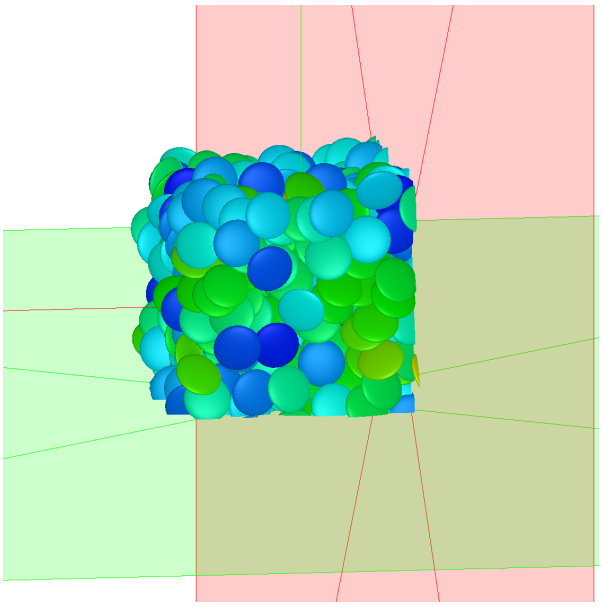


Figure 8: Slicing helpers are used here to slice a system.

5.2.5 Reference

The reference functionalities enable users to display a bounding box, grids, and axes[8] in the rendered model. These may be helpful for a variety of reasons, such as understanding the relative distance between molecules, visualising the tightest packing within a specified shape, or simply for giving a viewer spatial context. The currently implemented bounding box is calculated from each molecule's Mesh, but should be substituted with the unit box used in the computer simulation that generated the configuration.

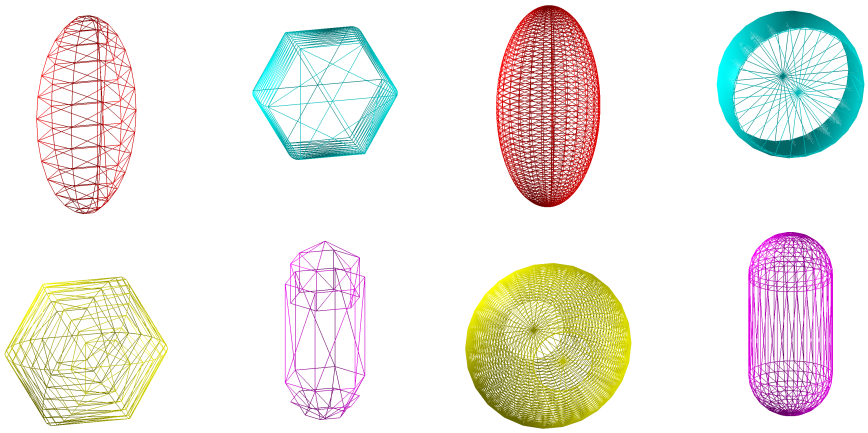
5.3 Performance

In order to ensure accessibility to users with older hardware or without sophisticated graphics cards, the option to decrease the level of detail is provided. In essence, the LOD setting determines how many vertices each shape's geometry has. Decreasing this number, especially for systems with many molecules, improves performance. Samples of each of the QMGA shapes at their lowest and highest LODs are shown in Figure 9.

[8] See Figures 6 and 7.

Initially, poor performance made WebMGA unusable on older systems, particularly on macOS. It was found that this occurred due to anti-aliasing. The removal of this process from the WebGLRenderer object improved performance significantly. Additionally, static rendering was implemented, so that visualisations are only rendered when there is a visible change to the system, as opposed to rendering them continuously using an animation loop. This decreased the computational load immensely, and allowed for satisfactory performance on a large variety of systems.

A performance test was developed using the Model and Chronometer objects, and is further discussed in Section 6.3.1.



(a) LOD 1. (b) LOD 5.

Figure 9: Comparison of QMGA shapes rendered on WebMGA at lowest and highest LOD.

5.4 I/O

The final step for the development of WebMGA was to bring the state of the View and the contents of each Set into a simple, lightweight file format and developing a system for saving and loading these files. The integrity of the entire system can be tested at once by uploading files with different configurations and visualisation settings. As such, sample files served as essential system testing tools.

When users wish to upload entirely new configurations, they need only to

generate a basic Model object in JSON format and upload it. To specify which orientation type their model utilises, they may enter `'q'`, `'v'`, `'a'`, or `'e'`, for the `"orientationType"` entry. The manual for uploading custom configurations is included in Appendix 8.3. An error-handling system was included to notify users of any issues with their generated files and to prevent the program from crashing. A single molecule configuration's Model object in JSON format is shown in Figure 10.

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

Figure 10: Template JSON input file, for specifying Model data.

When users wish to save their visualisations' data, `.webmga` files are generated by creating a View object with each sub-menu's specifications and saving its contents onto a JSON string. Additionally, the Model object, which holds the information stored in each Set object, is stored in the same way. Uploading a `.webmga` file regenerates the Model and View objects, including the camera positioning. A sample `.webmga` file is included in Appendix 8.4.

Finally, the option to export the current visualisation as a PNG with specified dimensions was added.

6 Evaluation

6.1 Visualisation Features & Usability

Each visualisation feature was tested manually upon its completion before any additional development was done. As such, a majority of the delivered functionality fulfills its purpose to an excellent standard.

Additionally, by recreating the the images provided by Prof. M. P. Allen that were rendered with QMGA with WebMGA, using only their respective input files, a strong understanding of the user experience and work processes that must be undertaken to create high quality visualisations was gained.

6.1.1 Strengths

- WebMGA's user interface, as shown in Figure 11, was improved from its predecessor's. The usability heuristic of aesthetics and minimalist design, as discussed by Nielsen [36], was at the center of the front-end development process.
- The inclusion of helpers[9] for lighting and slicing purposes improves the usability of these features significantly as they enhance the visibility of the system's status [36].
- The process of selecting a Set to specify its parameters and colouring options is effective and intuitive.
- Navigating the VisualisationMenu and GeneralMenu is easy as each feature is placed in a cohesive setting.
- The addition of hints in the LOD and Reference sub-menus provide easily accessible explanations for features that may not be obvious to every user. The program could further benefit from the inclusion of additional hints.

6.1.2 Weaknesses

- The slicing system only removes the sections of molecules' that lie outside of the clipping planes, as opposed to entire molecules. As such, the remaining sections of the sliced molecules may impede viewing into the

[9] As shown in Figures 7 and 8.

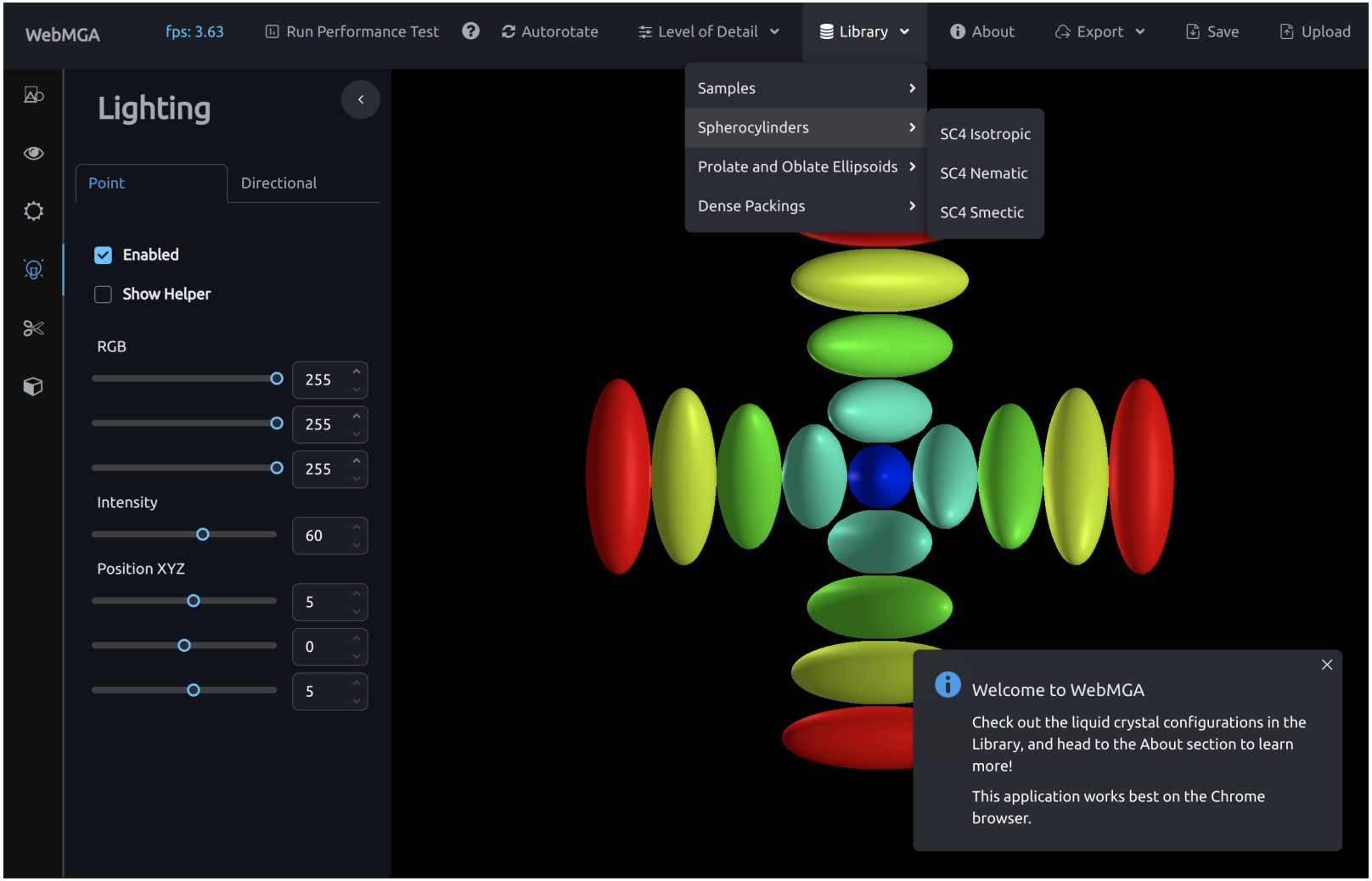


Figure 11: WebMGA's user interface.

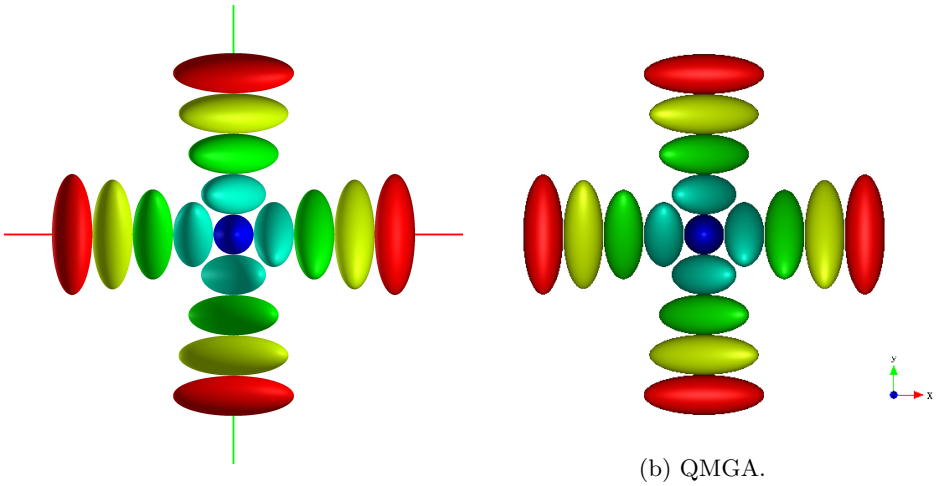
system configuration. Since this is the fundamental purpose of slicing, fixing this issue for future versions of WebMGA is paramount.

- The implemented numerical input boxes can be difficult to operate, as they may create `NaN` values or disallow the user from rewriting an input value. A possible solution for this would be to include confirmation buttons for every numerical input.
- The responsiveness of the web application was not tested to a satisfactory standard, given the time constraints. WebMGA was developed using Google Chrome, and as such, there may be additional bugs when it is used in FireFox and Safari.

6.2 Rendering Accuracy

The accuracy of WebMGA's visualisations was tested by comparing its outputs to those of QMGA given the same configuration data. Virtually identical results were achieved for all 13 of the tested configurations.

The dummy configuration comparison is included with axes enabled for both configurations, to show that the positioning and orientations are mapped to the same three dimensional space, as shown in Figure 12.



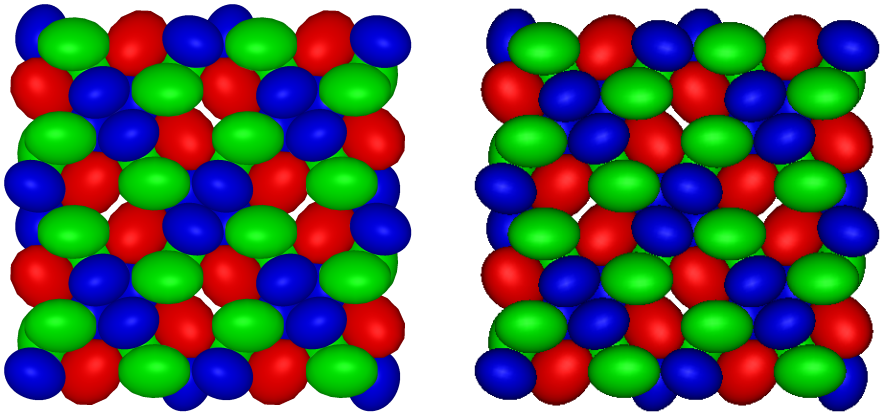
(a) WebMGA.

(b) QMGA.

Figure 12: Comparison of dummy configuration.

The Biaxial Crystal (Large) configuration was used to test the integrity of

the positioning and rotating system when orientations are defined by quaternion parameters in the input file. As is shown in Figure 13, the output image for WebMGA is identical to its QMGA counterpart. Additionally, this test shows that the system for interacting with multiple molecule sets in a single configuration works well for specifying sets with different visualisation options. In this case, each set is coloured according to a user-defined RGB value.



(a) WebMGA.

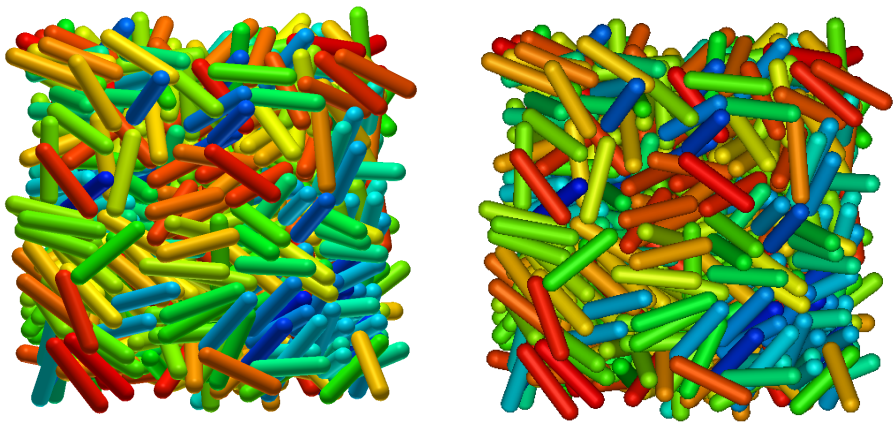
(b) QMGA.

Figure 13: Comparison of Biaxial Crystal (Large) configuration.

The SC4 Isotropic configuration, shown in Figure 14, defines its orientations according to unit vectors. Given this type of orientation parameter WebMGA also produces identical results. It should be noted that the 'length' parameter had to be doubled in WebMGA to achieve the desired result. This suggests that there may be a minor issue in the geometry generating code in the Spherocylinder class.

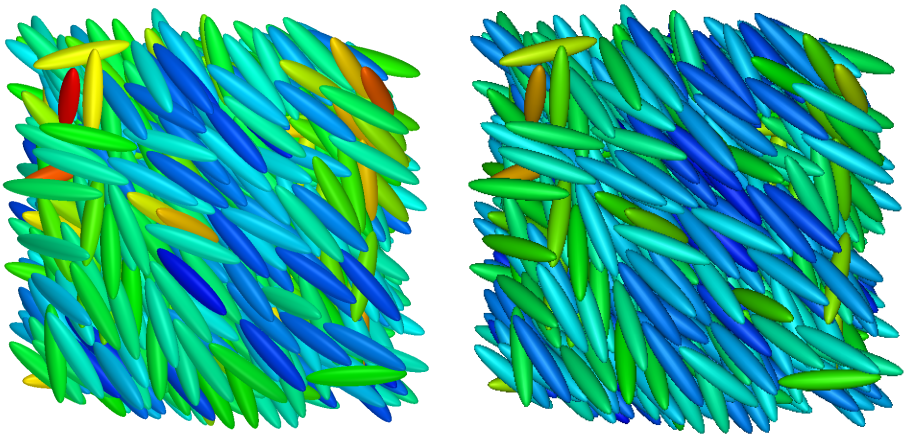
The E5 Nematic configuration, shown in Figure 15, is also defined in terms of unit vectors, and for this configuration and all others the parameters used were identical in QMGA and WebMGA. However, here the colouring algorithm maps colours to its molecules proportionately, but to lower values than its QMGA counterpart. Given that the purpose of colouring by the director is to compare the orientations of molecules within a system, this bears no major consequence. Still, this outcome suggests that additional testing for the colouring algorithm in the Set class is required.

Additional accuracy comparisons are included in Appendix 8.5. To view all



(a) WebMGA. (b) QMGA.

Figure 14: Comparison of SC4 Isotropic configuration.



(a) WebMGA. (b) QMGA.

Figure 15: Comparison of SC4 Isotropic configuration.

of the configurations that were tested, please refer to the library of configurations in the live version of WebMGA.

6.3 Performance

One of the aforementioned strengths of QMGA is its ability to render large scale systems smoothly, even in the range of 10^5 molecules. When it was decided to develop WebMGA in Javascript, it was acknowledged that achieving such performance was not feasible, as Javascript is slower than C++, due to its dynamic typing and automatic memory management processes. However, significant care was taken to ensure that WebMGA can still render large scale systems. To verify if this goal was met, three types of performance tests were carried out.

To carry out these tests a standardised procedure was used in which molecules were randomly generated within a $50 \times 50 \times 50$ box and the camera was programmatically placed and rotated while re-rendering the output image continuously. Using the Chronometer object, the average FPS and its standard deviation was logged every 12 seconds for each test. The tests were carried out on a machine with an Intel® Core™ i7-7700HQ CPU @ 2.80 GHz × 8 Processor and an Intel® HD Graphics 630 (Kaby Lake GT2) graphics card, while no other processes were running.

6.3.1 Level of Detail

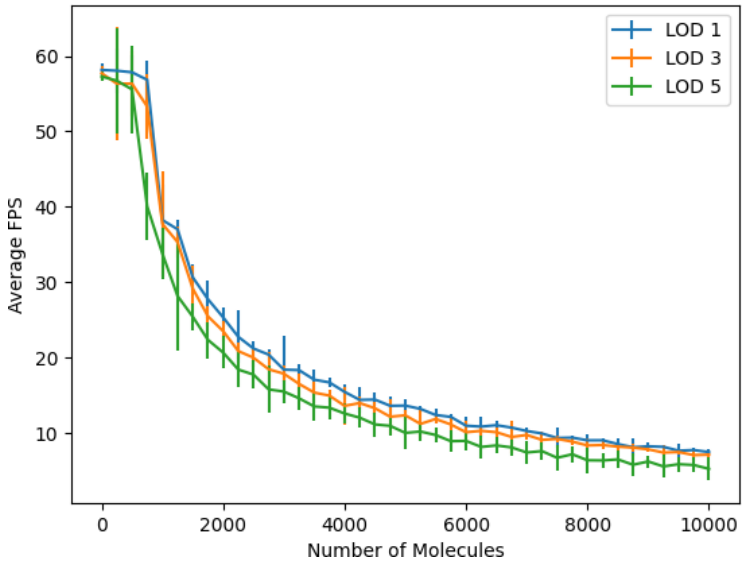


Figure 16: LOD test results, using ellipsoid geometry and Phong material.

The first test aimed to observe WebMGA's performance when rendering large systems with different LODs. This test utilised the largest molecule set size, 10^4 molecules.

The LOD adjusts how many vertices are generated for each molecule's geometry. By increasing the number of vertices, a smoother image can be produced, yet the performance of the entire system suffers, as is shown in Figure 16. The LOD does not map to vertex quantities linearly, as this mapping was chosen based on the appearance of the output images.

A satisfactory result was achieved, as the system can render up to 10^4 molecules at every LOD while remaining usable and rendering approximately 10 frames per second.

6.3.2 Materials

Threejs offers a variety of different materials for generating meshes, and each offers a different trade-off in performance and appearance. The Lambert material is the cheapest, as it does not simulate shininess or highlights. The Standard material is the most expensive, as it uses physically based rendering [37] to calculate the colour at each pixel of the molecules. The Phong material utilises the

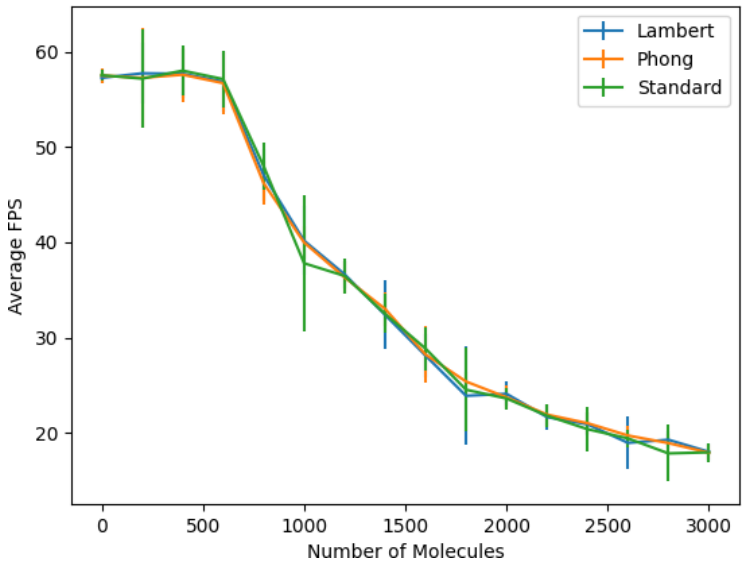


Figure 17: Material performance test results, using ellipsoid geometry and LOD 3.

Blinn-Phong reflection model [38], which is less computationally expensive than physically based rendering, but still simulates shininess. In practise, the three materials performed almost identically in WebMGA, but had some discrepancies in variance, as shown in Figure 17. Since WebMGA utilises static rendering, this difference in variance is not consequential.

The Phong material was chosen as WebMGA's default, as shininess is essential for discerning molecules of the same colour.

6.3.3 Shapes

Finally, the performance of the shapes rendered with code translated from QMGA was compared to the shapes included from the Threejs library. As shown by Figure 18, the former performed significantly worse. However, this is not due to inefficient calculations of their respective geometries, but rather to the complex nature of the shapes' geometries. If the wireframes of these shapes at the same LOD are compared, it is clear why the QMGA shapes do not perform as well. These are shown in Figure 19.

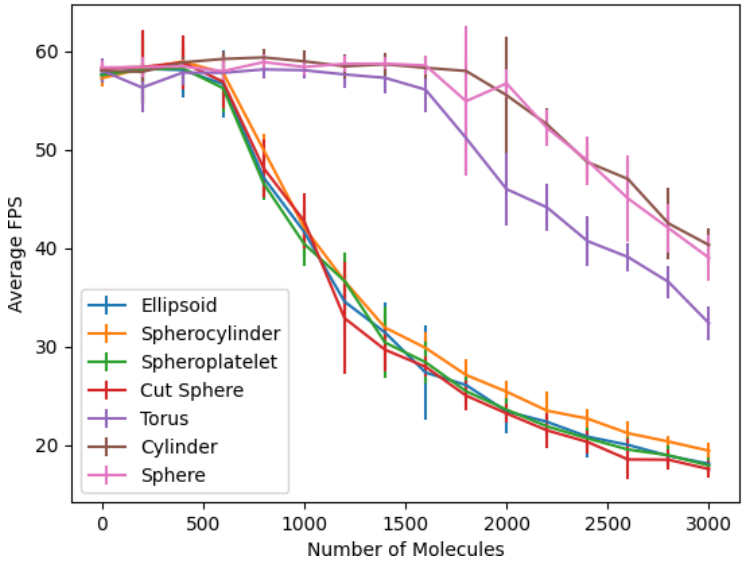
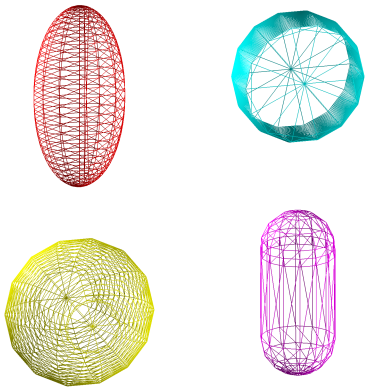
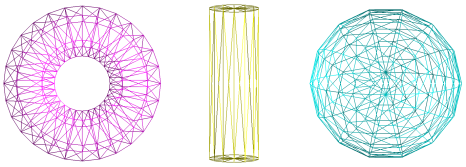


Figure 18: Shape performance test results, using Phong material and LOD 3.



(a) QMGA Shapes.



(b) Threejs Shapes.

Figure 19: Comparison of geometric complexity of QMGA shapes vs Threejs shapes at LOD 3.

7 Conclusion

7.1 Achievement

Overall, WebMGA was developed to a high standard that suffices for its use in a professional context. The system is able to render large scale molecular systems accurately and with good performance, while providing a wide array of robust visualisation options. As such, it is fair to say that the fundamental aspects of utility and performance that characterised QMGA as an effective visualisation tool were successfully replicated.

Furthermore, the initial aim to improve the accessibility of the application was met, as it is deployed and available for use on all major browsers[10] without the need of any installation procedure. Additionally, the goal of improving the user experience was mostly met, as WebMGA implements a polished user interface that is categorically organised in an intuitive way.

The application is not perfect and could benefit from further development, especially for its slicing system. There are also documented bugs that may take away from the user experience. These are included in Appendix 8.6. However, given the system's highly modular architecture and strict adherence to the common MVC pattern, it is likely that future developers will be able to polish the existing code base and expand it with reasonable effort.

Finally, the inclusion of the library feature not only replicates QMGA's utility, but successfully expands it, so as to provide value for educators and those interested in the science of liquid crystals.

7.2 Future Work

7.2.1 Debugging & Polishing

Improving the slicing functionality, as is discussed in Section 6.1.2, should be the top priority for future developers, as this feature provides immense value. Additionally, addressing the issues mentioned in the bug report would improve the user experience of the application.

7.2.2 Completing QMGA's Feature Set

Certain features included in QMGA were not included in WebMGA. Future development should aim to fulfill them.

[10] The software is distributed with the license shown in Appendix 8.3.

- **Periodic Boundary Conditions.** Utilising user-defined bounding boxes, the ability to fold molecule positions based on the bounding box limits should also be included [2]. Additionally, users may benefit from having the option to use periodic boundary conditions to fill the empty space around their configurations with mirror images of the contents of the user-defined unit box.

- **Occlusion Culling.** The performance of WebMGA with large scale systems could be improved with the inclusion an occlusion culling procedure [39]. In essence, occlusion culling is the practise of only rendering the molecules that are visible from the current camera position. Calculating which molecules are occluded, or behind opaque objects and, as such, invisible, may be computationally expensive. However, this approach should still be more efficient than not culling occluded meshes at all.

  It would be worth investigating the OcclusionQueryMesh package [40] in order to achieve this.

- **Video.** In QMGA, users are able to specify the velocity and orientation velocity of each molecule, so as to produce videos showing the movements within defined molecular systems. This is certainly achievable for WebMGA, if such a process is optimised adequately, and would be a substantial addition to the project.

- **Exporting at User-Defined Resolution.** In the current implementation, images are exported at the same resolution as the monitor on which the images are rendered. Ideally, users should be able to define their preferred resolution.

7.2.3 Labelling

WebMGA would significantly benefit from a labelling feature for its axes and for general purposes. This would be especially useful for educators and students utilising WebMGA, as they could read about their desired configurations while interacting with them.

7.2.4 Library

The long term vision for WebMGA is to perfect its utility as a liquid crystal visualisation software, and then generalise it to suit the visualisation of other

molecular structures. This may entail the inclusion of additional geometries that are better suited for different types of chemistry or physics. As the ability to visualise different types of systems grows, the application could provide immense value by continuing to amass different types of configurations and scientific explanations for them.

Furthermore, an API could be developed to allow users to embed WebMGA into their own websites to create a portable, interactive molecular visualiser to accompany their own scientific discussions. If professional users are encouraged to upload their visualisations to a database with a description and are appropriately credited for their work, WebMGA could become a general, web-based, visual scientific library.

7.3 Acknowledgments

We thank Dr. Guido Germano and Prof. M. P. Allen for their insights, suggestions, and guidance throughout the development of WebMGA. We also thank Prof. M. P. Allen for contributing QMGA configurations of various liquid crystal phases and providing a rigorous explanation of their nature. We thank Dr. Adrian Gabriel and Dr. Tobias Ritschel for their suggestions in selecting the best alternative for developing WebMGA and providing feedback throughout the project. Finally, we thank Neil Daeche, from the UCL CS Helpdesk, for his help in compiling QMGA at the initial stages of the project.

References

[1] Gabriel, Adrian T. "Qt-Based Molecular Graphics Application." *QMGA*, 2008, qmga.sourceforge.net/.

[2] Gabriel, Adrian T., Tim Meyer and Guido Germano. "Molecular Graphics of Convex Body Fluids." *Journal of Chemical Theory and Computation*, 4 (3), 468-476, 2008, doi.org/10.1021/ct700192z.

[3] Gabriel, Adrian T. "QMGA." *Feature Requests / 7 Easy to Use Install Procedure*, 2008, sourceforge.net/p/qmga/feature-requests/7/.

[4] React. *A JavaScript Library for Building User Interfaces*, Facebook Inc, 2021, reactjs.org/.

[5] Threejs. *JavaScript 3D Library*, 2021, threejs.org/.

[6] Rsuite. *React Suite*, 2021, rsuitejs.com/.

[7] De Gennes, Pierre-Gilles, and Jacques Prost. *The Physics of Liquid Crystals*. Oxford : Oxford University Press, 1993.

[8] Mahan, Gerald D., and Michael Widom. *Liquid Crystal.* Encyclopedia Britannica, 2019, www.britannica.com/science/liquid-crystal/.

[9] Allen, M. P., and D. J. Tildesley. *Computer Simulation of Liquids.* Oxford University Press, 2017.

[10] Liquid crystal overview diagram. *Encyclopædia Britannica*, www.britannica.com/science/liquid-crystal#/media/1/343083/2254.

[11] Nikishina, Margarita and Dmitri V. Alexandrov. "The Rate of Volume Change of Elliptical Particle in a Metastable Liquid." *28TH Russian Conference on Mathematical Modelling in Natural Sciences*, 030005, 2216, 2020, doi.org/10.1063/5.0003585.

[12] Zhang, Xinyu, Ding Liu, Hanya Jiang, and Junli Liang. "Particle Filter With Unknown Statistics to Estimate Liquid Level in the Silicon Single-Crystal Growth." *IEEE Transactions on Instrumentation and Measurement*, 69 (6), 2759–2770, 2020, doi.org/10.1109/TIM.2019.2930709.

[13] Park, Minji, and Heon Sang Lee. "Rotational Motions of Repulsive Graphene Oxide Domains in Aqueous Dispersion During Slow Shear Flow." *Journal of Rheology*, 64 (1), 29–41, 2020, doi.org/10.1122/1.5120323.

[14] Agarwal, Umang, and Fernando A. Escobedo. "Mesophase Behaviour of Polyhedral Particles." *Nature Materials*, 10 (3), 230–235, 2011, doi.org/10.1038/nmat2959.

[15] Peroukidis, Stavros D. "Biaxial Mesophase Behavior of Amphiphilic Anisometric Colloids: a Simulation Study." *Soft Matter*, 10 (23), 4199, 2014, doi.org/10.1039/C4SM00036F.

[16] Yuan, Ye, Weiwei Jin, Lufeng Liu, and Shuixiang Li. "Shape Effects on Time-Scale Divergence at Athermal Jamming Transition of Frictionless Non-Spherical Particles." *Physica A: Statistical Mechanics and Its Applications*, 484, 470–481, 2017, doi.org/10.1016/j.physa.2017.05.024.

[17] Taylor, Mark P. "Excluded Volume for Polydisperse Spheroplatelets." *Liquid Crystals*, 9 (1), 141–143, 1991, doi.org/10.1080/02678299108036773.

[18] Gabbrielli, Ruggero, Yang Jiao, and Salvatore Torquato. "Dense Periodic Packings of Tori." *Physical Review E*, 89 (2), 2014, doi.org/10.1103/PhysRevE.89.022133.

[19] Entov, Michael, and Misha Verbitsky. "Unobstructed Symplectic Packing by Ellipsoids for Tori and Hyperkähler Manifolds." *Selecta Mathematica*, 24 (3), 2625–2649, 2017, doi.org/10.1007/s00029-017-0353-3.

[20] Copar, Simon, Tine Porenta, and Slobodan Zumer. "Visualisation Methods for Complex Nematic Fields." *Liquid Crystals*, 40 (12), 1759–1768, 2013, doi.org/10.1080/02678292.2013.853109.

[21] POV Ray. *The Persistence of Vision Raytracer*, POV, Persistence of Vision Raytracer Pty. Ltd., www.povray.org/.

[22] Plato. *Plato Documentation*, The Regents of the University of Michigan, plato-draw.readthedocs.io/en/latest/#.

[23] JMOL. *An Open-Source Java Viewer for Chemical Structures in 3D*, jmol.sourceforge.net/.

[24] VMD. *Visual Molecular Dynamics*, Theoretical Biophysics Group, University of Illinois at Urbana-Champaign, 2020, www.ks.uiuc.edu/Research/vmd/.

[25] James, R. LCview. *UCL Liquid Crystal Modelling*, University College London, 2006, www.ee.ucl.ac.uk/~afernand/rjames/modelling/visualisation/guide/.

[26] OpenGL. *The Industry's Foundation for High Performance Graphics*, The Khronos Group Inc., 2021, www.opengl.org/.

[27] Qt. *Cross-Platform Software Development for Embedded Systems*, The Qt Company, 2020, www.qt.io/.

[28] Coin3d. *The Free and Open-Source Implementation of the Open Inventor API*, SGI Inc, 2020, coin3d.github.io/.

[29] Emscripten. *Emscripten 2.0.14 Documentation*, 2015, emscripten.org/.

[30] Zakai, Alon, et al. *Emscripten-Generated Code*, 2020, kripken.github.io/BananaBread/cube2/bb.html.

[31] OpenGL ES Homepage. *The Standard for Embedded Accelerated 3D Graphics*, The Khronos Group, 2011, www.khronos.org/opengles/.

[32] WebGL. *OpenGL ES for the Web*, The Khronos Group, 2011, www.khronos.org/webgl/.

[33] Bablyonjs. *Powerful, Beautiful, Simple, Open — Web-Based 3D Graphics Library.* Babylon.js, www.babylonjs.com/.

[34] Npm. 2021, www.npmjs.com/.

[35] OrbitControls. *Three.js Docs*, 2020, threejs.org/docs/#examples/en/controls/OrbitControls.

[36] Nielsen, Jakob. *Usability Inspection Methods.* Conference Companion on Human Factors in Computing Systems, 377-378, 1995, doi.org/10.1145/223355.223730.

[37] Pharr, Matt, and Humphreys, Greg. *Physically Based Rendering: from Theory to Implementation.* Elsevier/Morgan Kaufmann Publishers, 2017.

[38] Blinn, James F. *Models of Light Reflection for Computer Synthesized Pictures.* Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques, 1977, doi.org/10.1145/965141.563893.

[39] Pantazopoulos, Ioannis, and Spyros G. Tzafestas. *Occlusion Culling Algorithms: A Comprehensive Survey.* Journal of Intelligent & Robotic Systems, 35 (2), 123–156, 2002, doi.org/10.1023/A:1021175220384.

[40] Mrdoob, and Simon Paris. *WebGL2 Occlusion Queries Pull Request 15450.* GitHub, 2020, github.com/mrdoob/three.js/pull/15450.

8 Appendix

8.1 QMGA's Front-End

Screenshots of QMGA's UI are included.

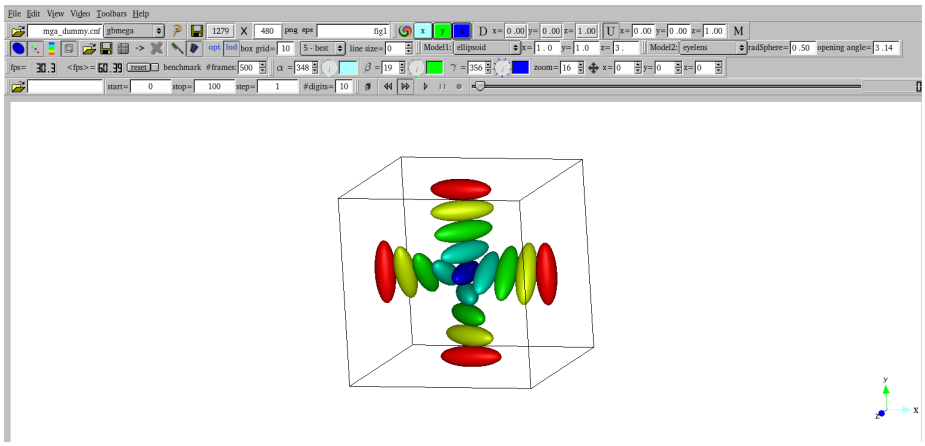


Figure 20: QMGA's full-screen UI.

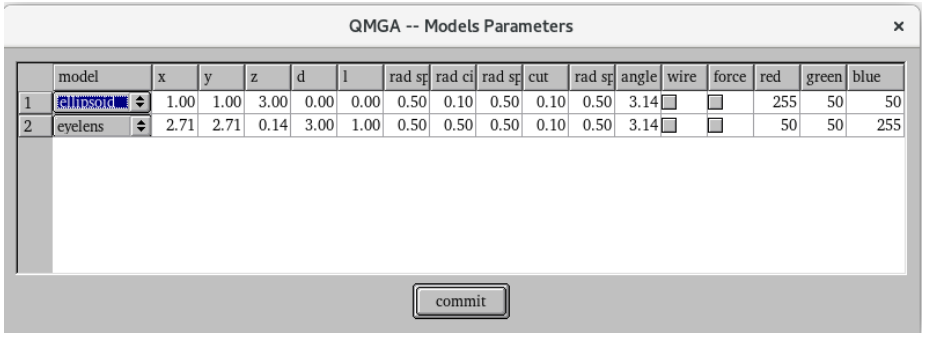


Figure 21: QMGA's model parameter input interface.

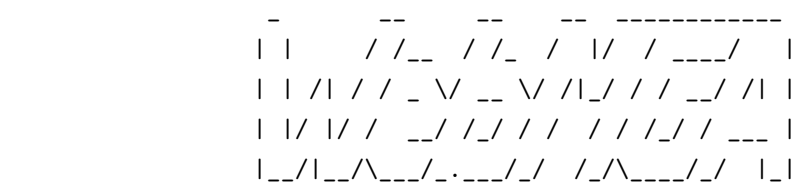
8.2 QMGA Compilation Issues Report

The report outlining the solution to the issues for compiling QMGA is included.

```
========================================================================
1. Add #include <unistd.h> to the files

    renderer.cpp
    dirview.cpp
    mainform.ui.h

This fixes "error: 'sleep' was not declared in this scope";
see https://stackoverflow.com/questions/10976176/c-error-sleep-was-
not-declared-in-this-scope and qmga_mavericks.README

========================================================================

2. In renderer.cpp substitute

      glLightfv(GL_LIGHT1, GL_AMBIENT, (float[]) {0.0f, 0.0f, 0.0f,
      0.0f});

with

      float t1[] = {0.0f, 0.0f, 0.0f, 0.0f};
      glLightfv(GL_LIGHT1, GL_AMBIENT, t1);

and similarly in 13 other following lines that call glLightfv,
glLightModelfv and glMaterialfv.

This fixes "error: taking address of temporary array";
see https://stackoverflow.com/questions/32941846/c-error-taking-
address-of-temporary-array

========================================================================

3. In qmga.pro or Makefile, add -lGL -lGLU to LIBS
```

```
=====================================================================

With the above corrections, QMGA compiles with Qt 3.3 on CentOS 7.9
and our current (November 2020) versions of freeglut (3.10.0) and g++
(4.8.5):

ssh -YJ ggermano@knuckles.cs.ucl.ac.uk ggermano@skate-linux.cs.ucl.
  ac.uk
set path = ($path /usr/lib64/qt-3.3/bin)
qmake
make
qmga

=====================================================================

4. There is only one warning when compiling renderer.cpp:

renderer.cpp: In member function 'void Renderer::renderColormap()':
renderer.cpp:617:52: warning: variable 'rasterPos3f' set but not
  used
     [-Wunused-but-set-variable] void (Renderer::*rasterPos3f)
     (float,float,float) = &Renderer::tglRasterPos3f;

This can be fixed commenting out the line that sets the unused
variable.
```

8.3 Manual



by Eduardo Battistini

```
========================================================================
                            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 2021. Carlos Eduardo Battistini Parra.

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] 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, https://astromarx.github.io/WebMGA.

[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",
                "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 in 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, Uploading, Saving, Exporting, and Uploading, as well as the Library of preset configurations and level of detail (LOD) setting.

By increasing the LOD, 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 its corresponding parameters. Also includes options for colouring manually or by the mesophase director and displaying shapes as wireframes.

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 a grid, axes (which may be multi-coloured), and a bounding shape. The size and colour of the grid and axes may be specified.

~~~~~~

CONTACT

~~~~~~

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

==========

THANK YOU FOR USING WEBMGA!

==========

8.4 Sample .webmga File


```
{
    "model": {
       "sets": [
           {
               "name": "Set A",
               "orientationType": "v",
               "positions": [
                   [0,0,0]
               ],
               "orientations": [
                   [0,1,0]
               ]
           }
       ]
    },
    "state": {
        "reference": {
           "boundingShapeEnabled": false,
           "activeShape": "box",
           "showAxes": false,
           "showGrid": false,
           "multicolour": true,
           "gridColour": {
               "r": 255,
               "g": 255,
               "b": 255
           },
           "size": 50
        },
        "ambientLight": {
           "ambientLightColour": {
               "r": 255,
               "g": 255,
               "b": 255,
               "i": 40
```

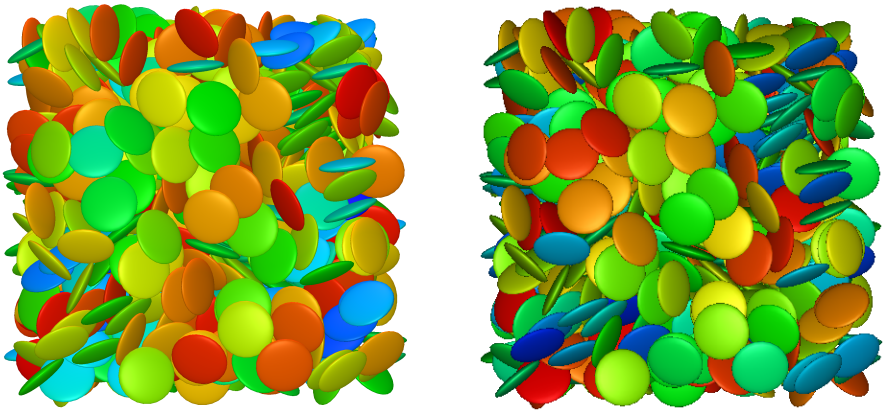
```
    },
    "backgroundColour": {
        "r": 0,
        "g": 0,
        "b": 0
    }
},
"pointLight": {
    "reset": 0,
    "active": "point",
    "enabled": true,
    "helper": false,
    "colour": {
        "r": 255,
        "g": 255,
        "b": 255,
        "i": 60
    },
    "position": {
        "x": 5,
        "y": 0,
        "z": 5
    }
},
"directionalLight": {
    "reset": 0,
    "active": "directional",
    "enabled": true,
    "helper": false,
    "colour": {
        "r": 255,
        "g": 255,
        "b": 255,
        "i": 50
    },
    "position": {
        "x": -5,
```

```
        "y": 0,
        "z": -5
    }
},
"camera": {
    "type": "orthographic",
    "lookAt": {
        "x": 0,
        "y": 0,
        "z": 0
    },
    "position": {
        "r": 10,
        "theta": 3.1,
        "psi": 0
    },
    "zoom": 50
},
"slicing": {
    "clipIntersection": false,
    "helpers": [
        false,
        false,
        false
    ],
    "x": [-50, 50],
    "y": [-50, 50],
    "z": [-50,50]
},
"model": {
    "active": 0,
    "reset": 0,
    "sets": [
        "Set A"
    ],
    "configurations": [
        {
```

```
                "title": "Set A",
                "shape": "Ellipsoid",
                "parameters": {
                    "names": [
                        "X",
                        "Y",
                        "Z"
                    ],
                    "vals": [
                        0.2,
                        0.4,
                        0.8
                    ]
                },
                "colour": {
                    "r": 255,
                    "g": 255,
                    "b": 255
                },
                "colourFromDirector": true,
                "displayAsWireframe": true
            }
        ]
    }
}
```

8.5 Additional Rendering Accuracy Comparisons

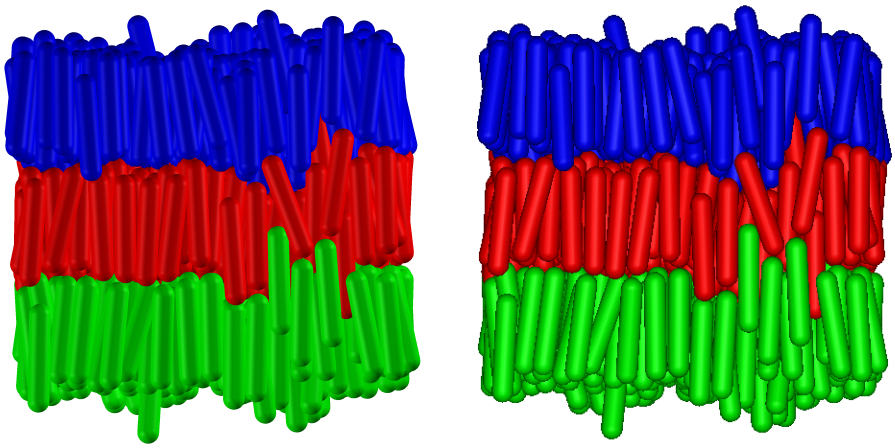
Comparisons of the O5 Isotropic and SC4 Smectic visualisations are included.



(a) WebMGA.

(b) QMGA.

Figure 22: Comparison of O5 Isotropic configuration.



(a) WebMGA.

(b) QMGA.

Figure 23: Comparison of SC4 Smectic configuration.

8.6 Bug Report

A list outlining UX issues and bugs is included.

Table 5: Bug report.

ID	Bug Report	Priority
1	***UX Issues***	
1.1	Responsiveness is not tested thoroughly on all browsers	High
1.2	Making the background black/white should be done with a button	Medium
1.3	`.webmga` files are not automatically formatted	Low
2	***Bugs***	
2.1	After a file is uploaded, the upload button becomes unresponsive and the page must be refreshed to upload a new file	High
2.2	The side menu formatting fails when the browser size is reduced in width by more than 50%	Medium
2.3	The computed bounding box does not update when a new configuration is loaded	Medium
2.4	Exported images are not named uniquely when using Safari	Low
2.5	Slicing number input box is hard to see on Firefox	Low