LONDON'S GLOBAL UNIVERSITY

UCL

WebMGA 3.0

Refinement of an Interactive Viewer for Coarse-Grained Liquid Crystal Models

Candidate Number: GYWT8[1]

MEng Computer Science

Supervisor: Guido Germano

Submission Date: 26th April 2024

[1]**Disclaimer:** This report is submitted as part requirement for the MEng in Computer Science 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

WebMGA 3.0 is a web-based visualisation tool for molecular simulation outputs which refines on previous versions by Eduardo Battistini (2021) and Yue He (2023). It can be used to produce 3D renders utilising a range of geometries for representation molecules. WebMGA was first developed to replace the older QMGA tool, which has remained unmaintained since 2009 and is difficult to install on modern systems, providing extended features over this older program.

This project aimed to fix existing issues in WebMGA and further enhance the existing feature set. Most effort was focused on implementing additional molecule geometries and optimising existing geometries, improving visualisation for the axes and director, visualising periodic repetitions of a configuration, and implementing a distance based variable level of detail performance optimisation.

Contents

0.1 Acknowledgements 1

1 Introduction **3**

2 Context **5**

2.1 Molecular Graphics 5

2.2 Liquid Crystal Modelling 6

2.2.1 Director 6

2.2.2 Periodic Boundary Conditions 6

2.2.3 Molecule Shapes 7

3 Requirements and Analysis **8**

3.1 Non Functional Requirements 8

3.2 Functional Requirements 8

3.3 Use Cases 8

4 Design and Implementation **10**

4.1 Colour from Director 10

4.1.1 WebMGA 2.0 Implementation 10

4.1.2 WebMGA 2.0 Bugs 10

4.1.3 Improvement Goals 10

4.1.4 WebMGA 3.0 Implementation 11

4.1.5 WebMGA 3.0 Bugs 11

4.2 Axes 11

4.2.1 WebMGA 2.0 Implementation 11

4.2.2 WebMGA 2.0 Bugs 13

4.2.3 Improvement Goals 13

4.2.4 WebMGA 3.0 Implementation 14

4.2.5 WebMGA 3.0 Bugs 16

4.3 Shapes 16

4.3.1 WebMGA 2.0 Implementation 16

4.3.2 WebMGA 2.0 Bugs 16

4.3.3 Improvement Goals . . . 16
4.3.4 WebMGA 3.0 Implementation . . . 19
4.3.5 WebMGA 3.0 Bugs . . . 31
4.4 File types . . . 31
4.4.1 WebMGA 2.0 Implementation . . . 31
4.4.2 WebMGA 2.0 Bugs . . . 31
4.4.3 WebMGA 3.0 Implementation . . . 31
4.4.4 WebMGA 3.0 Bugs . . . 32
4.5 Periodic Repetition . . . 33
4.5.1 Improvement Goals . . . 33
4.5.2 WebMGA 3.0 Implementation . . . 33
4.5.3 WebMGA 3.0 Bugs . . . 34
4.6 Optimisations . . . 34
4.6.1 WebMGA 3.0 Implementation . . . 34
4.6.2 WebMGA 3.0 Bugs . . . 35
4.7 Miscellaneous Improvements . . . 35

5 Analysis and Testing **37**
5.1 Level of Detail Performance . . . 37
5.2 Configuration Visualisations . . . 37

6 Conclusions and Evaluation **39**
6.1 Achievements . . . 39
6.2 Evaluation . . . 40
6.3 Future Work . . . 40
6.3.1 Axis Labelling . . . 40
6.3.2 Colour Palette Guide . . . 40
6.3.3 Colour Palette Adjustment . . . 41
6.3.4 Bug Fixes . . . 41

A Appendices **44**
A.1 Project proposal . . . 44
A.2 Interim Report . . . 48

0.1 Acknowledgements

This project was undertaken with the supervision of Guido Germano, University College London. It is a continuation of work by Eduardo Battistini (2021) and Yue He (2023), both University College London.

Giorgio Cinacchi, Autonomous University of Madrid, provided useful insight for certain parts of the project. For implementation of the lens molecule geometry, some sample molecule configurations and associated guidance was provided.

Chapter 1

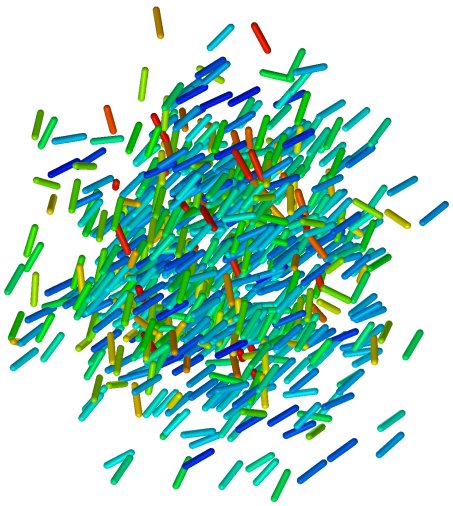
Introduction

WebMGA 3.0 is a visualisation tool for molecular simulation outputs which refines on previous versions by Battistini [1] and He [2]. It provides a more modern, maintainable, and accessible alternative to the older QMGA tool by Gabriel, Meyer, and Germano [3]. The program is available as a web app at REMOVED[1], with source code at REMOVED[2]. Development utilised JavaScript with the React[6] framework and three.js[7] 3D graphics library. This project aimed to fix existing issues in WebMGA and further enhance the existing feature set. Most effort was focused on implementing additional molecule geometries and optimising existing geometries, improving visualisation for the axes and director (defined in Section 2.2.1), visualising periodic repetitions of a configuration (defined in Section 2.2.2), and implementing a distance based variable Level of Detail optimisation.

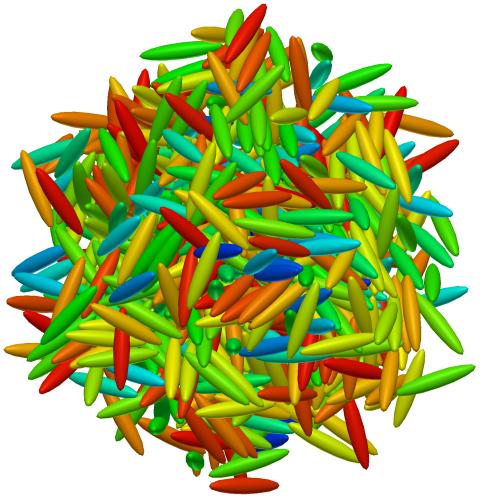
Since the project aimed to work on an existing program, development was structured to begin with bug fixing, followed by implementing previously identified missing features, and finally extending the feature set based on user feedback.

This report provides a background summary explaining the requirements and terminology for molecular simulation rendering (Chapter 2), descriptions for the changes required and made for WebMGA 3.0 (Chapters 3 and 4), and quantitative summaries for performance improvements and qualitative analysis of rendered systems (Chapter 5). Finally, achievements are summarised and the project successes evaluated (Chapter 6).

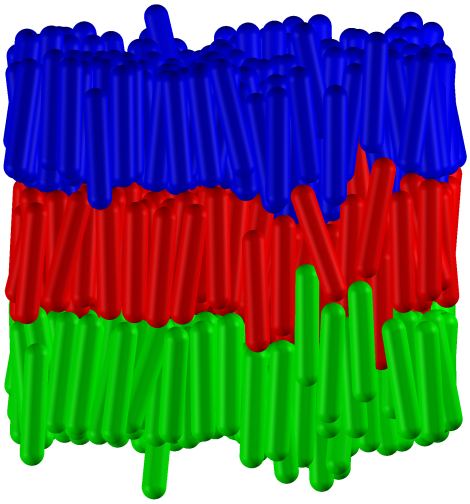
[1]Text removed for submission since URL contains author name. Visible at [4].
[2]Text removed for submission since URL contains author name. Visible at [5].



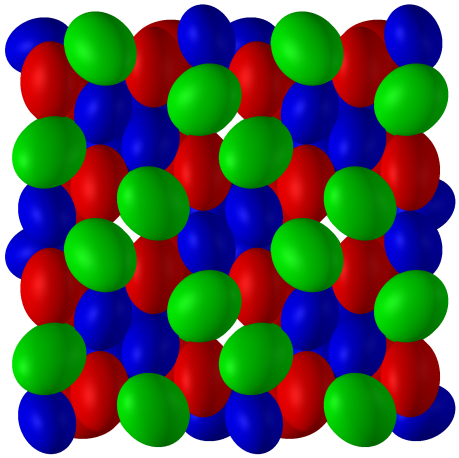
(a) Unfolded SC4 Nematic.



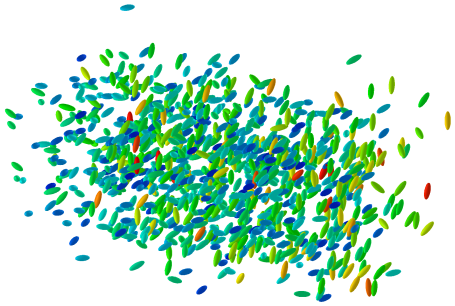
(b) E5 Isotropic.



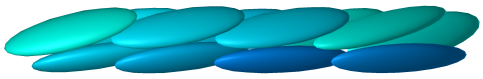
(c) SC4 Smectic.



(d) Biaxial Crystal.



(e) Unfolded E3 Chiral Nematic.



(f) HBC (In Cylinder).

Figure 1.1: Various configurations generated using WebMGA 3.0.

Chapter 2

Context

2.1 Molecular Graphics

The ability to visualise outputs from molecular simulations, particularly in the domain of liquid crystals, is important for understanding and communicating findings. QMGA[3] is a tool which can be used to generate 3D graphical representations of molecular configurations. Despite being unmaintained since 2009[8], it remains in active use to this day, having been used within the last year in publications by notable authors such as Ramírez González and Cinacchi [9] and Mazzilli, Satoh, and Saielli [10]. While another visualisation tool exists within the liquid crystal domain, LCview[11, 12], it produces plots of director and/or potential fields, rather than showing the structure of large multi-molecule system.

Since QMGA has not been updated in so long, it continues to depend on the severely dated Qt 3 framework (Qt 4 was released in 2005, 2 years before QMGA was released) requiring the installation of unmaintained and difficult to acquire libraries (e.g. the Debian Linux distribution removed all Qt 3 libraries in 2012[13]). Additionally, since the program is distributed as source code, it must be manually built by the user which is not trivial for inexperienced users. This is complicated further by the fact that modern C compilers fail without certain modifications to the source code (described by Battistini [1] in their "QMGA Compilation Issues Report"). All of these problems make installation on a modern system a significant barrier to usage. In preparation for this project, a successful install of QMGA was completed within a Debian 6 virtual machine, which verified the claims made regarding install difficulties.

WebMGA is a project begun by Battistini [1] in 2021 which aims to address this accessibility issue whilst replicating the functionality of QMGA. It was continued in 2023 by He [2]. It's written in JavaScript using the React[6] framework with the three.js[7] library for 3D rendering. This addresses the accessibility issues of QMGA since it can be easily accessed using just a web browser. While WebMGA contains full functionality

for rendering most molecule configurations from QMGA, it still has performance and functionality limitations, as well as some bugs. WebMGA 3.0 aims to address a majority of these issues.

2.2 Liquid Crystal Modelling

Most details regarding molecular simulation is not required to understand the implementations made for this project. Some key concepts which will be used throughout are defined in the following subsections, based on Allen and Tildesley [14] except where cited otherwise.

2.2.1 Director

Under a coarse-grained potential model, liquid crystal configuration have a long-range orientational order, and sometimes also a long range positional order. An order represents a preferred alignment for molecules within that system[15]. The orientational order can be described by a magnitude S, and a direction $\mathbf{n}$. $\mathbf{n}$ is typically referred to as the director. Both S and $\mathbf{n}$ can be derived from the order tensor $\mathbf{Q}$, which is defined as follows for a system containing N molecules each with unit vector principal axis direction $\mathbf{e}$,

$$\mathbf{Q} = \frac{3}{2N}\sum_{i=1}^{N}(\mathbf{e}_i \otimes \mathbf{e}_i) - \frac{1}{2}\mathbf{I} \tag{2.1}$$

$$\mathbf{e}_i = \begin{pmatrix} \mathbf{e}_{ix} \\ \mathbf{e}_{iy} \\ \mathbf{e}_{iz} \end{pmatrix}. \tag{2.2}$$

The director $\mathbf{n}$ is defined as the eigenvector that corresponds to the maximum eigenvalue of the order tensor $\mathbf{Q}$.

Since the director is a useful property for describing and understanding a system, it is convenient to be able to visualise both the director itself and how each molecule in the system aligns with it. Implementation of a director axis and director based axis colouring is discussed further in Sections 4.1 and 4.2.4, while molecule colouring based on director alignment was already implemented by Battistini [1].

2.2.2 Periodic Boundary Conditions

Typically molecular simulations can only be performed on small systems of $10 < N < 10,000$ molecules due to speed and/or storage constraints. With systems of this size, which is insufficient for simulating bulk liquids. Periodic boundary conditions allow modelling only a portion of an entire system[3], where an infinite lattice is simulated by repeating

a smaller simulation box[16]. In each repeated boxes, periodic images of the molecules in the simulation box move in the exact same way. When a molecule leaves through a face of the simulation box, one of its periodic images enters through the opposite face of the simulation box with the same movement properties.

Due to the necessary use of periodic boundary conditions for many molecular simulations, it may be useful to visualise a larger portion of the infinite lattice, as discussed in Section 4.5. Folding and unfolding of molecules based on periodic boundary conditions was already implemented by He [2].

2.2.3 Molecule Shapes

Molecular simulations model liquids as a collection of molecules. Each molecule has a shape approximately representative of the positions and sizes of the atoms they consist of. Allen [17] identifies a range of appropriate molecule geometries commonly used for hard-particle simulations; specifically the hard sphere, prolate ellipsoid, spherocylinder, double cut sphere, and spheroplatelet. In addition to these, QMGA implements the eyelens shape. Cinacchi and Torquato [18] have researched packings of molecules with a biconvex lens shape. All of these molecule shapes should be supported by WebMGA to successfully visualise models of each type. Implementations for geometry generation of all required types are discussed in Section 4.3.

Chapter 3

Requirements and Analysis

3.1 Non Functional Requirements

Non functional requirements are summarised in Table 3.1.TODO

3.2 Functional Requirements

For each WebMGA 3.0 development, Chapter 4 summarises the existing WebMGA 2.0 implementation and its limitations before discussing improvement goals and how they were implemented Table 3.2 summarises these improvement goals.TODO

3.3 Use Cases

Use cases requirements are summarised in Table 3.3.TODO

a	b
c	d

Table 3.1: Non Functional Requirements.

a	b
c	d

Table 3.2: Functional Requirements.

a	b
c	d

Table 3.3: Functional Requirements.

Chapter 4

Design and Implementation

4.1 Colour from Director

4.1.1 WebMGA 2.0 Implementation

A molecule's principal axis' alignment with the director can be visualised by assigning a colour from a palette corresponding to the angle between these vectors. This angle can be determined by,

$$\theta = \arccos(\mathbf{e} \cdot \mathbf{n}) \tag{4.1}$$

where $\mathbf{e}$ is the unit vector principal axis direction and $\mathbf{n}$ is the unit vector director.

WebMGA 2.0 currently selects a colour using this angle by rounding this angle to the nearest integer and using it as an index to sample from a list of RGB values. This matches QMGA's implementation.

4.1.2 WebMGA 2.0 Bugs

While this implementation is completely functional, it has the limitation of not distinguishing between angles within $1rad$ of each other. It also means an entire new palette file would need to be calculated and stored for any alternate colour range.

4.1.3 Improvement Goals

An alternative to sampling a discrete palette would be to use an HSL colour space rather than RGB, since a hue is continuously defined by some angle in the range of $[0, 2\pi$. In fact, the RGB values used in the palette file correspond to a hue range of 0 (red) to $\frac{-4\pi}{3}$ (blue). Since three.js supports defining colours using HSL values[19] this was possible to implement.

While not a strictly necessary improvement since there is likely no visible change for the user, this change was made largely as a task to help become more familiar with the

existing code base at the start of development.

4.1.4 WebMGA 3.0 Implementation

The function to sample a colour (as a three.js Color object) from a given **e** and **n** was rewritten. It now additionally takes in a palette start hue (referred to as A) and and a rotation to get to the end hue (referred to as **B**). **B** is defined as a vector percentage of the hue range to cycle through, with $\mathbf{B} = 0$ indicating 0% and $\mathbf{B} = 1.0$ indicating 100% (i.e. $\mathbf{B} = 1$ results in an end hue of A, sweeping over hues in the positive direction, while $\mathbf{B} = -1$ results in an end hue of A, sweeping over hues in the negative direction). A and **B** by default take on a value of 0 (red) and $\frac{-2}{3}$ (which results in an end hue of $\frac{-4\pi}{3}$, which is blue) respectively to preserve the old colour range.

After a θ value is derived as in Equation (4.1), an HSL colour is derived using the following process: First,

$$\theta = \begin{cases} 4\theta & if \theta < \frac{\pi}{2} \\ 4(\pi - \theta) & \text{otherwise.} \end{cases} \tag{4.2}$$

since θ can only be in the range $[0, \pi)$, and colouring should be identical for θ values symmetrically in $\frac{\pi}{2}$ since the director is equivalently defined as its reverse. Multiplication by 4 occurs since θ for a hue should be in the range $[0, 2\pi)$ rather than $\left[0, \frac{\pi}{2}\right)$. Hue is then derived from the following formula,

$$H = \frac{(A + \mathbf{B}\theta) \mod 2\pi}{2\pi}. \tag{4.3}$$

mod 2π is applied to ensure an invalid hue does not occur if B is erroneously defined with a value greater than 1.

Saturation (S) and lightness (L) values of 1 and 0.5 respectively are used since this subjectively gives a well presented colour. The HSL tuple is applied to a three.js Color object which is returned.

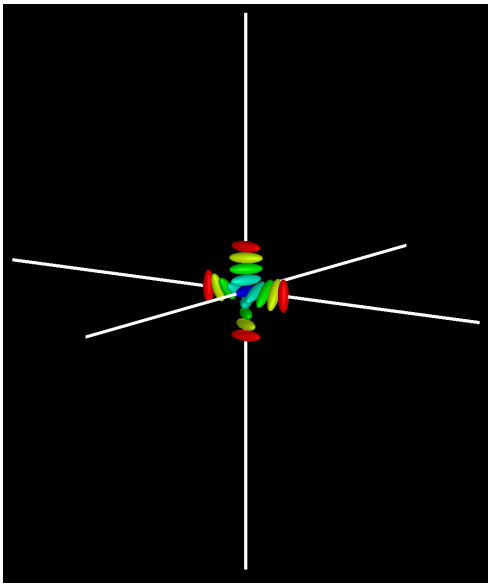
4.1.5 WebMGA 3.0 Bugs

The new colours appear indistinguishable from the previous implementation, indicating a successful implementation.

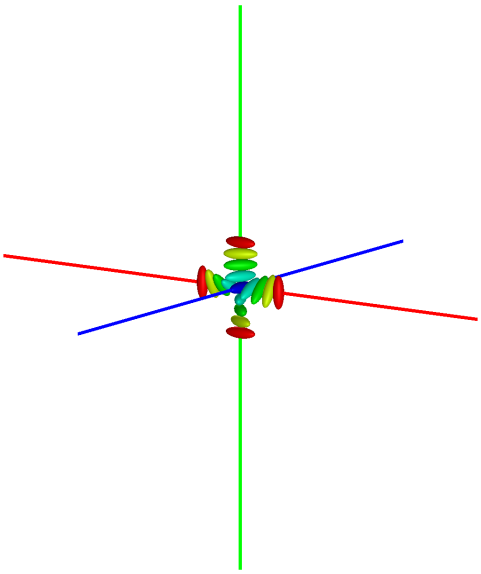
4.2 Axes

4.2.1 WebMGA 2.0 Implementation

In WebMGA 2.0, the 3D axes are displayed as shown in Figures 4.1a and 4.1b, and controlled through the user interface as shown in Figure 4.1d (visibility and colour toggles).

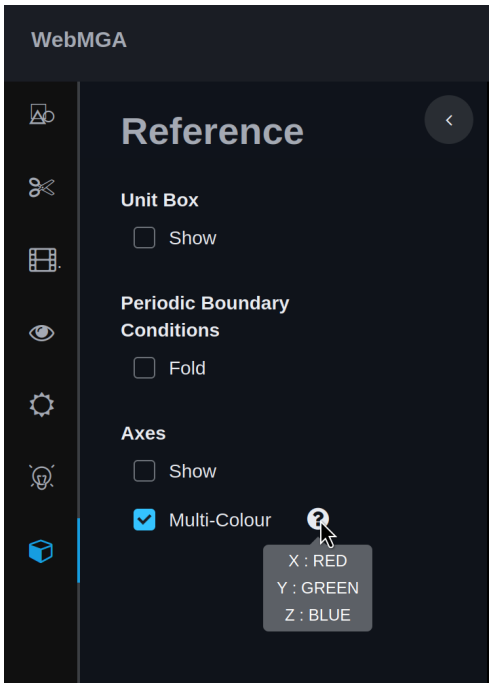


(a) Colour disabled



(b) Colour enabled

(c) Colour disabled (light background)



(d) GUI controls

Figure 4.1: Axes in WebMGA 2.0

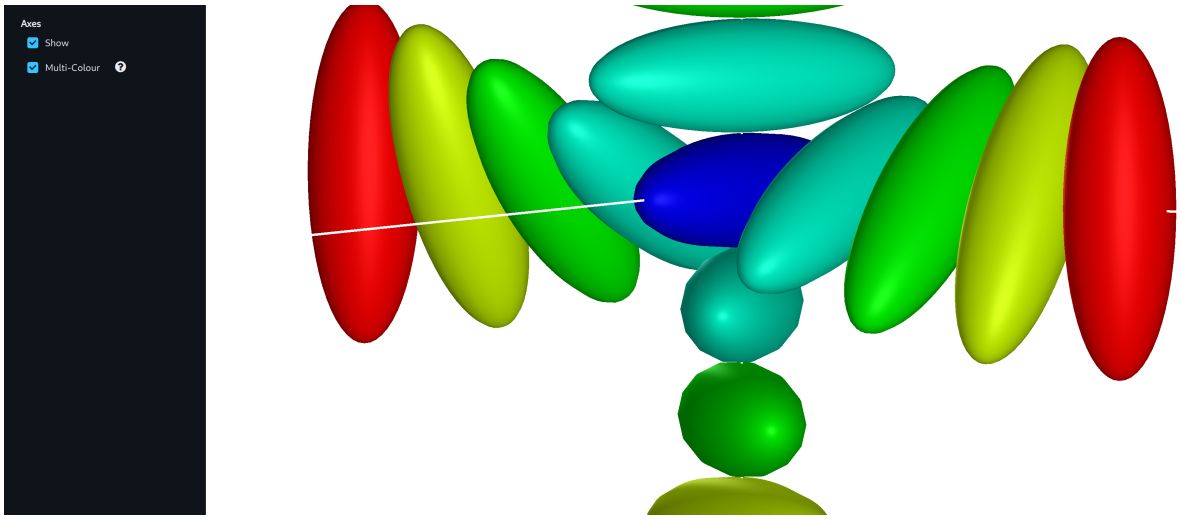


Figure 4.2: Bug where axes are not coloured despite the "Multi-Colour" toggle being enabled when axes are first enabled.

Axes take the form of three lines of fixed lengths in the x, y, and z directions. Each line's midpoint is the lab fram coordinate $(0, 0, 0)$, where all axes meet. Axes extend in both positive and negative directions. They are not shown by default and, when first enabled, are uncoloured. When coloured, the x axis is red, the y axis is green, and the z axis is blue.

Visibility is toggled using the "Show" button and colour is toggled with the "Multi-Colour" button. A question mark icon is next to the "Multi-Colour" which shows a tooltip when hovered specifying the axis colour scheme.

4.2.2 WebMGA 2.0 Bugs

When the axes are toggled to visible for the first time, if the "Multi-Colour" toggle has not been interacted with first, the axes will be uncoloured, despite the "Multi-Colour" toggle being enabled by default. This is shown in Figure 4.2. To enable colour for the first time, the "Multi-Colour" toggle must be disabled and then re-enabled.

When the environment is set to light mode (white background) with coloured axes disabled, the axes become difficult to view since they retain a white colour as default, blending into the background as shown in Figure 4.1c.

4.2.3 Improvement Goals

- Axes are unlabelled
  - Axes should be changed to extend only in the positive direction
- Director(see Section 2.2.2 for definition) is not shown

  - An additional line should be shown indicating director direction

- Colours should be labelled or meaningful
  - Colour axes according to angle with director (as in Section 4.1)
- Axes should not be obscured
  - Place axes in screen corner rather than centre
- Axes should be clearly distinguishable
  - Ensure axes retain contrast with background under light and dark views

4.2.4 WebMGA 3.0 Implementation

Axes Positions

The existing implementation was found to be needlessly convoluted so was largely stripped out. For example, coloured and uncoloured axes were implemented entirely separately, resulting in a large amount of duplicated code and convoluted logic flow. The bug identified in WebMGA 2.0 regarding uncoloured axes showing with "Multi-Colour" enabled, for example, was found to occur due to incorrect colour object initialisation, meaning what should be "Multi-Colour" axes showed as uncoloured since the colours are not defined when these lines are loaded the very first time.

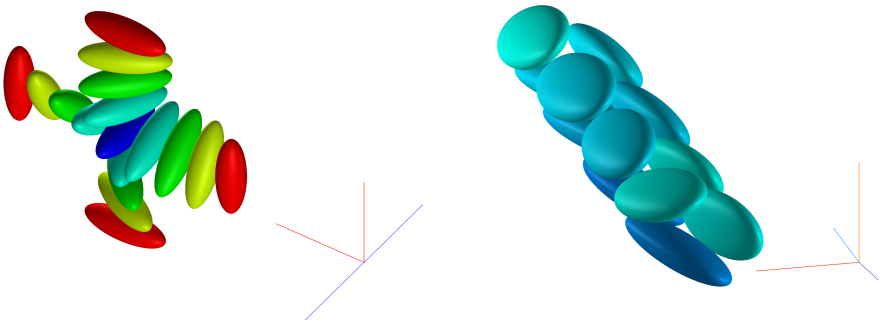
In the new axes code, they are simply defined in terms of an axes centre point, three axes vectors, and an axis length scale. The axes vectors are handled in the lab frame so are trivially defines as $x = (1, 0, 0)$, $y = (0, 1, 0)$, and $z = (0, 0, 1)$.

Since the axes centre needs to remain in a fixed position on screen at all times, it needs to be defined relative to the camera. Three.js provides a method on any world object which converts from object relative coordinates to the lab frame, so this is used to trivially place the axes centre into the lab fram as required. Since this relationship changes when an object, in this case the camera, moves, the axes centre must therefore be redefined on any camera movement. This process also does not account for changes to intrinsic camera properties, importantly camera zoom. The axes therefore need to be scaled proportionally to the camera's zoom level on any zoom change.

Using the lab frame centre point and the axes vectors and scales, axis lines are trivially defined as,

$$l_0 = c \tag{4.4}$$

$$l_1 = c + szv \tag{4.5}$$



(a) (b)

Figure 4.3: Axes in WebMGA 3.0

where l_0 and l_1 are the axis line start and end, c is the axes centre, v is the axis vector, s is the axis scale factor, and z is the zoom factor of the camera. These can be recalculated and applied on every camera change. A Three.js Line object is constructed for each axis using the line start, end, and a colour.

During development of this feature, it was found that the axes would jump around the screen on camera movements rather than remain in the corner. Initially I believed this was an error in my implementation of the line origin calculation based on the camera position. The real cause of the problem was that the WebMGA camera update hook was incorrectly implemented and only called on certain types of camera movements. The position update function was changed to be called as part of the scene's "onBeforeRender" callback, provided as part of three.js, to ensure the axes are updated at every frame render immediately before drawing the scene (and always after any user input is processed). This mitigated the jumping issue.

Director

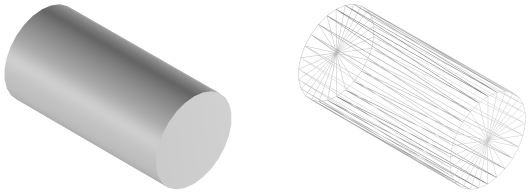
Plotting the director is made simple using the above setup. A new axis is simply defined using Equations (4.4) and (4.5), with v set to the already computed director vector.

Axes Colouring

It was decided that a meaningful colouring for the axes lines (including the director) would be using the same colour scheme as for molecule colour (Section 4.1). This can be done easily since all axes have a defined direction vector which can be passed to the "colour_from_director" function (Section 4.1.4). The resulting colour is simply passed as part of the Line object constructor.

Axes Summary

Some result can be viewed in Figure 4.3.



(a) Old model. (b) Old wireframe.

Figure 4.4: Cylinder

4.2.5 WebMGA 3.0 Bugs

The axes are currently lacking labels which would be useful for identifying their intended. Ideally, the end of each axis line should have a letter indicating x, y, z, or $\mathbf{n}$ (for the director).

4.3 Shapes

4.3.1 WebMGA 2.0 Implementation

WebMGA 2.0 implements the following molecule shapes:

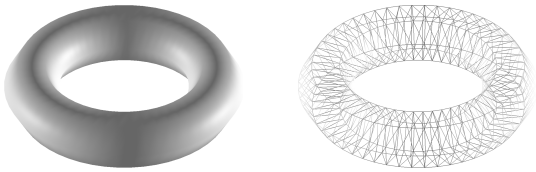
- Sphere (Figure 4.11)
- Ellipsoid (Figure 4.12)
- Spherocylinder (Figure 4.24)
- Spheroplatelet (Figure 4.13)
- Cut Sphere (Figure 4.16, implemented as a double cut sphere)
- Cylinder (Figure 4.4)
- Torus (Figure 4.5)

Notably missing but useful are the single cut sphere, the spherical cap, and the lens. The cylinder and torus shapes are present since the three.js library provides easily callable predefined meshes, however serve little practical purpose since no realistic molecular configuration would model using these.

4.3.2 WebMGA 2.0 Bugs

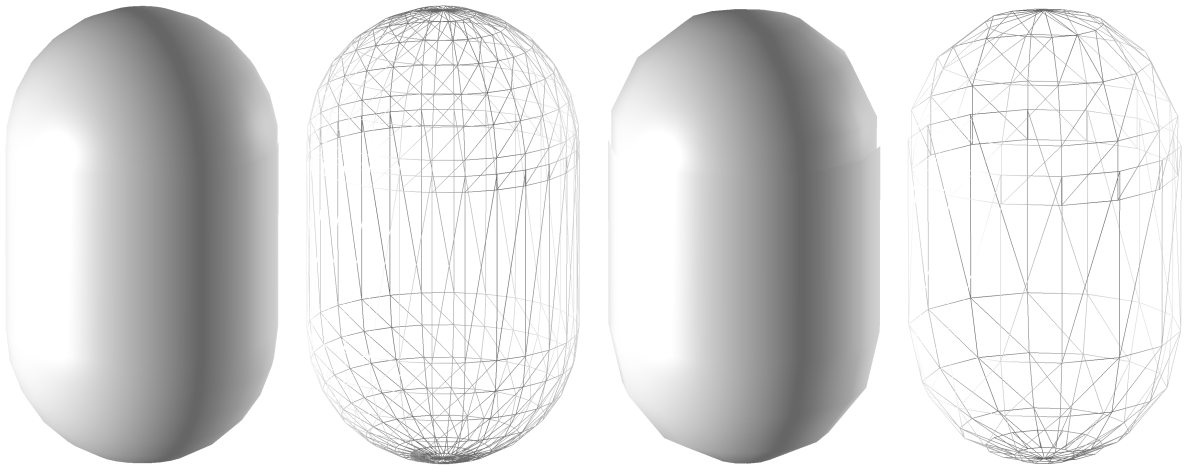
4.3.3 Improvement Goals

- Problem
  - Fix

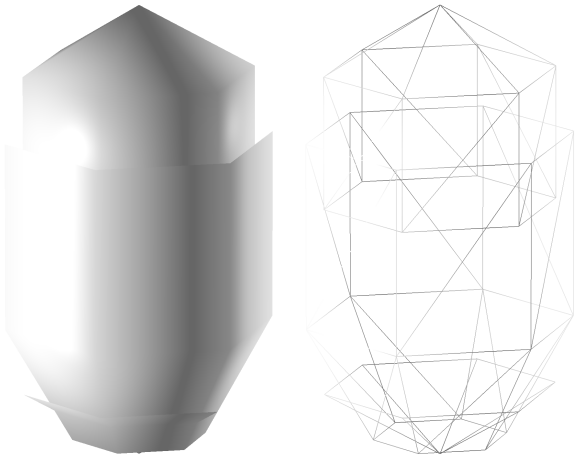


(a) Old model. (b) Old wireframe.

Figure 4.5: Torus

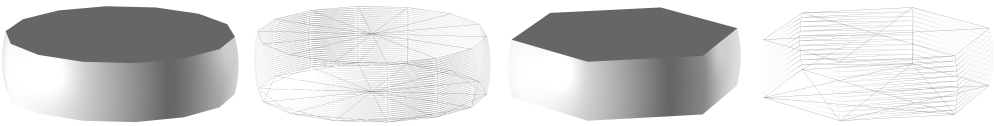


(a) High detail shape. (b) High detail mesh. (c) Mid detail shape. (d) Mid detail mesh.



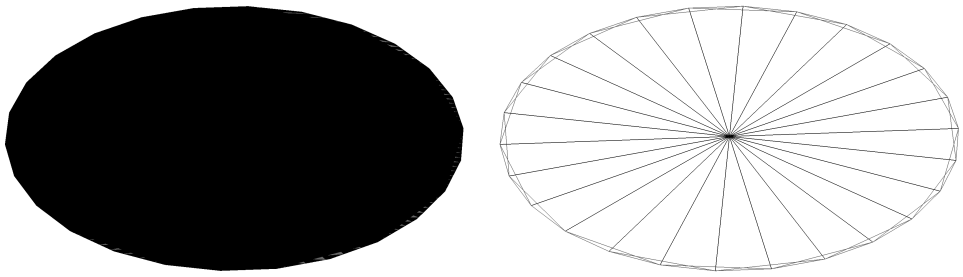
(e) Low detail shape. (f) Low detail mesh

Figure 4.6: Bad spheocylinder mesh generated by WebMGA 2.0 with various mesh qualities.

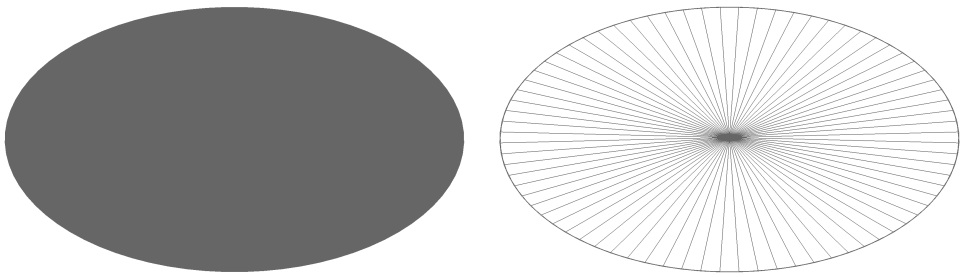


(a) WebMGA 2.0 (b) WebMGA 2.0 (c) WebMGA 2.0 (d) WebMGA 2.0

Figure 4.7: Notably higher mesh quality vertically for double cut sphere with WebMGA 2.0



(a) WebMGA 2.0 Shape (b) WebMGA 2.0 Wireframe



(c) WebMGA 3.0 Shape (d) WebMGA 3.0 Wireframe

Figure 4.8: Buggy shape representation when double cut sphere has 0 height

4.3.4 WebMGA 3.0 Implementation

Sphere

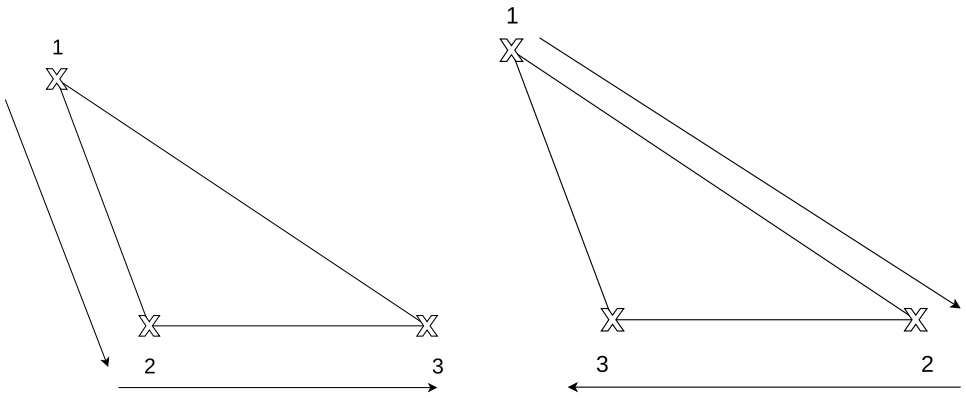
Key to the new shape implementations is the implementation for the sphere (Figure 4.11, parameter "Radius"). The sphere mesh is generated by sampling points across the sphere's surface in such a way as to split it into a finite number of flat, triangular sub-faces as shown in Figure 4.10. This sampling is performed with the spherical coordinates system for some sphere radius r, azimuthal angles θ, and polar angles ϕ, converted to an equivalent Cartesian form. A point in spherical coordinate space is denoted $\mathbf{r}_\mathrm{s}$, while a point in Cartesian space is denoted $\mathbf{r}_\mathrm{C}$,

$$\mathbf{r}_\mathrm{s} = \begin{pmatrix} r \\ \phi \\ \theta \end{pmatrix} \tag{4.6}$$

$$\mathbf{r}_\mathrm{C} = \begin{pmatrix} r\sin\phi\cos\theta \\ r\sin\phi\sin\theta \\ r\cos\phi \end{pmatrix}. \tag{4.7}$$

Any unique point on the origin centred r sphere can be uniquely defined by some (θ, ϕ) pair. Therefore, to evenly space points across the surface, a set of θs and ϕs is generated by taking n (essentially a measure of mesh quality) evenly spaced values over the interval of a full circular rotation ($[0, 2\pi)$). Each unique pairing (ϕ, γ), along with r, is used to produce the full set of Cartesian vertices using Equation (4.7). This method is sufficient to produce a sphere mesh as in Figure 4.11a from WebMGA 2.0. The sampling is modified slightly for WebMGA 3.0 to produce a mesh as in Figure 4.11c by offsetting each row such that points on one row lie half way between a pair of points on the row above since it produces a slightly more visually satisfying mesh. The code was rewritten from scratch since most other shapes result from slight modifications to the sphere generation process, and the initial WebMGA 2.0 implementation was over-complicated and proved difficult to extend.

Vertex Ordering: Since WebMGA 2.0 implements the back face culling optimisation[20] for mesh triangles, during the triangle generation process it must be ensured that vertices are arranged correctly to ensure the front face is on the exterior of the shape. When back face culling is used, the renderer will only shade triangles which it believes are facing towards the camera, and skip this for triangles facing away. The direction a triangle faces is defined by its winding. The two possible windings are shown in Figure 4.9. To ensure the triangle winding is correct during sphere generation, the vertex rows and columns are arranged such that, using the numbering from Figure 4.9, if vertex 1 is at index i on row j, then vertex 2 and 3 will be placed at indexes i and $i + 1$ on row



(a) Normal faces out of screen. (b) Normal faces into screen.

Figure 4.9: Triangle vertex ordering effect on normal direction (using right hand rule).

$j + 1$. When triangles are defined, they can be read from the vertex array in this order as appropriate.

Optimisation: Some optimisations are implemented to efficiently generate a full set of vertices while sampling only $\frac{1}{4}$ of the points around the sphere's surface. This uses the fact that the origin centred sphere is symmetrical in each of the xy, xz, and yz planes. Points for the quarter sphere can be generated by applying Equation (4.7) with all pairings of $\frac{n}{2}$ evenly spaced $\theta \in [0, \pi)$, and $\frac{n}{2}$ evenly spaced $\phi \in [0, \frac{\pi}{2}]$. These points are arranged in a 3d array corresponding to rows (from ϕ) and columns (from θ).

An additional set of points for the diagonally opposite quarter is trivially generated with correct vertex ordering by copying the original quarter vertices and negating the x and y values for each vertex to mirror in the xz and yz planes. TODO FINISH

TODO OPTIMISATION DIAGRAM

Ellipsoid

The ellipsoid shape (Figure 4.12, parameters "X", "Y", "Z") can be represented as an origin centred sphere of radius 1 scaled in the x,y and z directions by some scalar value in each direction. This can be represented by a slightly modified form of the Cartesian sphere equation in Equation (4.7), where $\mathbf{S}$ denotes some scaling vector,

$$\mathbf{r}_\mathrm{e} = \begin{pmatrix} s_x r \sin\phi \cos\theta \\ s_y r \sin\phi \sin\theta \\ s_z r \cos\theta \end{pmatrix} = \mathbf{S} \odot \mathbf{r}_\mathrm{C}. \tag{4.8}$$

From this formulation, it can be seen that an ellipsoid can be generated by slightly modifying the vertex sampling process for a sphere, whilst leaving the rest of the mesh building process unchanged. A sphere point can be sampled using Equation (4.7) with

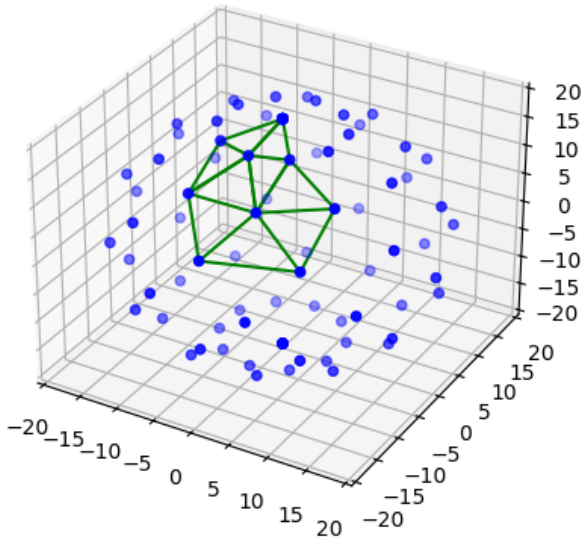
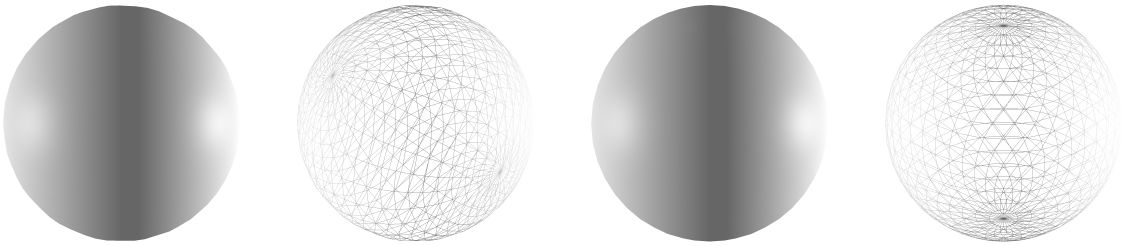
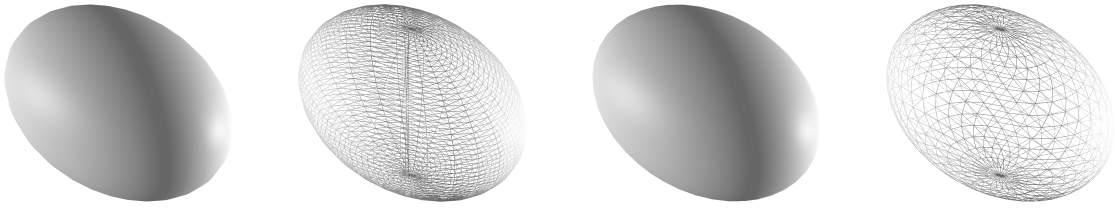


Figure 4.10: Example sphere vertex distribution (9×10 vertical×horizontal samples). Some mesh edges shown to demonstrate mesh construction from vertices.



(a) Old model. (b) Old wireframe. (c) New model. (d) New wireframe.

Figure 4.11: Sphere molecule mesh implementation.



(a) Old model. (b) Old wireframe. (c) New model. (d) New wireframe.

Figure 4.12: Ellipsoid molecule mesh implementation.

radius 1 and then multiplied by the scaling vector $\left(s_x, s_y, s_z\right)^\mathsf{T}$ to give an equivalent result to Equation (4.8).

In the program this is implemented by creating an "Ellipsoid" class as a child of the "Sphere" class and overriding the "sample_sphere()" method. Since this implementation is so simple, the JavaScript code is provided below:

```
//Ellipsoid mesh generator
export class Ellipsoid extends Sphere {
    //Scale factor in [x, y, z] directions
    scale: number[];

    constructor(x: number, y: number, z: number) {
        //Derive from origin centred sphere of radius 1
        super(1);
        this.scale = [x, y, z];
    }

    //Samples from ellipsoid instead of sphere
    sample_sphere(radius: number, theta: number, phi: number): number[] {
        //Multiply origin centred sphere coordinates by scale vector
        return math.dotMultiply(super.sample_sphere(radius, theta, phi), this.scale);
    }
}
```

Spheroplatelet

The spheroplatelet shape (Figure 4.13, parameters "RadSphere" and "RadCircle") is generated by modifying a generates sphere mesh (see Section 4.3.4). Vertices are iterated over and pushed outwards by applying to the below formula, following on from Equation (4.7) with c representing "RadCircle", $\mathbf{n}$ representing the sphere vertex normal in the x, y plane, and $\mathbf{r}_\mathrm{p}$ representing a vertex coordinate,

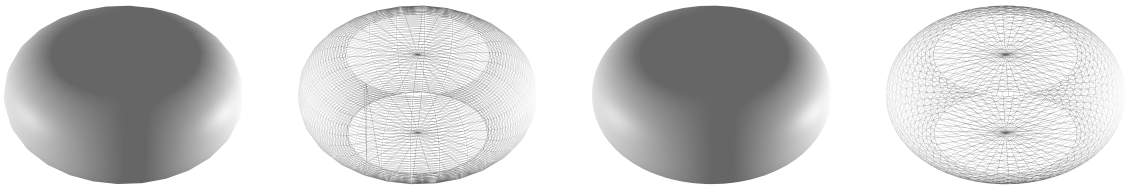
$$\mathbf{n} = \begin{pmatrix} r_x \\ r_y \end{pmatrix} \tag{4.9}$$

$$\mathbf{r}_\mathrm{p} = \mathbf{r}_\mathrm{C} + \frac{c\mathbf{n}}{\|\mathbf{n}\|_2}. \tag{4.10}$$

This will leave an empty circle of points at the top and bottom of the transformed geometry which can be filled by generating an additional vertices at the top and bottom respectively by averaging the coordinates for each vertex on the corresponding circle's edge, splitting it into triangles. This can be observed on Figure 4.13d.

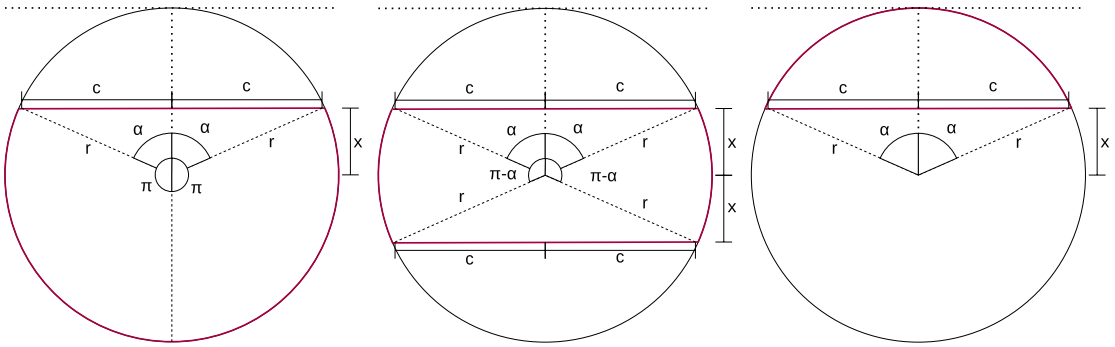
Cut Sphere

The cut sphere shape (Figure 4.15, parameters "Radius", "zCut") is implemented simply by sampling the sphere as before but over a reduced range of ϕ values. Since the sphere



(a) Old model. (b) Old wireframe. (c) New model. (d) New wireframe.

Figure 4.13: Spheroplatelet molecule mesh implementation.



(a) Cut sphere. (b) Double cut sphere. (c) Cap.

Figure 4.14: Cap and cut sphere shape diagrams. Lens outlines are shown by a red line, black lines demonstrate construction. Diagrams implemented by author using draw.io[21].

will not be completed, an empty circular face is left which can be filled by generating an additional vertex by averaging the coordinates for each vertex on the circle's edge, splitting it into triangles. This can be observed on Figure 4.15b.

A cut sphere is parameterised in WebMGA using a parent sphere radius and a zCut distance. These are shown in Figure 4.14a, with characters r and x respectively. The new range of ϕs can be seen in the diagram as the range $[\alpha, \pi)$, which can be reinterpreted in terms of r and x as follows,

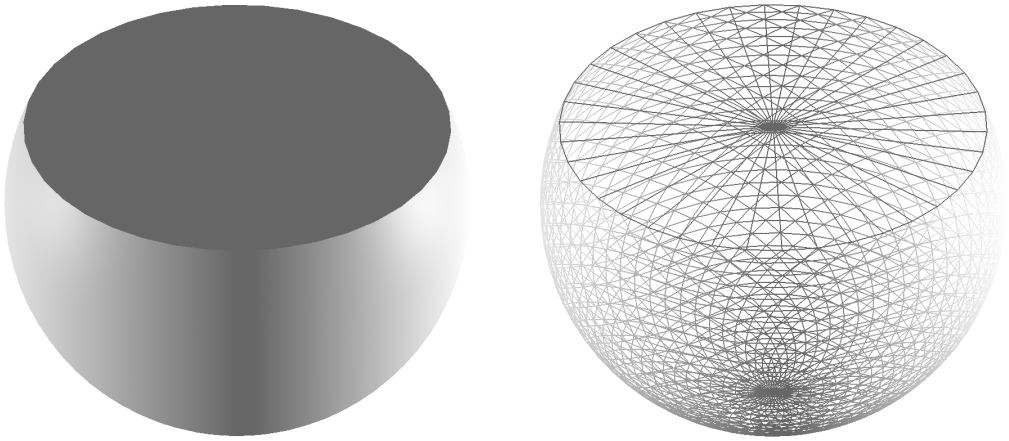
$$\alpha = \arcsin \frac{c}{r} \tag{4.11}$$

$$c = \sqrt{r^2 - x^2} \tag{4.12}$$

$$[\alpha, \pi) = \left[\arcsin \frac{c}{r}, \pi\right) \tag{4.13}$$

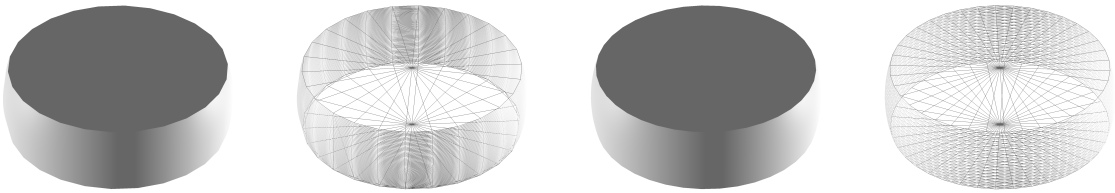
Double Cut Sphere

The double cut sphere shape (Figure 4.16, parameters "Radius", "zCut") is implemented simply by sampling the sphere as before but over a reduced range of ϕ values. Since the sphere will not be completed, two empty circular faces are left which can be filled by generating two additional vertices by averaging the coordinates for each vertex on the corresponding circle's edge, splitting it into triangles. This can be observed on Figure 4.16d.



(a) New model. (b) New wireframe.

Figure 4.15: Cut sphere



(a) Old model. (b) Old wireframe. (c) New model. (d) New wireframe.

Figure 4.16: Double cut sphere molecule mesh implementation.

A double cut sphere is parameterised in WebMGA using a parent sphere radius and a zCut distance. These are shown in Figure 4.14b, with characters r and x respectively. The new range of ϕs can be seen in the diagram as the range $[\alpha, \pi - \alpha)$, which can be reinterpreted in terms of r and x as follows,

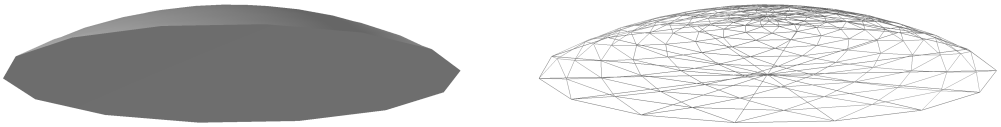
$$\alpha = \arcsin \frac{c}{r} \tag{4.14}$$

$$c = \sqrt{r^2 - x^2} \tag{4.15}$$

$$[\alpha, \pi - \alpha) = \left[\arcsin \frac{c}{r}, \pi - \arcsin \frac{c}{r}\right) \tag{4.16}$$

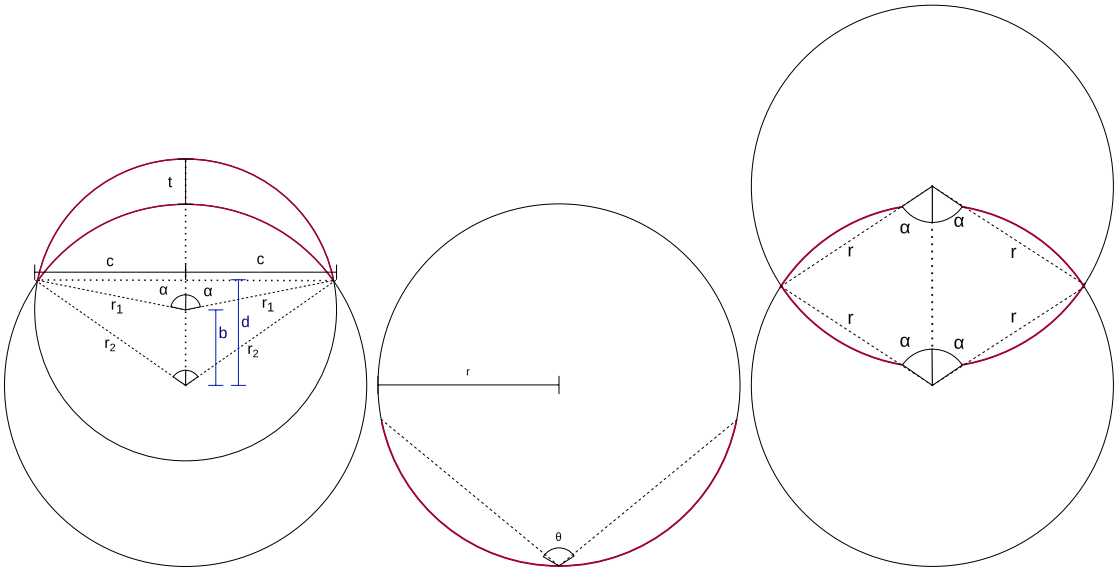
Cap

The cap shape (Figure 4.17, parameters "Radius", "zCut") is implemented simply by sampling the sphere as before but over a reduced range of ϕ values. Since the sphere will not be completed, an empty circular face is left which can be filled by generating an additional vertex by averaging the coordinates for each vertex on the circle's edge, splitting it into triangles. This can be observed on Figure 4.17b.



(a) New model. (b) New wireframe.

Figure 4.17: Cap



(a) Lens. (b) Cinacchi lens. (c) Biconvex lens.

Figure 4.18: Lens shape diagrams. Lens outlines are shown by a red line, black lines demonstrate construction. Diagrams implemented by author using draw.io[21].

A cap is parameterised in WebMGA using a parent sphere radius and a zCut distance. These are shown in Figure 4.14c, with characters r and x respectively. The new range of ϕs can be seen in the diagram as the range $[0, \alpha)$, which can be reinterpreted in terms of r and x,

$$\alpha = \arcsin \frac{c}{r} \tag{4.17}$$

$$c = \sqrt{r^2 - x^2} \tag{4.18}$$

$$[0, \alpha) = \left[0, \arcsin \frac{c}{r}\right). \tag{4.19}$$

Lens

The lens shape (Figure 4.19) is created by assembling either two caps (Section 4.3.4) or a cap and a a cut sphere (Section 4.3.4) to recreate the format shown in Figure 4.18a. For the concave part of the lens, a cap is generated with angle parameter α. For the concave part of the lens, if the required θ is greater than $\frac{\pi}{2}$ then a cut sphere is used, else a cap. The flat face of both parts is removed and the concave part is transformed such

that the circular edge matches the circular edge of the convex part. Both parts are then transformed such that the lab frame origin is located at the pole of the convex half of the lens.

For the concave part of the lens, it will appear invisible due to back face culling since the triangle normals face outwards by default which will be the inside of the lens. To mitigate this, ordering for rows of vertices must be reversed in the vertex array to move from the ordering in Figure 4.9b to an ordering matching Figure 4.9a.

In code, the lens and cut sphere are parameterised in terms of a radius and a cut circle circumference. These correspond to (r_1, c), (r_2, c) in Figure 4.18a for the concave and convex parts respectively. There are a few possible parameterisations which can define a lens, from which required values are derived. The two implemented by WebMGA are discussed below.

Base Lens: This parameterisation consists of two radii (r_1, r_2 in Figure 4.18a) and an opening angle (α in Figure 4.18a). r_1 and r_2 are trivially the two radii provided, while c is defined as,

$$c = r_1 \sin \alpha \tag{4.20}$$

This parameterisation is implemented in the code as the "BaseLens" class since it was the easiest to implement the previously described vertex generation with, however is not shown to the user since it did not seem as intuitive to set up as the thick lens parameterisation.

Thick Lens: The thick lens is WebMGA's default parameterisation for the lens as shown for the user. It is parameterised as "Radius" (r_1 in Figure 4.18a), "Thickness" (t in Figure 4.18a), and angle ((α in Figure 4.18a). It is implemented as a subclass of "BaseLens" which generates its parameters from those provided. r_1 and α are given. r_2 is derived as follows (using c derived from Equation (4.20)),

$$b = r_2 + t - r_1 \tag{4.21}$$

$$d = b + r_1 \cos \alpha \tag{4.22}$$

$$r_2^2 = c^2 + d^2 \tag{4.23}$$

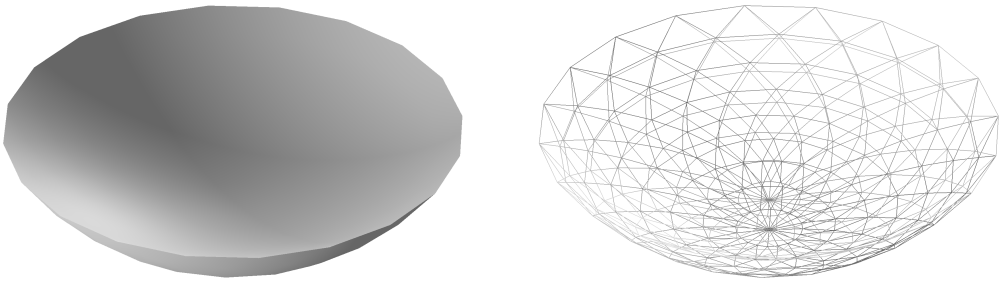
$$r_2^2 = r_1^2 \sin^2 \alpha + (b + r_1 \cos \alpha)^2 \tag{4.24}$$

$$r_2^2 = r_1^2 \sin^2 \alpha + b^2 + r_1^2 \cos^2 \alpha + 2br_1 \cos \alpha \tag{4.25}$$

$$r_2^2 = r_1^2(\sin^2 \alpha + \cos^2 \alpha) + b^2 + 2br_1 \cos \alpha \tag{4.26}$$

$$r_2^2 = r_1^2 + b^2 + 2br_1 \cos \alpha \tag{4.27}$$

$$r_2^2 = r_1^2 + (r_1^2 + r_2^2 + t^2 - 2r_1r_2 - 2r_1t + 2r_2t) + 2(r_2 + t - r_1)r_1 \cos \alpha \tag{4.28}$$



(a) New model. (b) New wireframe.

Figure 4.19: Lens

$$2r_1r_2 - 2r_2t - 2r_2r_1\cos\alpha = 2r_1^2 + t^2 - 2r_1t + 2(t - r_1)r_1\cos\alpha \tag{4.29}$$

$$2r_2(r_1(1-\cos\alpha) - t) = 2r_1^2(1-\cos\alpha) + t^2 + 2tr_1(\cos(\alpha) - 1) \tag{4.30}$$

$$r_2 = \frac{2r_1^2(1-\cos\alpha) + 2tr_1(\cos(\alpha) - 1) + t^2}{2(r_1(1-\cos\alpha) - t)}. \tag{4.31}$$

Cinacchi Lens

During development, some sample configurations requiring the lens molecule shape were provided by Giorgio Cinacchi. This is named the "Cinacchi Lens" in WebMGA (parameters "Radius"). For these configurations, Cinacchi uses a specific lens configuration as shown in Figure 4.18b which is parameterised using only a single r value,

$$\cos\alpha = 1 - \frac{1}{2\pi r^2} \tag{4.32}$$

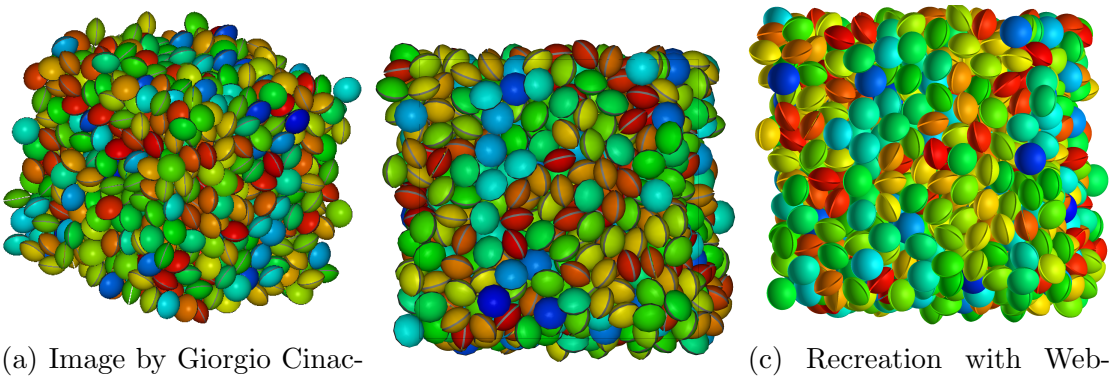
$$\alpha = \arccos\left(1 - \frac{1}{2\pi r^2}\right). \tag{4.33}$$

This produces an infinitely thin lens with some aperture angle dependent on the radius. The Cinnachi lens is implemented simply as a parameterisation of the base lens in Section 4.3.4 where the two radii are both r, and the angle is derived from Equation (4.33).

A screenshot produced using QMGA was provided by Cinacchi to assist in visually verifying the shape produced. This is shown in Figure 4.20a. A recreation was produced in QMGA as shown in Figure 4.20b, then WebMGA as shown in Figure 4.20c. This appears to verify a correct implementation.

Biconvex Lens

The biconvex lens shape (Figure 4.21, parameters "Radius", "Angle", "Separation") is implemented as a specific inititalisation of the "BaseLens"(Section 4.3.4) where $r_2 = -r_1$.

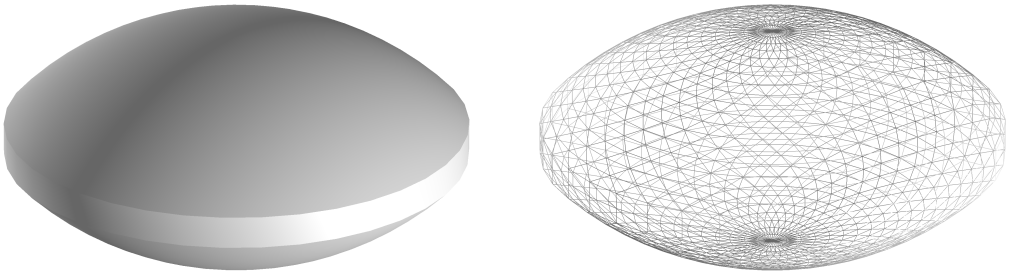


(a) Image by Giorgio Cinacchi.

(b) Recreation with QMGA.

(c) Recreation with WebMGA.

Figure 4.20: Lens setup required by Giorgio Cinacchi.

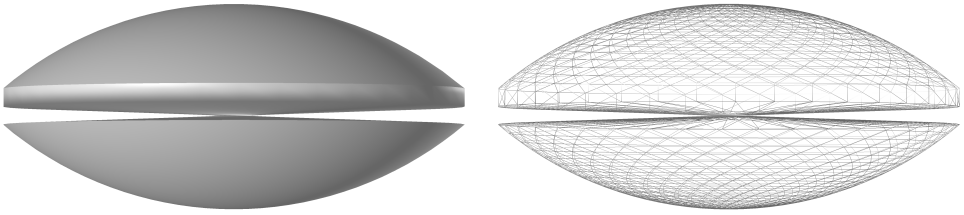


(a) New model.

(b) New wireframe.

Figure 4.21: Biconvex lens

Since the configurations provided by Cinacchi (Figure 4.20a) aimed to emulate a biconvex lens using two separate lenses, but exhibited a small gap between the two parts, it was decided that a separation parameter would also be provided. This is implemented by moving the top part up and bottom part down by half the separation value. A row of vertices is inserted between to split the flat edge into triangles as seen in Figure 4.21b. A complication encountered is that by default the bottom half of the lens is not aligned correctly which results in a twisting effect visible in Figure 4.22, so all vertex rows must be shifted such that the correct configuration applies for triangle generation.



(a) Shape.

(b) Wireframe.

Figure 4.22: Incorrectly configured biconvex lens where halves are incorrectly aligned.

Spherocylinder

The spherocylinder shape (Figure 4.24, parameters "Radius", "Length") can be represented as an origin centred sphere of radius r scaled in the z directions by (half of) some length value in each z direction (positive/negative). This can be represented by a slightly modified form of the Cartesian sphere equation in Equation (4.7) (where l denotes stretch length),

$$\mathbf{r}_{\mathrm{c}} = \begin{pmatrix} r\sin\phi\cos\theta \\ r\sin\phi\sin\theta \\ r\cos\theta + n \end{pmatrix} = \mathbf{r}_{\mathrm{C}} + \begin{pmatrix} 0 \\ 0 \\ n \end{pmatrix} \tag{4.34}$$

$$n = \begin{cases} \frac{l}{2} & \text{if } r\cos\theta > 0 \\ -\frac{l}{2} & \text{if } r\cos\theta < 0 \\ 0 & \text{otherwise.} \end{cases} \tag{4.35}$$

.

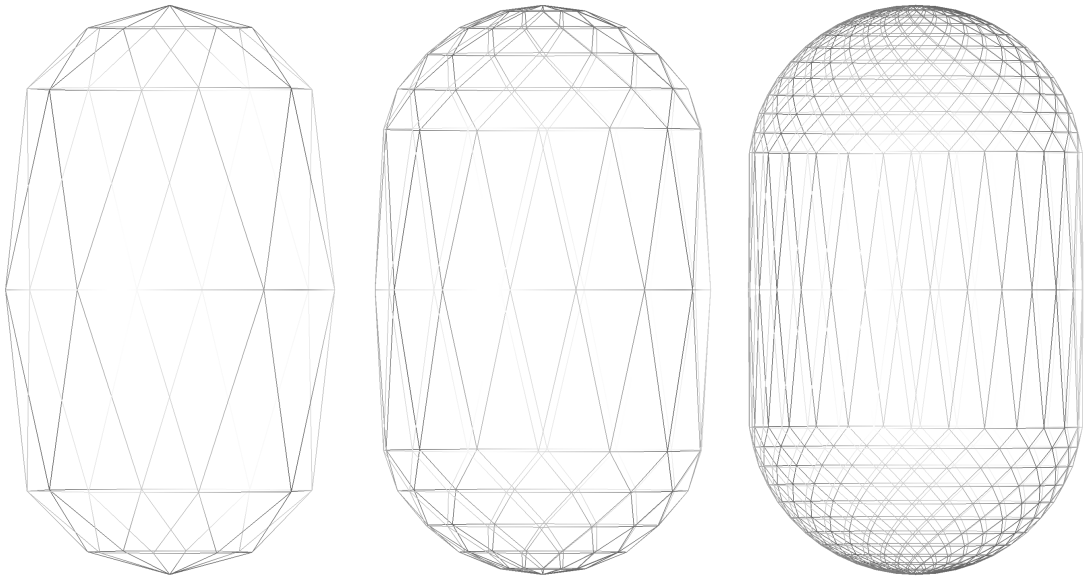
Initial Attempt From Equations (4.34) and (4.35), it can be seen that a spherocylinder can be approximated by slightly modifying the vertex sampling process for a sphere, whilst leaving the rest of the mesh building process unchanged. A sphere point can be sampled using Equation (4.7) with radius r and added to the scaling vector $\left(0, 0, n\right)^{\mathsf{T}}$ as defined in Equation (4.35) to give an equivalent result to Equation (4.34).

In the program this was implemented by creating a "Spherocylinder" class as a child of the "Sphere" class and overriding the "sample_sphere()" method. Since this implementation is so simple, the JavaScript code is provided below:

```
//Spherocylinder mesh generator
export class Spherocylinder extends Sphere {
    //Scaling vector (either side of centre) to stretch sphere into spherocylinder ([0, 0, length / 2])
    length_scaling_vector: number[];

    constructor(radius: number, length: number) {
        //Derive from origin centred sphere of chosen radius
        super(radius);
        this.length_scaling_vector = [0, 0, length / 2];
    }

    //Samples from spherocylinder instead of sphere
    sample_sphere(radius: number, theta: number, phi: number, epsilon: number = 1e-15): number[] {
        let sphere_coordinate: number[] = super.sample_sphere(radius, theta, phi);
        //Stretch point in z direction by scale vector, matching stretch direction to sign of original vertex z
        //Unchanged if z is (approximately) 0
        if (Math.abs(sphere_coordinate[2]) < epsilon) {
        } else if (sphere_coordinate[2] > 0) {
            sphere_coordinate = math.add(sphere_coordinate, this.length_scaling_vector);
        } else if (sphere_coordinate[2] < 0) {
            sphere_coordinate = math.subtract(sphere_coordinate, this.length_scaling_vector);
        }
        return sphere_coordinate;
    }
}
```



(a) Low mesh density. (b) Medium mesh density. (c) High mesh density.

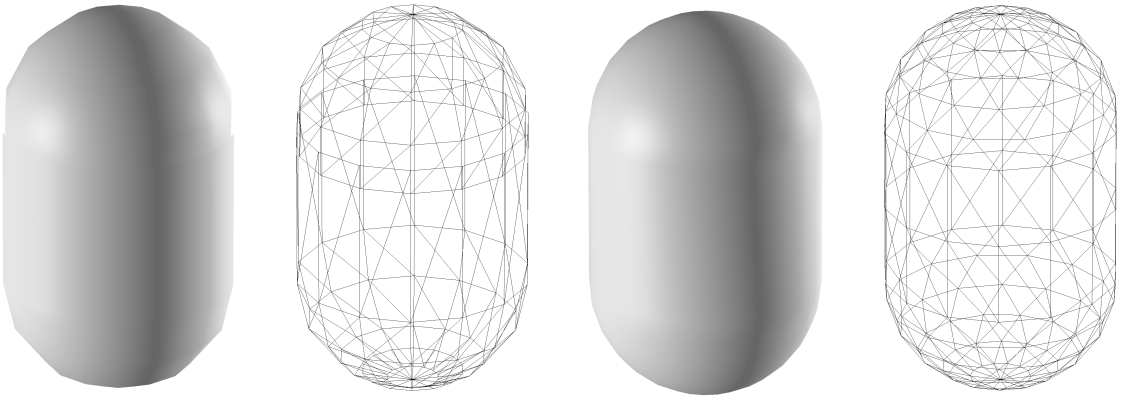
Figure 4.23: Initial spherocylinder implementation. Visible tapering can be observed, particularly with low mesh density.

Unfortunately, this process produced visually unsatisfying results with the sides of the spherocylinder visibly tapering, particularly with low detail meshes. This can be seen in Figure 4.23. After producing the biconvex lens (Section 4.3.4), an alternate, much simpler solution became apparent which avoided this issue.

Second Attempt A spherocylinder can also be considered a special case of the biconvex lens. A biconvex lens with no separation and aperture angle $\frac{\pi}{2}$ produces a sphere with the given radius r. Increasing the separation parameter causes the two hemispheres to move apart such that a spherocylinder is produced. The spherocylinder can therefore simply be considered a special case of the biconvex lens with aperture angle $\frac{\pi}{2}$, and can be implemented entirely through class inheritance as shown:

```
//Spherocylinder mesh generator
export class Spherocylinder extends BiconvexLens {
    constructor(radius: number, length: number) {
        super(radius, Math.PI / 2, length);
    }
}
```

This produced the result shown in Figure 4.24.



(a) Old model. (b) Old wireframe. (c) New model. (d) New wireframe.

Figure 4.24: Spherocylinder molecule mesh implementation.

4.3.5 WebMGA 3.0 Bugs

Most shapes were successfully implemented fully and appear as expected. The shading on the spherocylinder in Figure 4.24c appears slightly incorrect at the boundary between the curved section and the flatter section, however this is more subtle than WebMGA 2.0's incorrect spherocylinder mesh. A fix for this should be investigated. It likely arises due to some error in calculation of vertex normals.

4.4 File types

4.4.1 WebMGA 2.0 Implementation

WebMGA 2.0 supports only its own JSON-based file format as defined by Battistini. This could prove an obstacle to users since, in practice, different formats are output when running molecular dynamics simulations.

4.4.2 WebMGA 2.0 Bugs

N/A

4.4.3 WebMGA 3.0 Implementation

WebMGA 3.0 implements two new file formats for defining molecular configurations as defined below.

CNF Format (.cnf)

The CNF format is widely used for representation of molecular configurations. LAMMPS is one example of a molecular dynamics simulator which uses this, typically used on highly

Molecule count												
Unit box X length (lx)												
Unit box Y length (ly)												
Unit box Z length (lz)												
Not used	Not used											
Position (rx)	Position (ry)	Position (rz)	Velocity (vx)	Velocity (vy)	Velocity (vz)	Orientation (ex)	Orientation (ey)	Orientation (ez)	Orientational velocity (ux)	Orientational velocity (uy)	Orientational velocity (uz)	Molecule ID
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 4.1: CNF format molecule configuration.

Unit box X half length ($lx/2$)	Unit box Y half length ($ly/2$)	Unit box Z half length ($lz/2$)				
Shape parameter	Position X (rx)	Position Y (ry)	Position Z (rz)	Orientation X (ex)	Orientation Y (ey)	Orientation Z (ez)
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 4.2: Cinacchi format molecule configuration.

parallel computers[22]. The CNF format is specifically designed to allow the highest possible performance while preserving some amount of human readability.

Table 4.1 shows the structure of a file of this format. Rows represent lines in the file. Each value is represented by a signed float of format -1.000000, where digits before the decimal are omitted if not present. Values are separated by spaces, padded to align decimal points.

For WebMGA 3.0, a parser script was written in JavaScript which builds a WebMGA JSON configuration from the ".cnf" file provided. Specifically, a unit box is constructed from (lx, ly, lz), and molecule positions and orientations are obtained from corresponding pairs of $((rx, ry, rz), (ex, ey, ez))$ for some molecule id. All other parameters are dropped since they aren't used by WebMGA. Molecules are ordered in an array according to their id.

Cinacchi Format (.qmga)

See Table 4.2 for the structure of a file of this format. Rows represent lines in the file. Each value is represented by a signed float of format -1.00000000, where digits before the decimal are omitted if not present. Values are separated by spaces, padded to align decimal points.

For WebMGA 3.0, a parser script was written in JavaScript which builds a WebMGA JSON configuration from the ".qmga" file provided. Specifically, a unit box is constructed from (lx, ly, lz), and molecule positions and orientations are obtained from corresponding pairs of $((rx, ry, rz), (ex, ey, ez))$. The shape parameter is dropped since molecule shape is not defined by the file. Molecules are ordered in an array as they are encountered.

4.4.4 WebMGA 3.0 Bugs

WebMGA ignores the shape parameter from the ".qmga" format configuration. Since some shapes in WebMGA require multiple parameters, and the shape to use is not defined within the file, I could not see a sensible way to automate applying this. The user must

manually enter this value after selecting a molecule shape in the "Models" menu. This is not ideal since a user should expect their configuration to appear correctly as soon as they load the file.

4.5 Periodic Repetition

4.5.1 Improvement Goals

A description for periodic boundary conditions is given in Section 2.2.2 regarding how a small simulation box simulates a subset of an infinite lattice. It may be useful to visualise a larger subset of this infinite lattice by repeating the simulation box a number of times. Additionally, the capability of repeating a smaller system is useful for producing a realistic, much larger configuration for testing the performance of WebMGA with increased molecule counts due to the lack of availability of real test configurations of such sizes.

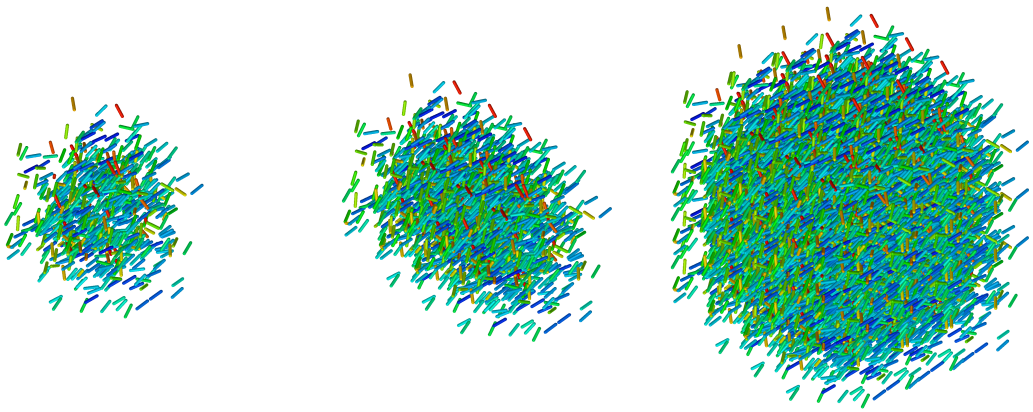
4.5.2 WebMGA 3.0 Implementation

A few modifications needed to be made to implement this feature.

First, the "reference" tab in the side menu was modified to include inputs for the repeat count in the 'x' 'y' and 'z' directions. With a value of zero, there should be only a single instance of the configuration along the corresponding axis. Setting to one adds a repeat in the positive and negative direction along the axis, with bounding box faces touching (i.e. with a value of 0, there will be 1 instance of the configuration, with 1 there will be 3 instances, 2 there will be 5 instances etc.). When multiple directions have a value larger than 0, the configurations are repeated such that a single large box is produced (i.e. it is ensured there are no gaps, for example a configuration of $(1, 1, 1)$ will give a box of dimensions $3 \times 3 \times 3$ with 27 total instances of the initial configuration).

Repetition of the configuration was implemented by changing the "setState" method. After building a configuration, the repeats parameters are read and each molecule is duplicated $x \times y \times z$ times according to the following pseudocode where 'unitBoxSize' is a 3D vector defining half each dimension of the configuration's unit box,

```
FOR molecule of molecules
  FOR x=-X to X
    FOR y=-Y to Y
      FOR z=-Z to Z
        newMolecule = molecule.clone
        newMolecule.position += unitBoxSize * [x, y, z]
        scene.add(newMolecule)
      end FOR
```



(a) $(0, 0, 0)$ (b) $(1, 0, 0)$ (c) $(1, 1, 1)$

Figure 4.25: Demonstration of periodic repetition of a configuration, labelled with repetition parameter of format (x, y, z).

```
        end FOR
    end FOR
end FOR
```

.

After implementing this feature, it was found that when changes were made to the configuration the duplicated molecules were not updated. This was fixed by modifying the molecule set generation function to remove and then reduplicate the extra molecules upon changes.

To enable the user to change repetitions, a 'Repeats' heading was added to the 'Reference' sub menu with X, Y and Z values set to 0 by default. The user can enter any integer value and the configuration updates as need.

4.5.3 WebMGA 3.0 Bugs

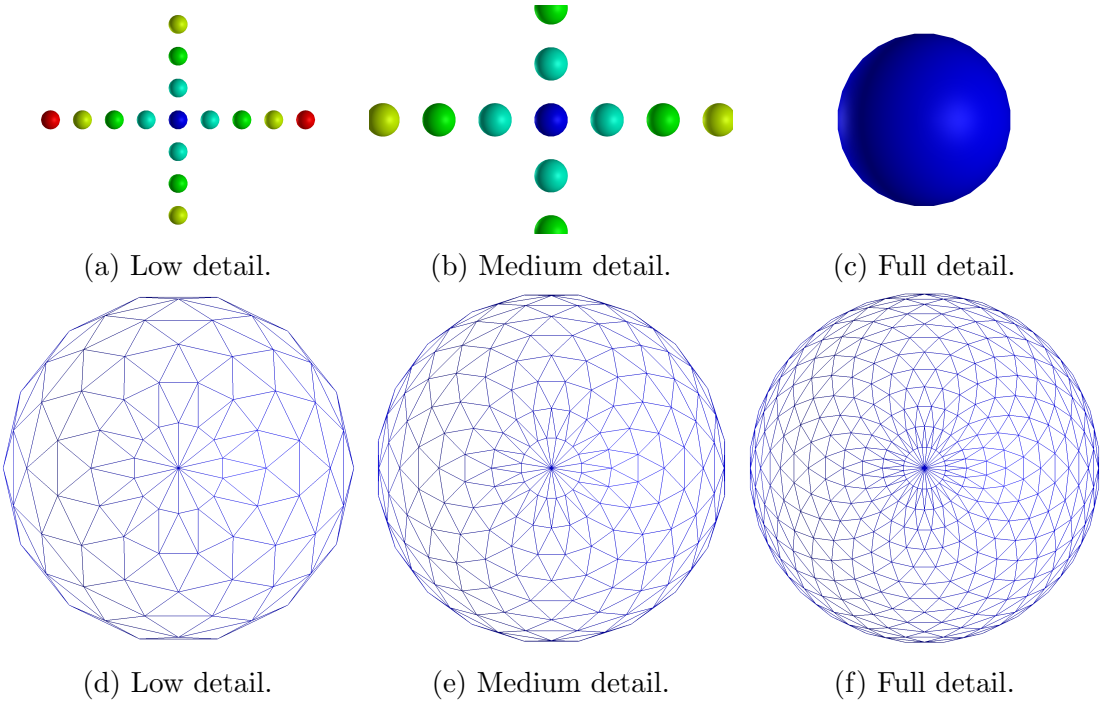
TODO

4.6 Optimisations

4.6.1 WebMGA 3.0 Implementation

Distance Based Levels of Detail

WebMGA 3.0 now utilises distance based variable LOD. This is implemented using three.js' built in LOD class [23] which acts as if it were a regular three geometry, except three.js automatically switches meshes at set camera distances. Meshes are generated at



(a) Low detail. (b) Medium detail. (c) Full detail.

(d) Low detail. (e) Medium detail. (f) Full detail.

Figure 4.26: Model complexity is decreased at subjectively chosen camera distance thresholds with minimal visible loss in quality.

each LOD level up to the Level of Detail setting selected by the user and added to the LOD object with some distance as shown by this code,

```
lod_object.addLevel(mesh, distance)
```

.

Lower detail meshes are used at further distances, with the distances selected by manual tuning such that the geometry change is difficult to observe. Examples of this works are shown in Figures 4.26 and 4.27.

Performance analysis for this optimisation is discussed in Section 5.1.

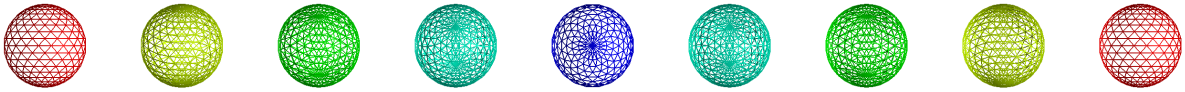
4.6.2 WebMGA 3.0 Bugs

TODO

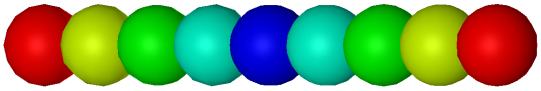
4.7 Miscellaneous Improvements

- Various GUI and model state synchronisation issues fixed
  - Bounding box now updates correctly on new model loading
  - Model update function fixed to run immediately before every frame update rather than at arbitrary intervals (avoid delayed state changes)

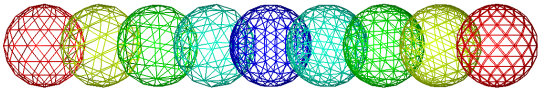
(a) At a similar distance, all objects are identical.



(b) At a similar distance, all meshes are identical.



(c) Further spheres have a minor decrease in visible quality.



(d) Further spheres have a minor decrease in triangle density.

Figure 4.27: Demonstration of decreased mesh quality for distant object. "Level of Detail" setting has been reduced below default for a more visible geometry reduction.

- Outdated and vulnerable dependencies updated
- Fixed various type inconsistencies in code which could result in unexpected behaviour and added additional validation checks
- Changed shape geometry complexity increments (Level of Detail slider) to a base 2 logspace to scale with more sensible intervals
- Fixed the scene update function to be called immediately before any frame render rather than at an arbitrarily selected interval (as part of a bug fix for Section 4.2.4)

Chapter 5

Analysis and Testing

5.1 Level of Detail Performance

5.2 Configuration Visualisations

LOD Enabled	Mesh Quality	Repeats	Distant Framerate (7)	Nearby Framerate (50)	Nearest Framerate (100)
True	Highest	$(0, 0, 0)$	64.74	33.37	35.79
False	Highest	$(0, 0, 0)$	19.84	16.76	23.62
True	Default	$(0, 0, 0)$	75.7	46.83	41.83
False	Default	$(0, 0, 0)$	74.78	35.23	36.89
True	Lowest	$(0, 0, 0)$	56.38	36.78	42.66
False	Lowest	$(0, 0, 0)$	66.34	31.75	38.28
True	Highest	$(1, 1, 1)$	5.12	8.54	8.31
False	Highest	$(1, 1, 1)$	0.93	2.26	4.45
True	Default	$(1, 1, 1)$	5.41	7.55	8.37
False	Default	$(1, 1, 1)$	10.74	10.60	13.90
True	Lowest	$(1, 1, 1)$	5.22	7.27	8.03
False	Lowest	$(1, 1, 1)$	9.97	17.29	20.83

Table 5.1: Distance based variable level of detail performance analysis results. Performed using the "Unfolded SC4 Nematic" configuration shown in Figure 1.1a at 3 zoom levels, with 3 mesh qualities, and with 2 periodic repetition settings (see Section 4.5).

Chapter 6

Conclusions and Evaluation

6.1 Achievements

WebMGA 3.0 has successfully, and without any regressions, implemented a range of feature extensions and bugfixes to WebMGA 2.0. A larger range of useful molecule geometries are now available: the cut sphere, the spherical cap, the generic lens, the Cinacchi lens, and the biconvex lens.

Axes have been reimplemented to appear more clearly to the user, unobstructed by the molecular configuration, with an additional line now indicating the director and all lines coloured to indicate director alignment.

New filetype support (CNF and Cinacchi format) has been included to allow directly loading configurations directly from real molecular simulation applications, reducing an obstacle to adoption by researchers who no longer need to convert to WebMGA's previous application specific json-based format.

Periodic repetition settings are now included which allow for visualisation of a larger segment of a bulk liquid which has been simplified using periodic boundary conditions.

A distance-based variable level of detail optimisation has been included which allows for more responsive visualisation of very large configurations with high geometry quality settings with minimal perceivable visual degradation.

Various miscellaneous bugfixes were also implemented which should result in a generally improved user experience, particularly regarding UI setting and model synchronisation.

Some bugs and additional useful features which would result in a more polished user experience still persist and are discussed in Section 6.3.

6.2 Evaluation

Overall WebMGA works well for performing its required visualisation tasks, with superior visual results compared to QMGA. Sufficiently high framerates are achieved to allow for responsive user interaction with reasonably sized configurations.

There are some substantial shortcomings in the implementation which are worth addressing. One of the key problems encountered during development was a lack of comments within the existing code, making it very difficult to determine what many sections were intended to do. Some additional comments were added, however it would still be difficult for a new developer to easily pick up the project. A particularly prevalent problem in the code is that many functions take in a tuple of values or a json format as an input, rather than being separate function parameters. These formats were rarely if ever documented, significantly obfuscating how these methods operate or should be called, and required manual debug inspection to determine their required format.

There were many minor bugs throughout the program which resulted in a desynchronisation between the state selected in the UI and the render shown. While I was able to patch many of these relatively simply by fixing minor logic errors in the code, it is difficult to determine if any of these still exist since there are a large number of setting combinations possible and some of these bugs existed only with very specific combinations. These issues seem to result from the Model-Controller-View structure WebMGA was designed around not being entirely successfully implemented, with the controller failing in some cases to update the Model (graphics render) on all View (GUI setting) updates. The state of certain configuration settings is often stored separately in all three of these parts of the program, resulting in two possible stages for desync to occur between View and Controller, and Controller and View. State should have been consistently stored and referenced only from the Controller.

6.3 Future Work

6.3.1 Axis Labelling

Axes remain unlabelled. It would be clearer if these were labelled with text x, y, z, and $\mathbf{n}$, with $\mathbf{n}$ representing the director.

6.3.2 Colour Palette Guide

Currently, there is no visual guide for a molecule's colour derived from its principal axis angle with the director. While this can be seen relatively intuitively, a colour palette should be displayed somewhere in the UI which shows a mapping of colour to angles (in the range $[0, \pi)$).

6.3.3 Colour Palette Adjustment

The current palette for molecule colouring based on its principal axis' director alignment assumes a colour range with hue 0 (red) through to $\frac{-4\pi}{3}$ (blue). A different range is easily obtained with the current current colour sampling code by simply giving a different start and end point. This could be added to the GUI to allow for specifying some custom colouring scheme if needed.

6.3.4 Bug Fixes

A thorough review of as many setting combinations as possible should be performed to identify further desynchronisation conditions between the visualisation shown and changed settings.

Bibliography

[1] Eduardo Battistini. Webmga, 2021. URL: https://students.cs.ucl.ac.uk/2019/group3/WebMGA/diss.pdf.

[2] Yue He. Webmga 2.0, 2023.

[3] Adrian T Gabriel, Timm Meyer, and Guido Germano. Molecular graphics of convex body fluids. *Journal of Chemical Theory and Computation*, 4(3):468–476, 2008.

[4] Webmga 3.0. URL: https://joe-down.github.io/WebMGA-3.

[5] Webmga 3.0 github repository. URL: https://github.com/joe-down/WebMGA-3.

[6] React. URL: https://react.dev/.

[7] Three.js. URL: https://threejs.org/.

[8] Qmga sourceforge.net. URL: https://sourceforge.net/projects/qmga/files/qmga/.

[9] Juan Pedro Ramírez González and Giorgio Cinacchi. Densest-known packings and phase behavior of hard spherical capsids. *The Journal of Chemical Physics*, 159(4), 2023.

[10] Valerio Mazzilli, Katsuhiko Satoh, and Giacomo Saielli. Phase behaviour of mixtures of charged soft disks and spheres. *Soft Matter*, 19(18):3311–3324, 2023.

[11] Richard James, Eero Willman, FA FernandezFernandez, and Sally E Day. Finite-element modeling of liquid-crystal hydrodynamics with a variable degree of order. *IEEE Transactions on Electron Devices*, 53(7):1575–1582, 2006.

[12] LCview. URL: https://www.ee.ucl.ac.uk/~afernand/rjames/modelling/visualisation/.

[13] Qt 3 Debian removal. URL: https://wiki.debian.org/qt3-x11-freeRemoval.

[14] Michael P Allen and Dominic J Tildesley. *Computer Simulation of Liquids*. Oxford University Press, Oxford, 2nd edition, 2017.

[15] Ronald Y Dong. Orientational order. *Nuclear Magnetic Resonance of Liquid Crystals*:53–89, 1997.

[16] Weidong Wu, Joseph Owino, Ahmed Al-Ostaz, and Liguang Cai. Applying periodic boundary conditions in finite element analysis. In *SIMULIA community conference, Providence*, pages 707–719, 2014.

[17] Michael P Allen. Molecular simulation of liquid crystals. *Molecular Physics*, 117(18):2391–2417, 2019.

[18] Giorgio Cinacchi and Salvatore Torquato. Hard convex lens-shaped particles: characterization of dense disordered packings. *Physical Review E*, 100(6):062902, 2019.

[19] Three.js color management. URL: `https://threejs.org/docs/#manual/en/introduction/Color-management`.

[20] Opengl face culling. URL: `https://www.khronos.org/opengl/wiki/Face_Culling`.

[21] draw.io. URL: `https://www.drawio.com/`.

[22] Aidan P Thompson, H Metin Aktulga, Richard Berger, Dan S Bolintineanu, W Michael Brown, Paul S Crozier, Pieter J In't Veld, Axel Kohlmeyer, Stan G Moore, Trung Dac Nguyen, et al. Lammps-a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales. *Computer Physics Communications*, 271:108171, 2022.

[23] Three.js lod. URL: `https://threejs.org/docs/#api/en/objects/LOD`.

Appendix A

Appendices

A.1 Project proposal

Project Plan
WebMGA 3.0: Refinement of an Interactive Viewer for Coarse-Grained Liquid Crystal Models

Joe Down

10th November 2023

Abstract

Produce, document, and benchmark an enhanced version of an existing web-based tool for visualising coarse-grained liquid crystal models. Provide insights into related computer graphics performance and optimisation.

1 Supervisor Information

- Guido Germano, Professor of Computational Science
- g.germano@ucl.ac.uk

2 Aims and Objectives

- Fulfill any further requirements from relevant academic stakeholders.
- Identify and correct unexpected or unintuitive program behaviour.
- Address missing features and shortcomings identified by previous dissertations.
- Implement trivial usability enhancements (labels and scales etc.)
- Test using new configurations such as those using other specialised particle shapes or much larger molecule counts (attained from academic stakeholders or perhaps through generation).
- Identify performance bottlenecks and attempt to identify and apply optimisations. Analyse and compare performance resulting from enhancements.
- Comment on performance and effectiveness of optimisations (e.g. backface culling) in the particular case of configurations consisting of a very large numbers of molecules (implement new benchmark in place of unrealistic existing one).
- Investigate efficiency of different structures for storing molecular configurations (e.g. LAMMPS .cnf format vs current .json format).

3 Expected Outcomes / Deliverables

- A working, enhanced, version of WebMGA featuring no feature regressions.
- Documentation to allow further development of WebMGA.
- Reporting on optimisations to the WebMGA rendering process and their performance impact, with the intention to provide more general comments on computer graphics in other similar situations.
- Feedback from relevant academic stakeholders.
- New molecular configurations for other relevant scenarios.
- Strategies for testing and evaluating contributions.
- Sufficient documentation of any scientific background knowledge required to understand implementations.

4 Work Plan

- Summer to late September
  - Meet supervisor
  - Perform background investigation of topic (i.e. read existing reports and investigate existing WebMGA from a user perspective)
- Up to late October
  - Wait for feedback from academic stakeholders
  - Identify bugs and gaps in existing features
  - Discuss important scientific information with supervisor
  - Begin familiarising with existing WebMGA codebase, particularly JavaScript, NodeJS, React, and threejs
- Late October to early February
  - Work on aims and objectives specified
  - Submit project plan, interim report as required
- Early to mid February
  - Lock features and analysis to be undertaken for rest of project
  - Outline structure of report
  - Begin preparing video presentation
- Mid February
  - Finalise and deliver video presentation
  - Aim to complete bulk of implementation
  - Perform analysis work

- Second half of Term 2
  - Focus on final report
  - Freeze development, aim only to fix bugs identified in analysis and report production
- End of Term 2
  - Discuss report with supervisor
  - Use feedback to work on further drafts
- Start of Term 3
  - Submit final version to supervisor
  - Perform any final report tweaks
  - Submit work completed by deadline

5 Ethical Concerns

I do not forsee this project requiring ethical approval. Any libraries or data used will be included compliant with relevant licenses, or not used if this is not possible. If any molecular configurations are provided from externally, they will be credited as requested.

A.2 Interim Report

Interim Report
WebMGA 3.0: Refinement of an Interactive Viewer for Coarse-Grained Liquid Crystal Models

Joe Down

January 2023

Abstract

Produce, document, and benchmark an enhanced version of an existing web-based tool for visualising coarse-grained liquid crystal models. Provide insights into related computer graphics performance and optimisation.

1 Supervisor Information

- Guido Germano, Professor of Computational Science
- g.germano@ucl.ac.uk

2 Progress

- Various bugfixes
  - Bounding box loading issues fixed
  - Fixed some incorrectly implemented keyboard controls
  - Fixed issues with model and renderer synchronisation
- Explored improved director calculations
  - Found to be impractical to implement due to poor performance and quality (i.e. limited functionality) of array mathematics libraries for JavaScript
- Re-implemented axes to be more useful
  - Transformed to a more appropriate screen position
  - Added director indicator
  - Colour based on director
  - Required changing render and model synchronisation to fix axes update "lag" bug
- Re-implemented colouring from director

  - Changed colouring definition from sampling a predefined palette, to a hue derived linearly from the vector dot product with the director

- Enhanced re-implementation of shape generation
  - Re-implement and simplify vertex generation for sphere, ellipsoid, spherocylinder, spheroplatelet, cut sphere
  - Cut sphere was previously erroneously cut from both ends
  - Implementation of cap shape
  - Implementation of lens shape
  - Changed model mesh complexity definition
    * Include a larger range of values
    * Move value increments to a logspace (powers of 2) due to diminishing returns of visual quality as complexity increases
    * Drastically simplified triangle generation from vertices
    * Removed mesh duplication previously present
    * Simplified normal generation
- Implementation of importing molecule configuration from standard .cnf files more commonly used in actual molecule simulation
- Prototype code for generating large configurations from tiling of smaller configurations (aiming to produce a more realistic benchmark)
- Viability investigation for distance based variable LOD
  - Found to conflict with existing InstancedMesh implementation for instanced rendering (conflicts with existing optimisation), will be revisited later

3 Work to Complete

3.1 Up to mid/late February

- Better axis labelling (i.e. arrowheads and letters)
- Fix significant model/UI value synchronisation issues
- Attempt to implement distance based variable level of detail either alongside instanced rendering if possible, or in place of it, and compare performance. Remove if found to be inefficient
- Improve benchmark to use a more realistic molecule configuration
- Further investigation of possible performance enhancements
- Fix separate director calculation issue for configurations with multiple molecule shapes (only a single director should be calculated)
- Implement improved image export functionality

  - E.g. specify output format based on common printing properties (size/dpi)

- Verify lens/cap/cut sphere coordinate definitions (i.e. where the centre is defined relative to mesh) with an academic working with these shapes
- Lock future project scope

3.2 Up to mid March

- Improve documentation of existing code
- Perform performance comparisons comparing combinations of optimisations implemented
- Verify experimentally whether CPU performance becomes the limiting for rendering speed in very large configurations
- Document all new implementations up to this point in. Address successes, failures, and compromises
- Continue (with reduced focus) any implementation which did not meet previous target deadline
- Semi-final code completion

3.3 By 22nd March

- Prepare video preview

3.4 By 12th April

- Document performance comparisons in report
- Document relevant scientific background (e.g. properties of molecule configuration such as director, use cases)
- Comment on scope for improvement, provide relevant research
- Semi-final report draft completion

3.5 By 26th April

- Address previous draft feedback
- Project completion