university of groningen

faculty of mathematics and natural sciences

VISUALISATION OF THE COSMIC WEB USING WEBGL

Bachelor's thesis

August 27, 2014

Student: Rick van Veen

Primary supervisor: prof. dr. J.B.T.M. Roerdink

Secondary supervisor: dr. M.H.F. Wilkinson

External supervisor: drs. J. Hidding

External supervisor: prof. dr. M.A.M. van de Weijgaert

CONTENTS

1 INTRODUCTION 3
1.1 The cosmic web . . . 3
1.2 Problem description . . . 3

2 REQUIREMENTS 4
2.1 The cosmic web visualisation requirements . . . 4
2.1.1 The data . . . 4
2.1.2 The visualisation . . . 4
2.1.3 Interactivity . . . 4
2.1.4 Performance . . . 5
2.1.5 Extensibility . . . 5
2.2 survey data visualisation requirements . . . 5
2.2.1 Visualising the survey data . . . 5
2.3 Connecting the two visualisations requirements . . . 6

3 DESIGN AND IMPLEMENTATION 7
3.1 Technologies . . . 7
3.1.1 Web Graphics Library - WebGL . . . 7
3.1.2 JavaScript 3D library - Three.js . . . 7
3.1.3 Polygon File Format - PLY . . . 8
3.2 The data . . . 8
3.2.1 The Cosmic Web Geometry . . . 8
3.2.2 The 2Mass Redshift Survey and Abell catalogue . . . 8
3.3 The cosmic web visualisation . . . 9
3.3.1 Converting the data . . . 9
3.3.2 Loading the data . . . 12
3.3.3 Colors . . . 13
3.3.4 Visualisation . . . 14
3.3.5 Results . . . 16

4 EVALUATION 17
4.1 The visualisation of the cosmic web . . . 17
4.1.1 The visualisation . . . 17
4.1.2 Interactivity . . . 17
4.1.3 Performance . . . 17
4.2 What is still missing . . . 17
4.3 Conclusion . . . 18

5 FUTURE WORK 19
5.1 The cosmic web visualisation . . . 19
5.2 The visualisation of, and connection with the survey data . . . 19

Appendices 19

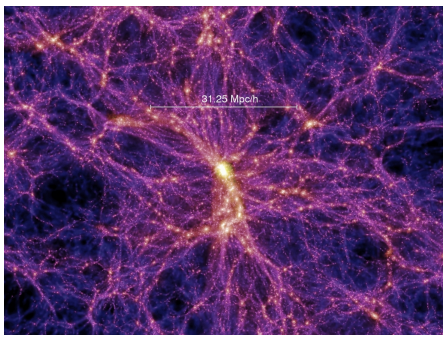
A PLY FILE STRUCTURE 20

B USER MANUAL 21

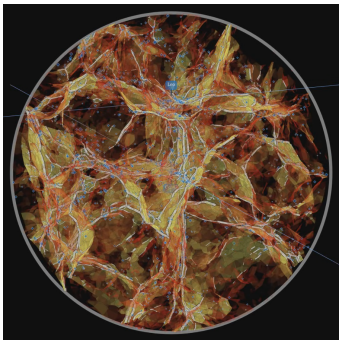
1 INTRODUCTION

1.1 THE COSMIC WEB

To understand the dynamics of the largest structures in the Universe, astronomers have built catalogues of millions of galaxies and tried to build a simulation that reproduces their distribution in space. When looking at this data from a large distance these galaxies trace a pattern the astronomers call the 'Cosmic Web'. In these patterns one can identify flattened sheets, elongated filaments and compact clusters surrounding large empty voids. These structures are very complex and hard to visualize, because of the size and complex multiscale structure of the data sets. E.g the image in figure 1a is generated by The Millenium Simulation Project [4] in what is called The Millennium Run. This simulation used more than 1 billion particles to trace the evolution of the matter distribution in a cubic region of the universe, over 2 billion light-years from the middle to each side. The simulation produced an output of 25 TB of data and kept the principal supercomputer at the Max Planck Society's Supercomputing Centre in Garching, Germany busy for more than a month. Figure 1b shows a screen shot of one of the movies made by SCAM, which is another example of a visualisation of the cosmic web made by J. Hidding [8]. The idea of building the prototype that is discussed in this thesis came from J. Hidding [8] and was inspired on his work done for the tool SCAM.



(a) The Millenium Run



(b) Visualisation generated by SCAM

Figure 1: Simulations of the cosmic web

1.2 PROBLEM DESCRIPTION

There does not exist a fast and interactive way to visualise the cosmic web. The goal of this project is to build a web-interface, that combines data from galactic databases, and the geometric data of the cosmic web into an interactive visualisation.

This thesis describes the work done for the prototype which tries to solve this problem using the WebGL technology, which will be discussed in section 3.1.1. The thesis is organised in the following way: We will start with a discussion of the requirements of the prototype in chapter 2. In chapter 3 we will discuss the design and implementation of the prototype: How does the prototype make sure the requirements are met. Then in chapter 4 we will discuss how well the prototype has satisfied the requirements. Finally in chapter 5 we will discuss some of the unsolved problems of, and possible extensions for, the prototype.

2

REQUIREMENTS

This chapter discusses the requirements of the prototype. Please recall the problem description in section 1.2. We can divide this problem into three components:

- An interactive visualisation of the cosmic web data.
- An interactive visualisation of the survey data[1].
- The connection between the two visualisations.

2.1 THE COSMIC WEB VISUALISATION REQUIREMENTS

The requirements of the visualisation component can be divided into four different subcomponents:

- What type of data does the visualisation accept as input.
- How should it visualise the data.
- What makes the simulation interactive, what kind of functionality is required.
- What are the performance demands.

All these subcomponents are discussed in the following subsections.

2.1.1 *The data*

The data that are needed for this module are data that describe the geometry of the cosmic web. These data are stored in the ply file format which will be discussed in section 3.1.3. The specifics for the ply files can be found in Appendix A. We will discuss the cosmic web geometry data in more detail in section 3.2.1.

2.1.2 *The visualisation*

The cosmic web geometric data must be visualised in such a way that the astronomy users can:

- Identify the filaments and walls clearly.
- Easily see the difference in density between the different walls and also between the different filaments.

2.1.3 *Interactivity*

To make the simulation interactive, users must be able to:

- Turn the camera in the visualisation so they can look at the data from different angles.
- Watch the walls and filaments data separately.

[1] Galaxy redshift survey data and/or data on the spacial distribution of galaxies

2.1.4 *Performance*

Performance is an important requirement for every interactive visualisation. The visualisation should satisfy the following requirements:

- The visualisation should achieve a frame rate of 60 frames per second
- The loading time of the data should be below 30 seconds.

Many different devices exist that can connect to the web and display the visualisation. The performance requirements should hold for the client with at least the following hardware and software specifications in order to satisfy the performance requirements:

- Hardware: Desktop computer, with a dedicated GPU.
- Software: Web browser, with WebGL support.

2.1.5 *Extensibility*

The user that manages the supply of data sets should be able to:

- Add/remove data sets to/from the application

The prototype should feature a way of doing this without having to change any code.

2.2 SURVEY DATA VISUALISATION REQUIREMENTS

This section describes the requirements for the visualisation of the survey data. A more detailed description of this survey data can be found in section 3.2.2.

2.2.1 *Visualising the survey data*

The prototype should feature a way of navigating through the cosmic web. The astronomy users are used to seeing the data of the cosmic web as an Aitoff-Hammer projection. This is why the prototype should feature an interactive Aitoff-Hammer projection of the survey data (section 3.2.2).

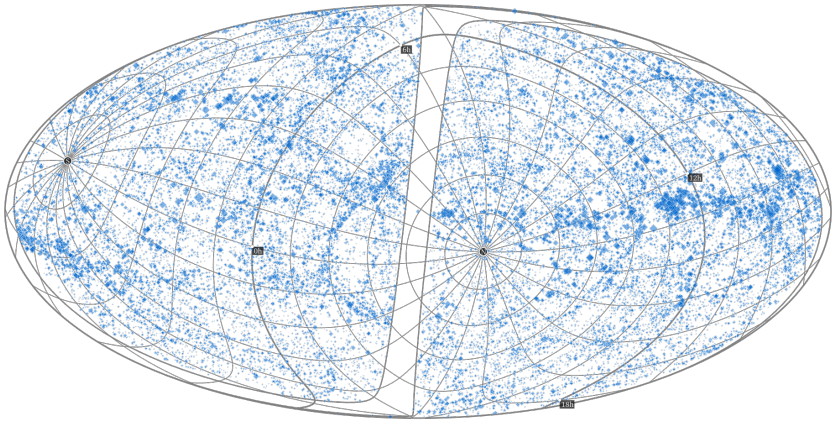


Figure 2: Aitoff-Hammer full-sky projection

Figure 2 shows an Aitoff-Hammer projection. This is a full-sky projection. This means that it shows a 360 degrees view of galaxies around us, like being in the middle of the sphere and being able to see everything around us at once. Figure 2 shows an empty vertical stripe in the middle and two empty

areas in the top right and bottom left of the image. The reason that these areas are empty is because our own galaxy, the Milky Way, lies here and we cannot see what is behind it.

2.3 CONNECTING THE TWO VISUALISATIONS REQUIREMENTS

The coordinates in the cosmic web geometry data are real coordinates. This means that the coordinates of the survey data are the same as in the cosmic web geometry data. Some of the galaxies can be identified by names or numbers. If possible some of the structures found in the cosmic web visualisation (chapter 2) should be labelled with these names and numbers found in the survey data. In this way the astronomy users have immediate feedback about which galaxies and structures are displayed.

The coordinates in the survey data and the coordinates of the cosmic web geometry data are not given in the same coordinate systems, so these coordinates need to be converted to one system. This will be discussed in more detail in section 3.2.2.

3 DESIGN AND IMPLEMENTATION

In this chapter we will discus the the design and implementation of the prototype of which the requirements are described in chapter 2. The development of the prototype has been divided into three phases. These phases have the same subjects as the components described in the requirements chapter.

- Phase 1: Interactive visualisation of the cosmic web.
- Phase 2: Interactive visualisation of the survey data.
- Phase 3: Connect the two visualisations together

As a side note: Because of time constraints the prototype only has an implementation of phase 1. Because of this the implementation of phase 2 and 3 will not be discussed.

This chapter will also discuss the technologies used by the prototype and the provided data for the visualisations.

3.1 TECHNOLOGIES

This section describes the technologies that are used.

3.1.1 *Web Graphics Library - WebGL*

The Khronos group is a not-for-profit industry consortium creating open standards for the authoring and acceleration of parallel computing, graphics, dynamic media, computer vision and sensors. WebGL is a 3D rendering API designed by the Khronos group for the world wide web. It is derived from OpenGL ES 2.0, and provides similar rendering functionality, but in an HTML context. WebGL is designed as a rendering context for the HTML Canvas element. The HTML Canvas provides a destination for programmatic rendering in web pages, and allows for performing that rendering using different rendering APIs [6]. WebGL is enabled in most of the popular browsers. A link is provided with a list of supported web browsers and how to get them: [7].

3.1.2 *JavaScript 3D library - Three.js*

The Three.js project group has built a JavaScript API on top of the WebGL API. This resulted in a framework which makes some simple things that would take a lot of code in plain WebGL as simple as a few lines of code. E.g. drawing a cube, which can be hundreds of lines of JavaScript and shader code in WebGL, is only a few lines when using the Three.js framework [2]. For this project we chose to use the Three.js framework because it gives a lower level of complexity and has a fairly large community, which could help with some of the problems that may be encountered. Because of the low abstraction level it will be easier to learn Three.js than just plain WebGL in the amount of development time. One could argue that the use of a framework could have a negative impact on the network performance, but with a file size of less than 430 KB this will be next to nothing compared with the geometric data files we need to download. The latest version of the Three.js framework can be found here: [2]

As a side note: Not everything about Three.js is perfect. The documentation of the framework is far from optimal, which could make some parts of the framework harder to understand and slow down the development.

3.1.3 *Polygon File Format - PLY*

The geometric data available for this visualisation is formatted using the polygon file format [1]. This format was chosen because of the freedom it gives in defining geometric shapes. Appendix A gives the structure of the ply files that the prototype can handle.

3.2 THE DATA

This section discussed the geometric and survey data that were available.

3.2.1 *The Cosmic Web Geometry*

The cosmic web is large scale structure obtained from a collection of observations done from the earth of galaxies around us. These observations cannot be used directly in a visualisation, so the data that actually represents the cosmic web geometry in polygon format is generated with what is called the adhesion model. This generation of polygon data is a complicated process that takes a starting condition of the universe in the shape of a density field. Then to simulate gravitation, an approximation method called the adhesion model is used. This geometric method results in the polygon data that is needed for the visualisation. To find the correct starting condition a process called reconstruction is applied. This process guesses a starting condition and lets this evolve over time, to the current time in the hope it will result in the observation the astronomers see today. If this process gets the wrong result the starting condition gets tweaked to get a better result.

As discussed in section 1.1 we can identify two kinds of structures: the flattened sheets (walls) and the elongated filaments (filaments). The cosmic web geometry data is also divided into these two types of geometric structures. The filaments are stored as a list of edges together with their density. This density represents the amount of matter. The walls are stored as a list of lists of points. The walls also store a density value. For a more detailed description of the ply files see appendix A.

3.2.2 *The 2Mass Redshift Survey and Abell catalogue*

The 2MASS redshift survey [3] is a ten-year project to map the full three-dimensional distribution of galaxies in the nearby Universe and contains around 45.000 galaxies (see figure 2). The Abell catalogue of rich clusters of galaxies is an all-sky catalog of approximately 4.000 rich galaxy clusters. The catalogues have a relation to the geometric data of section 3.2.1, so that the coordinates that are given in these catalogues correspond to the coordinates used in the geometric data. But the relation between these catalogues and the geometric data is not straightforward, the two catalogues use a different kind of coordinate system. The survey uses what is called the equatorial J2000 system and the galactic coordinate system. The Abell catalogue uses the equatorial 1950 system and the galactic coordinate system. The geometric data uses the super galactic system. If the catalogues and the geometric data are to be linked the easiest way to do this is to convert all the coordinates from the galactic system to the super galactic system.

The visualisation of the cosmic web is divided into four sections. Each of them discusses one of the following components:

- Converting the data (section 3.3.1)
- Loading the data (section 3.3.2)
- Colors (section 3.3.3)
- Visualising the data (section 3.3.4)

In section 3.3.5 the resulting visualisation is discussed.

3.3.1 *Converting the data*

As stated before, the available cosmic web geometry data is stored in the ply file format. As discussed in section 3.1.3 the ply file format is a very free standard of defining 3D objects and this makes it very easy to describe 3D objects in many cases. A drawback of this free format is that there are many different ways to define 3D objects. The structure of the cosmic web data did not follow the definition of a ply file that the default ply loader[1] of Three.js used. Because of this, the prototype has its own custom loader, see section 3.3.2 for a discussion of the implementation of this loader.

Because a new loader had to be built we were free to shift some of the responsibilities of the loader to another stage. A choice had to be made about parsing the ply file data beforehand in a pre-processing stage or to keep doing this at the client side in the new loader.

If parsing the ply file data would be done in the new loader, the ply files would have to be parsed every time a user requests to see a visualisation. The time it would take to parse the ply files on the client side, would depend on the computational powers of the client computer and this could have a negative effect on the performance of the visualisation. If the ply file would be parsed in a pre-processing stage and converted to an easier-to-load format, this would only have to be done once and would speed up the loading process. Because of the benefit of only having to do the computations on a data set once, the choice was made in favour of the pre-processing stage.

Implementation

This section discusses the implementation of the pre-processing stage in the prototype. Figure 3 shows the implementation in an image. The following numbers correspond to the numbers in this image.

1. The first step of the pre-processing stage is to convert the ply files describing the filaments and walls of the cosmic web. The conversion is implemented in the main conversion script called "ply_to_three.py".
2. The conversion script uses python script made available by J. Hidding [8] to parse the ply files.
3. The parser returns the data in an internal data format, to the main conversion script.
4. The main conversion script divides the converted data into two files. The first one is a json[2] file that stores the meta data of the actual data file storing the converted data.

[1] A loader is an object that handles the network request to the server for the data files. After receiving these files it also converts the information stored in these files to a format that can be visualised using Three.js

[2] JSON stands for JavaScript Object Notation and is special kind of JavaScript file in which Javascript objects can be defined.

5. The second file is a file in a binary format that stores the vertices, colors and faces.

The reason why the meta data (also called header) and the binary data are separated is because this way the conversion script and loader do not have to deal with a binary header of variable size, which makes the work that needs to be done on the client side easier. A discussion on how the header and data files look like for the filaments and walls data is discussed next.

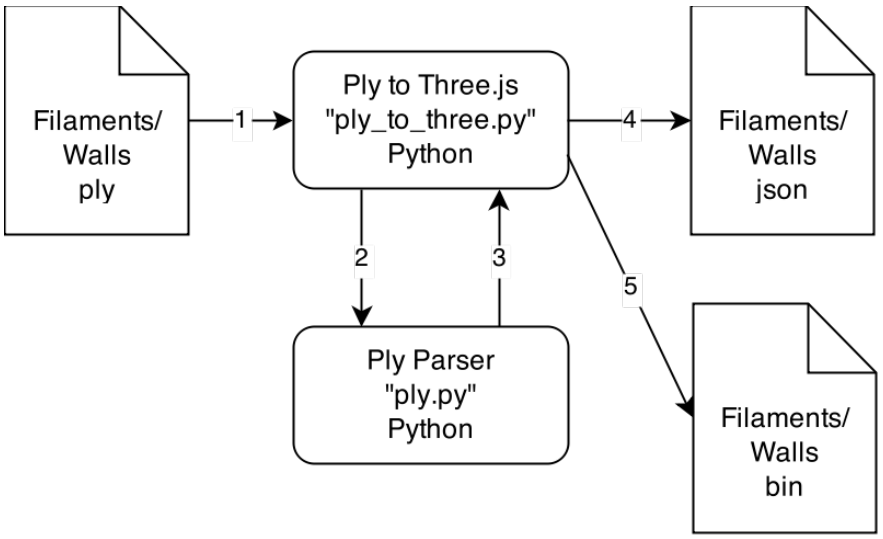


Figure 3: The pre-processing stage

The filaments

The structure of the filaments ply file can be found in Appendix A. The filaments are stored as a list of vertices and a list of edges. The edges are composed of two indices of vertices which form an edge. In the visualisation edges are represented by lines. In order to draw lines in Three.js we needed to convert the list of vertices and edges to a list of vertex pairs which form an edge together with a color per vertex. The main conversion script loops through the edge list and duplicates all the vertices in this list into one big vertex list. The density values of the edges are converted into colors (see section 3.3.3) and are stored per edge, the loader will convert this to store the colors per vertex, as discussed in section 3.3.2.

The walls

The structure of the walls ply file can also be found in Appendix A. The walls are stored in a list of vertices and a list of faces. These faces are lists of indices of vertices which form a wall together. Three.js cannot handle these polygons and thus these polygons need to be converted to triangles, which Three.js can handle. The algorithm that the conversion script uses can be found in figure 4.

```
algorithm PolygonToTriangles(V, C, F)
    input V is the list of vertices, C list of colors per face and F is the list of faces
    output The new lists: NV, NC and NF of vertices, colors and faces.
    for all faces f in F do
        I = list of indices stored in face f.
        c = color of the face f.
        for j ← 0 to j < length of I do
            Create three new vertices
            NV.append(V[I[0]])
            NV.append(V[I[j + 1]])
            NV.append(V[I[j + 2]])

            Add the color for this triangle
            NC.append(c)

            Create the new face
            NF.append([last three index numbers])
    return NV, NC, NF
```

Figure 4: Faces to triangles conversion

Figure 5 shows an illustration of what is happening in the inner for-loop of the triangle conversion algorithm shown in figure 4. The left most polygon has gone through the first iteration of the inner for-loop. The number of iterations increases to the right.

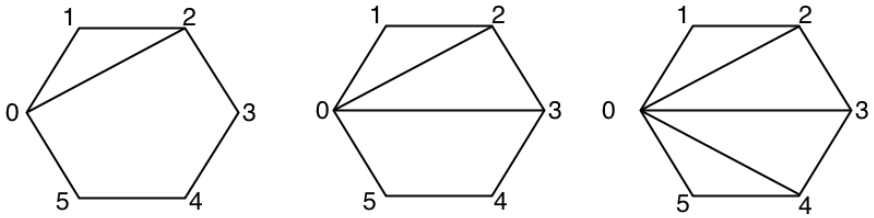


Figure 5: Polygon to triangles conversion. From left to right.

The conversion of both the filament and wall ply data expands the number of vertices, colors and faces. This has an effect on the file size of the files the prototype is going to send over the network. Table 1 shows how much the conversion increases the file size.

Filaments conversion				
File	Size(B)	vertices	colors	edges
Filaments.ply	8.796.741	382409	0	350.632
Filaments.bin	12.622.752	701.264	350.632	0
Walls conversion				
File	Size(B)	vertices	colors	faces
Walls.ply	51.721.725	2.029.612	0	1.089.873
Walls.bin	197.966.820	9.898.341	3.299.447	3.299.447

Table 1: File size increase (meta data file not included)

Table 1 shows that especially for the walls the file size increases to a size almost four times as big. With the average download speed in the Netherlands of 44, 8 Mbps (5, 6 MB/s) this would take about: 197 MB/5, 6 MB/s $\approx$ 35 seconds. We stated in the requirements (chapter 2) that the loading time should be between 0 and 30 seconds. So we need to find a way of lowering the time it takes to load the files, in order to fulfil the performance requirement. In the prototype this problem was solved by cutting the data set into smaller pieces. Then, to select one of these pieces a list is generated from a

json file which stores the available data sets. To see how to add more data sets, consult the user manual in Appendix B. The data sets are cut in a way that they represent galaxies increasingly further away.

3.3.2 *Loading the data*

After the data has gone through the pre-processing stage the data is almost ready to be loaded into the main visualisation script. The default way of describing a 3D object in Three.js is the Geometry class [2]. This class uses the Three.js vector, face and color objects to represent every point, color and face index. The earlier versions of the prototype also used this geometry class, but because of the size of the data set and the overhead that was created by storing all the data into JavaScript objects, it was not possible to get something onto the screen. Luckily the Three.js framework also provides a class called the BufferGeometry [2]. This Geometry class does not use the JavaScript object to store the data, but instead stores the data into large buffers. These buffers can then be sent directly to the GPU.

Implementation

In the loader these buffers are created by the JavaScript ArrayBuffer and Dataview objects [10]. The ArrayBuffer enables the loader to request one large array of raw binary data i.e. the binary filaments or walls files. This raw data cannot just be copied into the BufferGeometry buffers, but first need to be parsed. E.g. in the case of vertices to an array of 32-bit floats. The Dataview object enables us to look at an arbitrary chunk of the Arraybuffer by passing it the raw data with a start offset and the number of bytes we want to look at. With the methods of the Dataview object the raw data bytes can now be parsed to typed variables. These variables can then be processed and stored in the typed buffers of the BufferGeometry.

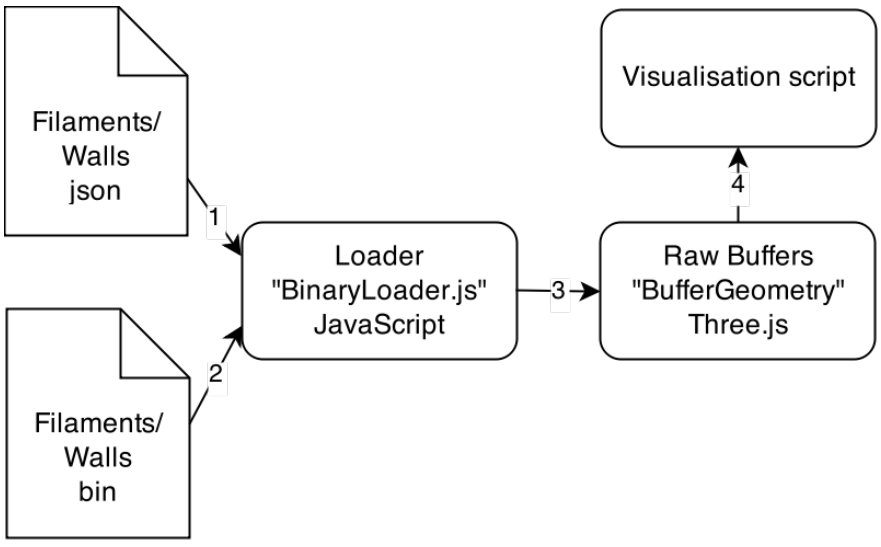


Figure 6: Loading process

In figure 6 the loading process has been illustrated. The loading stage is implemented in the BinaryLoader class. The following numbers represent the numbers in figure 6 and describe the loading process.

1. The loader first request the meta data (or header) file in which the number of vertices, colors and faces are stored, together with the name and location of the binary file containing the actual data.

2. The binary file is requested and loaded. This file stores the geometric data that represents a part of the cosmic web geometry.

3. The loader converts the data from the binary file to typed arrays to store in the BufferGeometry class. During this process the colors need to be multiplied. We multiply the colors because Three.js needs to have a color for every vertex in order to color an object correctly. For the edges of the filaments we need twice the number of colors and for the triangles of the walls we need three time the number of colors.

4. In the final stage the buffer geometry is returned to the main visualisation script that is described in section 3.3.4. This script will take care of the final stage of the visualisation.

Index buffer

One of the problems with WebGL is that the index buffers use 16 bit integer values. The BufferGeometry used to store the walls uses an index buffer and a number of indices that does not fit into a 16 bit integer. Luckily there is a solution to this problem: With the BufferGeometry a special offset object can be computed. The offset has a length of $2^{16} - 1 = 65.535$. We need to subtract one, because the buffer is used to store triangles and thus the final number needs to be divisible by three. With these offsets the buffer gets cut into multiple smaller buffers with the appropriate start offset and buffer size. This way the prototype can still have an object with more than $65.535/3 = 21.845$ triangles and use the faster BufferGeometry.

3.3.3 *Colors*

The colors are calculated during the pre-processing stage in the HSV color system. The HSV system is used because it is an intuitive way of defining colors. The HSV color system is a cylindrical coordinate representation. Figure 7 shows an image of the HSV cylinder. The H of the HSV color system stands for hue and is a value between 0 and 360 degrees. This angle represents a color on the cylinder. The S in HSV stands for saturation. The last letter V in HSV stands for value and represents the perceived luminance (Brightness).

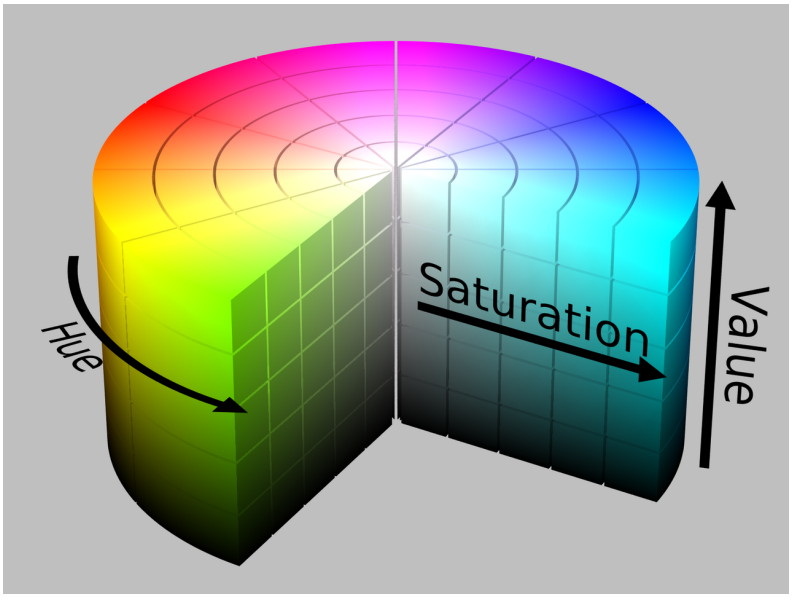


Figure 7: HSV cylinder

The density values of the walls and filaments are mapped to a range of hue values, so a range between 0 and 360. The lowest value of the range represents the lower densities and the higher value of the range the highest values. Because the Three.js framework cannot handle the HSV colors, the HSV colors are converted to RGB colors in the loader.

3.3.4 *Visualisation*

The visualisation stage is the last stage that the data goes through. In this stage the data has been converted and loaded into the BufferGeometry, see sections 3.3.1 and 3.3.2. Figure 8 represents the process in the main visualisation script. The following numbers represent the numbers in this figure.

1. and 2. In the visualisation script the BufferGeometry is combined with a Three.js Materials class. The Material class stores information for generating the shaders and this information will later be used by the Three.js framework to generate these shaders. The combination of the material and buffers as shown in figure 8 is represented by the Mesh class of the Three.js framework. In the case of the lines for the filaments the Three.js framework uses a special mesh class called "Line".

3. The mesh is added to the Scene class of Three.js. The scene object stores all relevant information about the rendering scene. E.g. Lighting, 3D objects and a camera object.

4. After everything has been added to the scene a special class called the WebGLRenderer is called to render the scene. The renderer convert all the materials and buffers to shaders and typed arrays and sends them to the GPU so they can be used to display the visualisation on the screen.

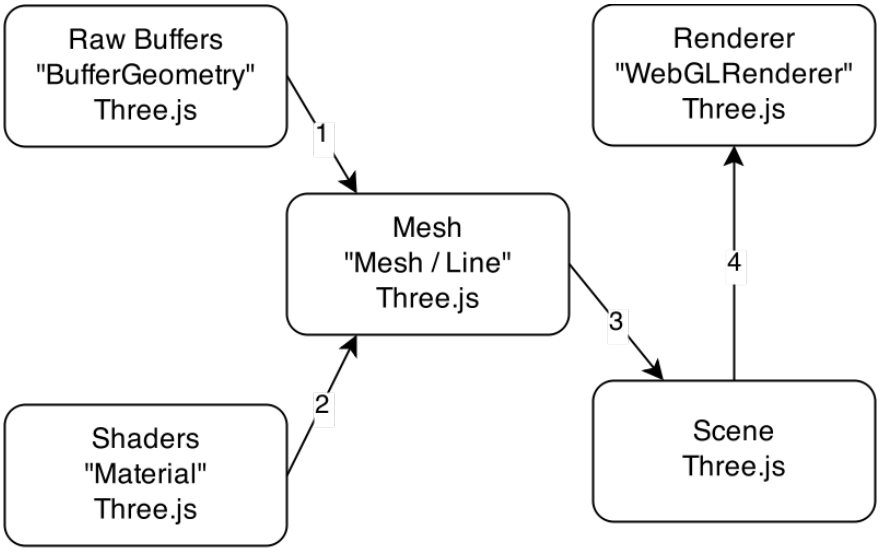


Figure 8: Process in the main visualisation script

Line thickness

In the requirements in chapter 2 we stated that an astronomy user should be able to easily identify the filaments and walls in the visualisation of the cosmic web. In order to achieve this we added an interface to the prototype with the option to change the line thickness of the filaments. Because the Windows operating system does not by default use OpenGl, on which WebGL is based (see section 3.1.1), the line thickness could not be changed. This made the filaments at some zoom levels hard to distinguish.
In a attempt to solve this problem we researched the possibility of using 3D polylines. Figure 9 shows a cube which is connected using these 3D lines. A 3D polyline is in contrast to a normal line a 3D object constructed of two spheres at the beginning and end coordinates of where the line should be, and a cylinder in between the spheres. The option to change the thickness of

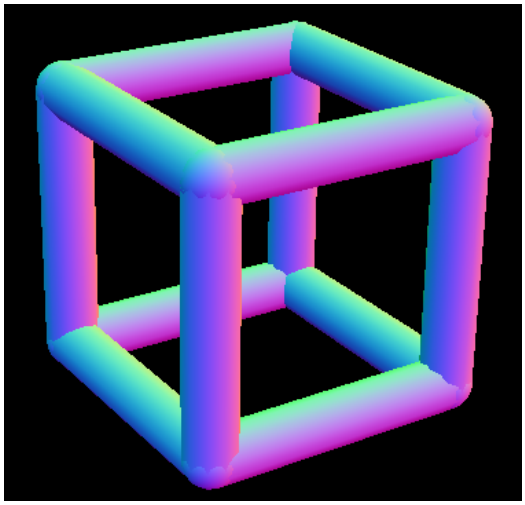


Figure 9: Cube using 3D polylines. The normal buffer is used for the colors.

the lines would change the diameter of the spheres and cylinders. Because every sphere and cylinder is constructed of triangles the number of points and faces explodes when using this method. The number of points and triangles even became so high that it has proven to be a too expansive solution for the line thickness problem. The spheres and cylinders were calculated on the client side using the Three.js framework, but it took to long to compute all the cylinders and spheres for the filament data to display anything onto the screen.

Interactivity

The requirements (see chapter 2) state that the visualisation needs to be interactive in the following way: The user needs to be able to look at the data from different angles. In order to achieve this the prototype uses a special class that controls the camera's position, called OrbitControls. This class listens to the mouse and even touch commands a user gives and will reposition the camera accordingly.

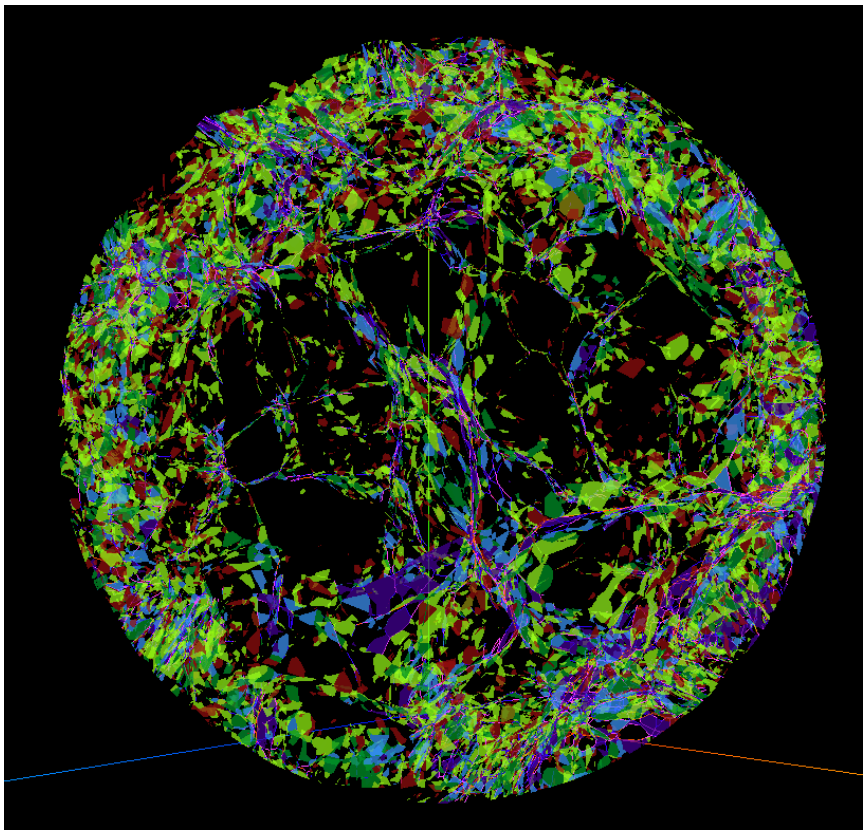
Another requirement is that the user has to be able to look at the filaments en walls separately. In order to achieve this a graphical interface has been added. The interface passes the variables to the material class and thus the shaders of the filaments and walls. The scene will then be redrawn by the main visualisation script. This interface features besides the option to hide and show the filament and wall data, also an option to adjust the opacity of the walls and to change the line thickness of the filaments. These options have been added to give the astronomy users the ability to adjust the visualisation to their preferences.

Extensibility

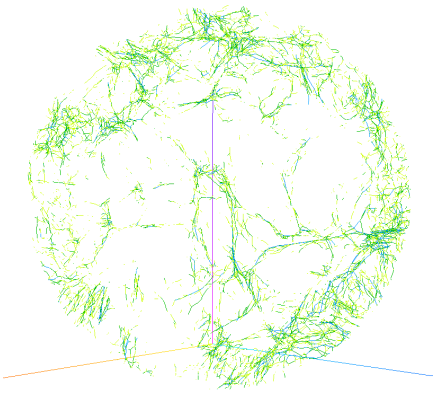
In order to fulfil the extensibility requirements the prototype uses a json file that stores the available cosmic web geometry data sets. This json file can be changed by the user that manages the available data sets to include more or less data sets. Appendix B describes how to do this. The main visualisation script request this json file and creates a web interface in which the user can select one of the available data sets to be visualised.

3.3.5 *Results*

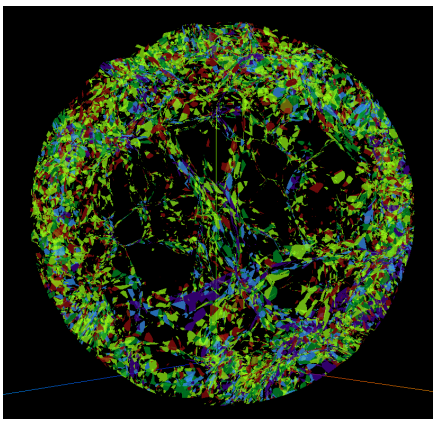
One of the reasons to build the prototype was to be able to show the structure of the cosmic web in a 3D and interactive way. The best way to look at the results would be to install and run the prototype (see Appendix B). But for completeness figure 10 shows the final output of the conversion, loading and visualisation (see section 3.3.1, 3.3.2 and 3.3.4) of the cosmic web geometric data. The 3d object shown in the images has the shape of a shell. In the middle of the image in figure 10a lays the earth, from which the data was collected (see section 3.2.1). In figure 10b the same data set is shown but without the walls. In figure 10c the walls are shown without the filaments.



(a) The result including the walls and filaments. The color range of the walls is: 0 to 180 degrees. The colors of the filaments lay between 180 - 360 degrees. See section 3.3.3 for an explanation of these values.



(b) The result without the walls. The color range of the filaments is inverted here.



(c) The result without the filaments

Figure 10: Results

4 EVALUATION

4.1 THE VISUALISATION OF THE COSMIC WEB

This section describes the evaluation of the prototype. In this section we try to answer the following question: Does the prototype do what it is supposed to do? In the next subsections the requirements that are discussed in chapter 2 are used to answer this question.

4.1.1 *The visualisation*

The requirements state that an user must be able to clearly identify the difference between the filaments, walls and their density values. The 3D polylines were an attempt to solve this problem, but this solution proved to be too expansive (see section 3.3.4). The prototype implements the distinction in densities currently with colors only. When the client runs on a Linux based operating system the thickness of the lines can be changed, but for Windows the prototype fails to do this.

4.1.2 *Interactivity*

The requirements for interactivity stated that users should be able to interact with the data set in the following ways. Firstly by being able to look at the data from different angles and secondly by being able to hide and shows parts (filaments and walls) of the data sets. The prototype has a graphical user interface (see section 3.3.4), that enables the users to hide and show the walls and filaments. The GUI also has the added options to change the opacity of the walls and the width of the filaments. By using a class that listens to the users input and that controls the camera, the prototype enables the users to look around in the data set (see section 3.3.4). The implementation meets the specified requirements for interactivity, but improvements can still be made. These improvements will be discussed in chapter 5.

4.1.3 *Performance*

One of the most time consuming problems was the performance. The prototype has been tested on a windows desktop machine[1] with a dedicated GPU. Because of the use of the much faster BufferGeometry (see section 3.3.2) the frame rate of the visualisations is easily above 60 fps. In order to ensure that the loading time did not exceed the required loading time of 0 to 30 seconds the data files were stored in a binary format and when this was not enough they were cut in smaller pieces (see section 3.3.1). All the implemented solutions improved the performance so much that it is even possible to run the visualisation on a tablet[2].

4.2 WHAT IS STILL MISSING

Phase 2 and 3 described in the beginning of chapter 3 are not implemented. These components will be discussed in the future work chapter. The reason that these two components are not implemented is mainly because the amount of time there was left after designing and implementing phase 1 was

[1] Processor: i5-2500k @ 3.30 Hz, 8 GB DDR3 memory, Nvidia gtx 560 GPU, Windows 8.1 pro operating system

[2] Asus, Google Nexus, 2013 model

insufficient for the amount of work it would take to design and implement these two components.

4.3 CONCLUSION

Many requirements are not exactly measurable. How can it be said that something is easily identified or how can it be measured that people easily can see the difference in densities. We hope that, maybe with some further development this tool can become a new way of looking at the cosmic web structure. To answer the question in section 4.1 and the problem description in section 1.2: No, the prototype at this point is not yet an interactive visualisation that combines the data from survey databases and the geometric data of the cosmic web. At this point the prototype is only an interactive visualisation of the cosmic web geometry. If we consider the requirements of the part that has been designed the prototype is successful.

5 FUTURE WORK

This chapter describes the problems that still needs to be solved and the ideas for the future of the prototype.

5.1 THE COSMIC WEB VISUALISATION

Most of the work has been done on the visualisation of the cosmic web. Still the visualisation could be improved. A few possible improvements:

- Add standard camera positions to the visualisation. These camera positions could be meaningful places for the astronomer that uses the visualisation. E.g. a camera position in the middle of the data set (the location of earth) pointing in the direction the astronomer usually sees the sky.
- The ability to map the colors of the densities on the client side. This could probably be done with a set of custom shaders. Instead of calculating the colors beforehand the custom shaders could be given the densities as attributes and a way of using these densities in a colour calculation.
- Find a better way of visualising the filament data, so the structures become more clear and the density values more distinguishable.

5.2 THE VISUALISATION OF, AND CONNECTION WITH THE SURVEY DATA

Because this part of the project was not implemented, all the requirements are part of this future work section. An overview is given here:

- An interactive Aitoff-Hammer projection needs to be developed. A user could look at this projection and decide what piece of the data he or she would like to be visualised. This selection could than be sent back to the server. The server then dynamically creates the data set with the given variables and sends back this data set.
- In order to give the user feedback during the visualisation the survey data and the visualisation of the cosmic web should be connected. This means the prototype needs a way of labelling parts of the visualisation of the cosmic web with the names and numbers of the galaxies in the survey data.

A PLY FILE STRUCTURE

This section describes the structure of the header of the ply files used to describe the cosmic web. The converter that converts the ply files can only handle files with these elements and properties. For a manual on how to convert the data see appendix B. For a more detailed description on how to read the ply files in A.1 and A.2 see [1].

A.1 WALLS

```
ply
format binary_little_endian 1.0
comment walls
element vertex 2670
property float x
property float y
property float z
element face 1408
property list uchar uint vertex_indices
property float density
end_header
```

A.2 FILAMENTS

```
ply
format binary_little_endian 1.0
comment filaments
element vertex 656
property float x
property float y
property float z
element edge 635
property uint vertex1
property uint vertex2
property float density
end_header
```

B USER MANUAL

This manual describes the prerequisites, how to install the prototype, how to convert and upload the data, and how to select a data set.

B.1 PREREQUISITES

Server requirements: The prototype should run on any simple web server. The client: The best results are achieved with the chrome web browser running on a computer with a dedicated GPU. A benefit of performing the visualisation on a computer running a Linux based operating system is that the user will be able to change the line thickness of the visualisation.

B.2 INSTALLING

Copy the source files to a folder on the web server that is accessible on the internet.

B.3 CONVERTING THE DATA

In order to add a new visualisation the ply data files first need to be converted. The conversion is done with the ply_to_three.py script. For the conversion to work the ply files must follow the structure and names as described in the ply files of appendix A. If the ply files satisfy this prerequisite then the files can be converted using the following commands:

```
python ply_to_three.py -i filaments.ply -o filaments[.json] -t filaments
```

For the walls use the same command only with the "walls" option:

```
python ply_to_three.py -i walls.ply -o walls[.json] -t walls
```

The script also has a help function that can be consulted with the following command:

```
python ply_to_three.py -h
```

B.4 UPLOADING THE DATA

The prototype does not provide a way of uploading any of the files. But it is required that the two files (json and bin files) are uploaded to the same folder. If these files are not in the same folder the prototype will not be able to visualise the data set.

B.5 ADD THE SIMULATION TO THE LIST

The website request the available data sets from a json file stored at the data folder of the prototype. An entry for a data set in this json file looks like this:

```
"name":{
   "filaments":"./folder/filaments.json",
   "walls":"./folder/walls.json",
   "description": "Some description"
},
```

To add a data set add an entry, such as above, to the file and fill in the name, the location of the filament and wall meta data files and a description or comment about the data set.

At the top left of the screen, after entering the web address of the server running the prototype a drop down list appears. In this list all available data sets can be found. In order to display a visualisation of a data set select one from the list and click on the submit button.

[1] Paul Bourke. PLY - Polygon File Format. http://paulbourke.net/dataformats/ply/.

[2] Mr. Doob. Three.js specifications r67. https://github.com/mrdoob/three.js/, April 2014.

[3] Hucha et al. The 2MASS Redshift Survey - Description And Data Release. 2011.

[4] Springel et al. The Millennium Simulation. http://www.mpa-garching.mpg.de/galform/virgo/millennium/, 2005.

[5] Khronos Group. The OpenGL ES Specification. http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf, November 2010.

[6] Khronos Group. WebGL Specification. https://www.khronos.org/registry/webgl/specs/1.0/, March 2013.

[7] Khronos Group. Getting a WebGL Implementation. http://www.khronos.org/webgl/wiki/Getting_a_WebGL_Implementation, February 2014.

[8] J. Hidding. The Kapteyn Astronomical Institute, University of Groningen. Personal communication.

[9] J. Hidding. Phd thesis. 2014.

[10] ECMA International. ECMAScript Language Specification, standard ECMA-262 5.1 Edition. http://www.ecma-international.org/ecma-262/5.1/, June 2011.

[11] W. Schaap R. van de Weygaert. The Cosmic Web: Geometric Analysis. 2009.

[12] World Wide Web Consortium. HTML5. http://www.w3.org/TR/html5/, April 2014.