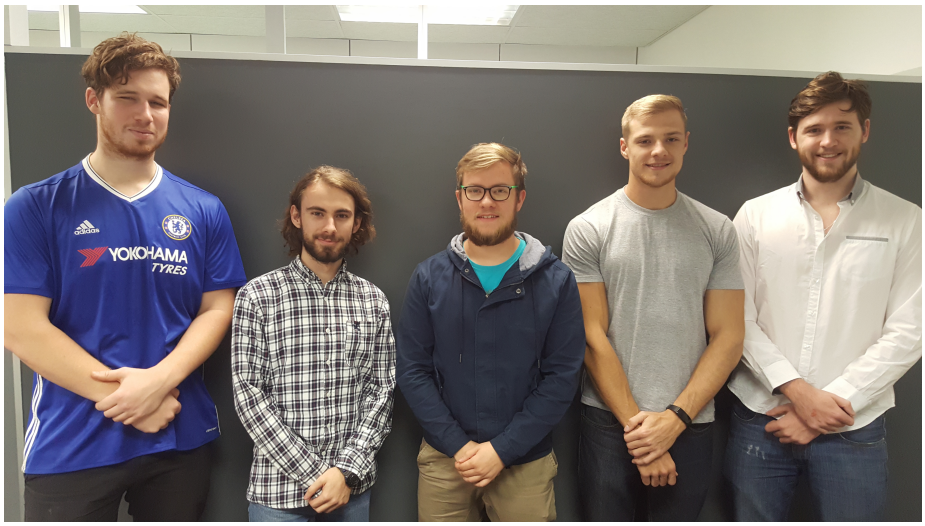

Absolute|0|

Voxc.js
Requirements Specification

Name	Student Number
Chris Dreyer	15072623
HD Haasbroek	15046657
Cameron Trivella	14070970
Pearce Jackson	14044342
Idrian van der Westhuizen	15078729

Contents

1 Introduction

1.1 Purpose

This SRS document aims to stipulate the requirements of the voxc.js library to aid in the development process and to ensure that a functional and usable product is delivered.

1.2 Product Scope

Voxc.js is meant to be an easy to use and easy to maintain JavaScript library, similar to how three.js is a library for WebGL. The purpose of the project is to provide users a way to import Voxel models into a webpage that would be using the Voxc.js library and use these Voxel models as a coordinate system to create newly genereated mesh object according to a rules file with a specified structure. The user will then be able to export the textured and rendered object.

1.3 References

https://pages.github.com/

http://threejs.org/

http://coffeescript.org/

http://es6-features.org/#Constants

http://www.typescriptlang.org/

http://www.codebelt.com/typescript/typescript-es6-modules-boilerplate/

http://giacomotag.io/typescript-webpack/

2 Architectural design

The voxc.js library revolves around different converters achieving different results by manipulating the ouput of other converters in different ways and then again providing their output to other converters. This whole process imitates a physical pipeline starting from one converter and ending at the last converter. Therefore we used the Pipeline Software Architecture to guide our structure and our architecture. The Pipeline Architecture resembles a physical pipeline and consists of a chain of processing elements. Each of our converters will act as a element in the pipeline, also called a filter in the pipeline. Each of these filters will use the output of the previous filter and then again provide its output as input to the next filter.

One problem with the structure of this architecture is the fact that Converter

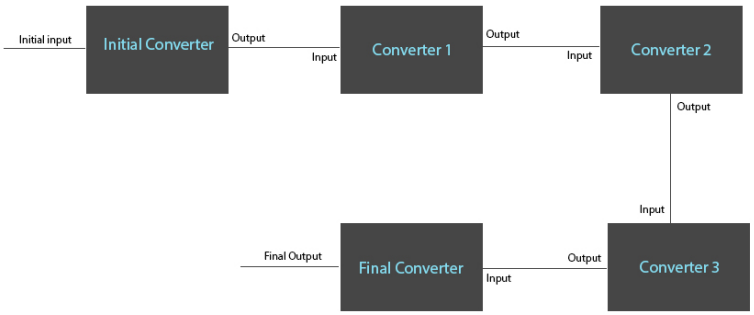

Figure 1: The voxc.js pipeline structure

3 might want to receive input directly from Converter 1 rather than Converter 2, and therefore an additional architecture is needed to enable the converters or filters to be loosely coupled. A Microservices Architecture is a perfect solution to the mentioned problem. By making our services lightweight and fine-grained we can allow the services to be loosely coupled and to be used interchangeably in the pipeline structure. Another major benefit of the use of this architecture is the fact that it makes the system easier to understand and inheritantly makes it easier to develop and test new features of the library. This is a major benefit

because essentially the voxc.js library is made to be tinkered with and changed to suit ones own needs.

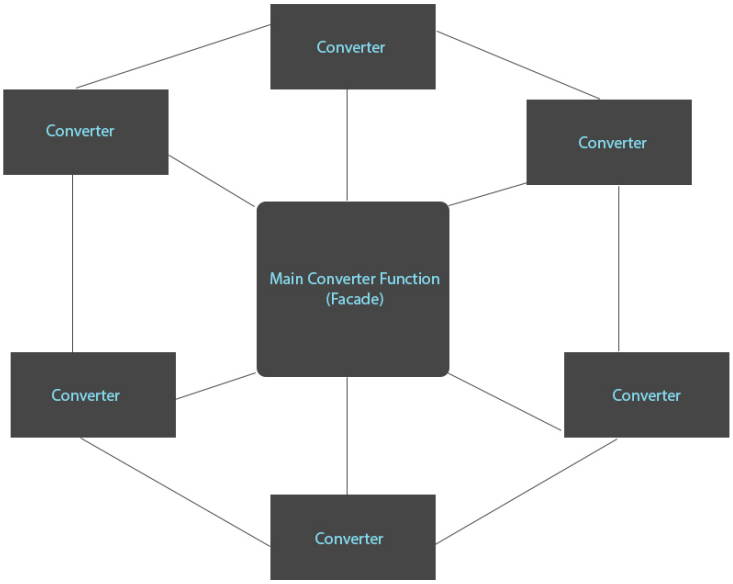

Figure 2: Pipeline architecture and Microservices architecture combined

3 Testing technologies and framework

We have incorporated two main technologies to aid us in testing each segment of our Voxel project. These technologies are Jasmine and Karma. Jasmine is an open source testing framework that is installed through the npm using Karma. It is used for unit testing of Javascript functions to ensure that the functions provide the expected output and to ensure that the functions behave as expected. With Jasmine distinct functions will tests various aspects and with the aid of the "expect" statement expected result is compared to the actual results and any mismatch will return a fail. Using Jasmine does require the tester to learn a new form of syntax that is unique to Jasmine.

With regards to Karma, it is a testing environment where testers need not worry too much about the smaller details but rather are able to tests and receive rapid results. Karma also keeps a vigilant eye on the different files and when there is a change it sends out a signal to rerun the test code. This en-

sures that all copies are up to date. Karma also enables the tester to customize the testing experience by specifying what aspects to include and to use, these specifications are saved in a "my.conf.js" file. To run and build the tests we used Travis-CI, Travis CI is a hosted, distributed continuous integration service used to build and test projects hosted at GitHub. Travis CI automatically detects when a commit has been made and pushed to a GitHub repository that is using Travis CI, and each time this happens, it will try to build the project and run tests.

4 External Interface Requirements

4.1 User Interfaces

The users should be able to interface with voxc.js through our web interface [github pages] that will be hosted on Github Pages. They will mainly be using a laptop or a desktop to interface with the library. However the library itself should be able to interface with any website on any device with a web browser that has support for WebGL.

The end users should thus be able to use the library for any webinterface they program that allows for the use of WebGL.

4.2 Software Interfaces

The system should incorporate several different languages in order to function. For the rules file, JSON objects should be used so that the users can manipulate the file in any text editor based on set conventions and structure.

MagicVoxel should be used for the creation and manipulation of voxel objects. The library should be able to handle the .obj filetype for the voxel objects.

The library should use TypeScript [Typescriptlang.org] as the JavaScript variant to enable future developers and current developers to debug easier. The library should run on all major internet browsers that support WebGL [ES6-features].

The library should be compiled together using webpack [typescript-webpack], as to allow end users to simply download a single Javascript file for use in their own projects.

4.3 Communications Interfaces

HTTP will be used to handle GET and POST requests and FTP will be used for file uploads and downloads.

Due to the way Chrome handles the loading of local files it should be made

clear to the user that in order to run anything locally that they would need to make use of some sort of server such as a node server or even xampp [threejs.org].

5 System Requirements

Requirement ID	Requirement description
R1	User should be able to upload their files from the webinterface
R1.1	User uploads valid .obj, png and rule file
R1.2	User uploads valid .obj and rule file. Should default png
R1.3	User uploads valid .obj and png. Should default rule file
R1.4	User uploads valid .obj. Should default png and rule file
R1.5	User uploads valid js arrays
R1.6	User uploads valid 3D matrix
R1.7	User uploads valid array of images
R2	Should be able to convert .obj to JSONMAtrix
R3	Should be able to convert array of images to JSONMAtrix
R4	Should be able to convert JSONMAtrix to Three-js mesh
R4.1	Should be able to create visual 3D-matrix
R4.2	Should be able to apply rule file
R4.2.1	Should apply textures and change matrix according to rule file
R4.2.2	Should be able to apply rule file with use of cellular-automata
R5	Should be able to convert Three-js mesh into .obj
R5.1	Converted .obj should be exportable/downloadable
R5.2	Should contain all textures for ease of use

Requirements ID	File upload	OBJ file upload with rules file	File upload without rules feature	Obj To Array	Array To Mesh	Mesh To Obj	IMGs To Array
R1	X		X				
R1.1	X	X					
R1.2	X	X					
R1.3	X		X				
R1.4	X		X				
R1.5	X		X				
R1.6	X		X				
R1.7	X		X				
R2				X			
R3							X
R4					X		
R4.1					X		
R4.2							
R4.2.1					X		
R4.2.2					X		
R5						X	
R5.1						X	
R5.2						X	

6 System Features

6.1 Download of library

6.1.1 Description and Priority

The end user need to be able to download a single javascript library which they should be able to apply and use in their own webpage. This feature has the highest priority as without it no other set of features will come together and be easliy usable by end users.

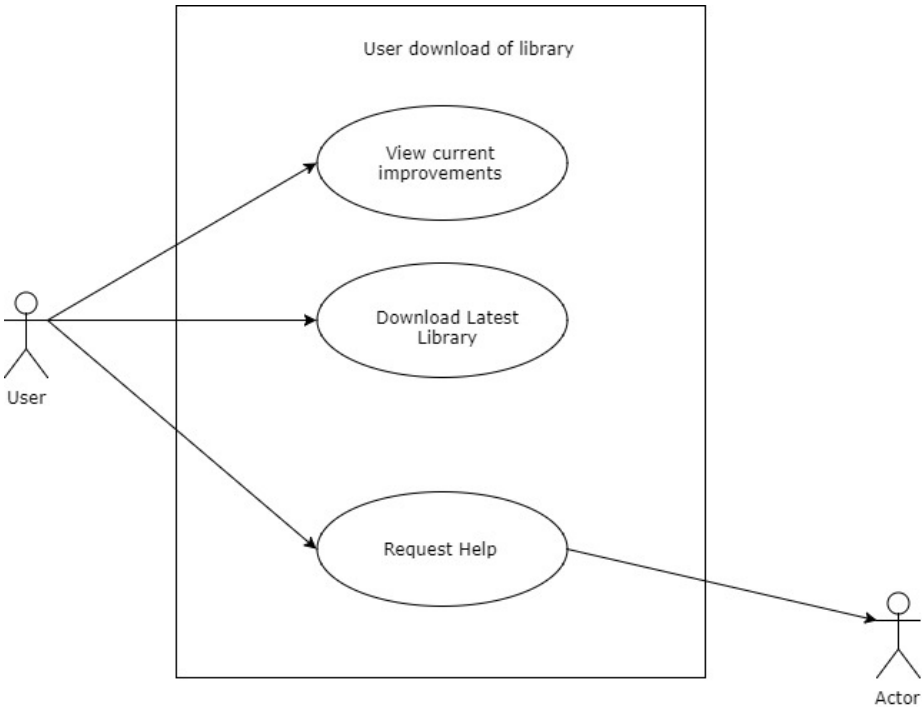

Figure 3: Use Case Diagram for download of library

6.2 File upload

6.2.1 Description and Priority

This feature will be needed for the webinterface, but is not required, thus it has a low priority. The feature will get user input in the form of a file upload, the feature should be able to determine what file is being uploaded and then through use of a strategy method determine the most valid course of action, i.e. selection of the 1st converter.

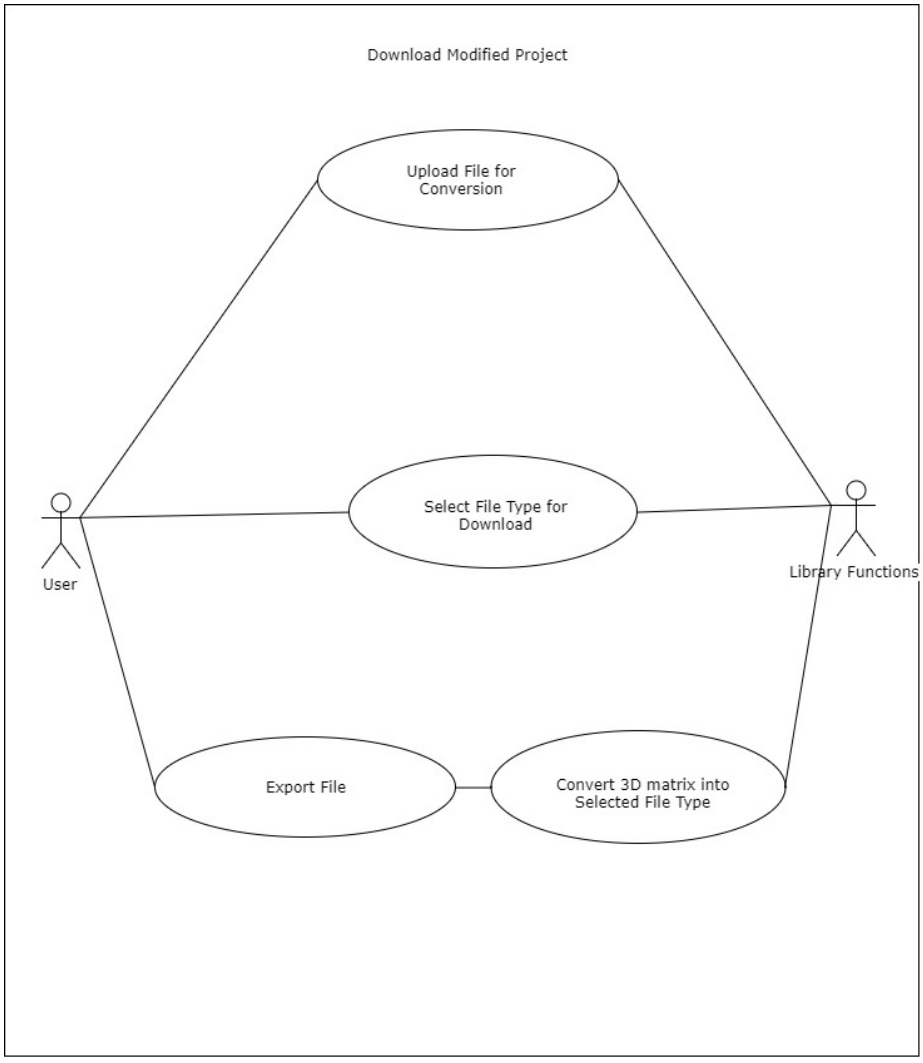

Figure 4: Use Case Diagram for User upload and download of files

6.3 OBJ file upload with rules file

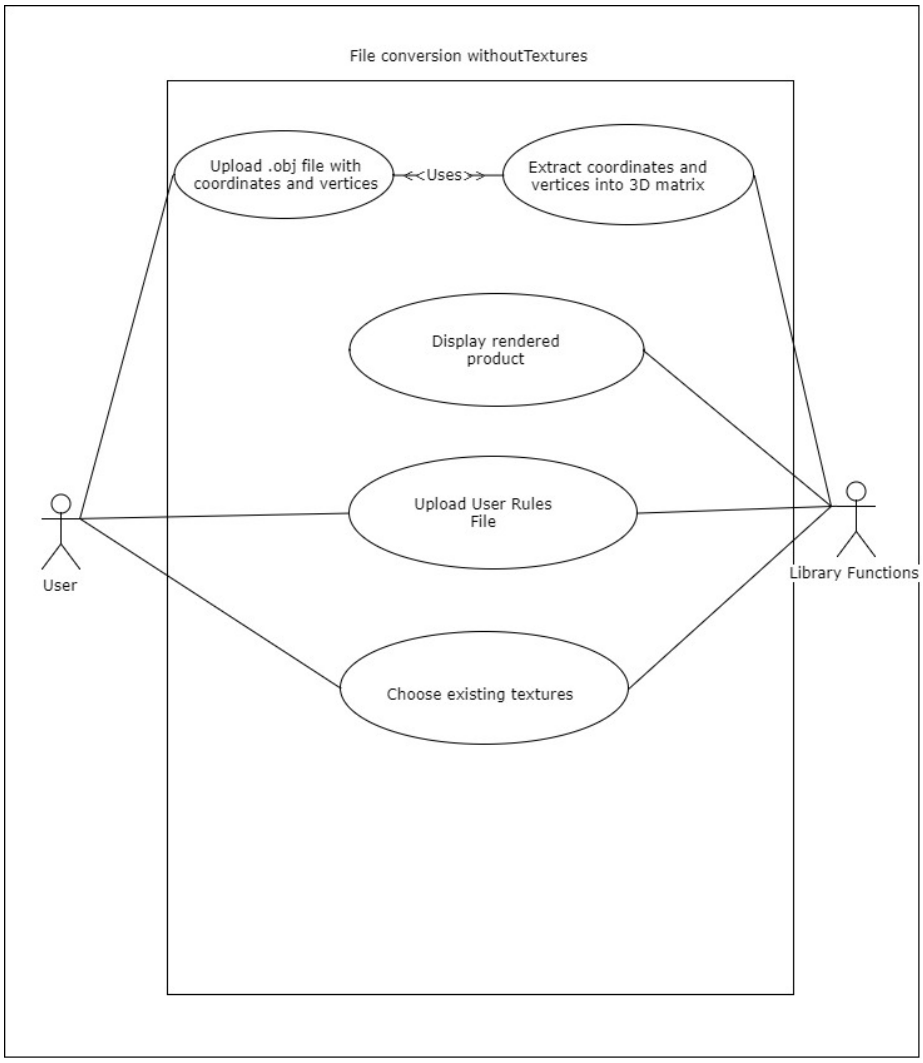

Figure 5: Use Case Diagram for User upload with rules

6.3.1 Stimulus/Response Sequences

Stimulus: The user uploads his file with the rules file included.
Response sequences: The system responds by using the files the user uploaded to extract them.

The system then determines the appropriate 1st converter and also provides the later converters with the uploaded rule file.

6.4 File upload without rules feature

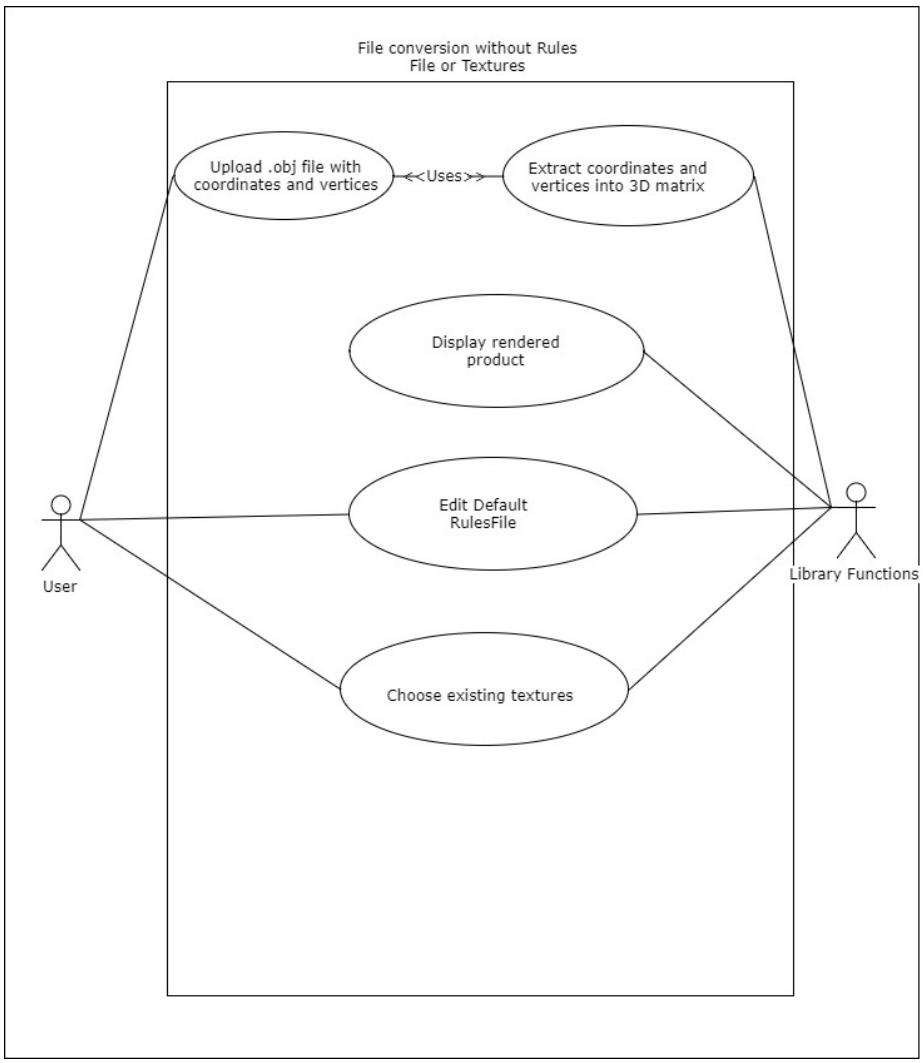

Figure 6: Use Case Diagram for User upload with rules

6.4.1 Stimulus/Response Sequences

Stimulus: The user uploads his file with the rules file excluded.
Response sequences: The system responds by using the files the user uploaded

to extract them.

The system then determines the appropriate 1st converter and also provides the later converters with the default rule file provided by the system.

6.5 ObjToArray

6.5.1 Description and Priority

This feature is the first step in the .obj file upload process and is required in order to allow the use of subsequent converters. It is thus of medium-high priority as the user need not upload or use an obj file, but can instead use one of the other initial converters for different file types.

The ObjToArray class should take in and obj file, preferably one created with the use Magicavoxel, that would then convert it into a JSONMatrix array containing the color values of each voxel from the obj.

6.6 ArrayToMesh

6.6.1 Description and Priority

This feature is an intermediate/bridging converter where the outputs of each initial converter would need to go through in order to apply the rule file and create the new 3D object a.k.a the new mesh. It is beacuse all converters need to go through this converter eventually that it has a very high priotity, because if this converter fails then all other before and after converters would fail as a result.

The converter will start using the outputted JSONMatrix of previous converters as a coordinate system in order to determine where textures and objects should go according to the supplied rule file.

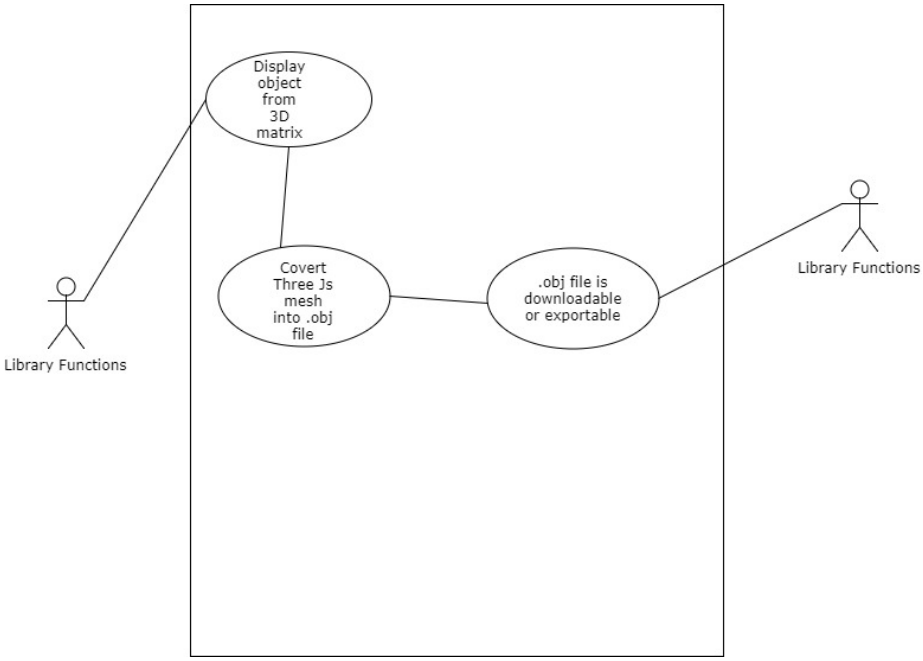

Figure 7: Use Case Diagram for display of mesh

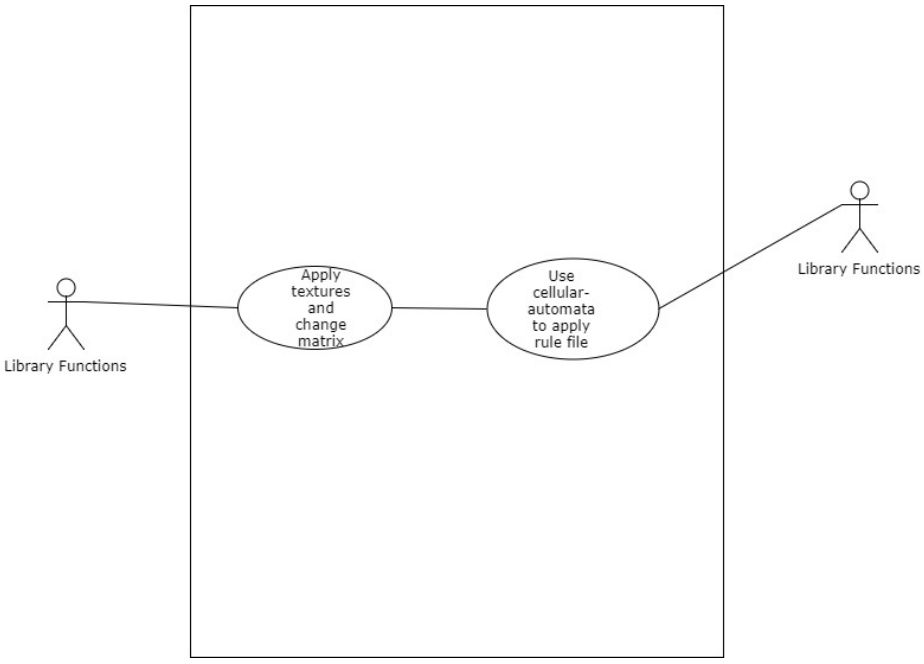

Figure 8: Use Case Diagram for applying of the rules file

6.7 MeshToObj

6.7.1 Description and Priority

This feature provides one of the end products of the converter pipeline and is optional to the end user, but is asked for by the client. It thus has a medium priority as most end users would probably ignore this feature, but because the client wants to use it, its priority is elevated.

6.8 IMGsToArray

6.8.1 Description and Priority

This feature is the first step in the array of images file upload process and is required in order to allow the use of subsequent converters. It is thus of medium-high priority as the user need not upload or use an image array file, but can instead use one of the other initial converters for different file types.

The IMGsToArray class should take in and array of images that would then be converted into a JSONMatrix array containing the color values of each pixel from the array of images.

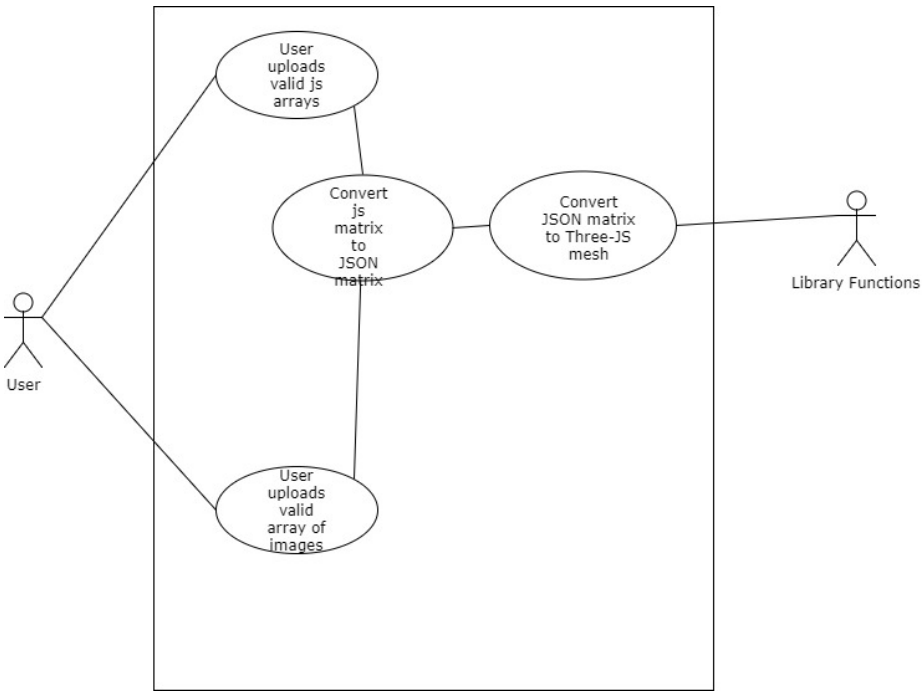

Figure 9: Use Case Diagram for images and array conversion

7 Other Nonfunctional Requirements

7.1 Performance Requirements

7.1.1 Time to respond to an uploaded file

A user is required to upload a voxel object to the library and then should be allowed to mainpulate a rules file if they choose to do so or they should be able to use a predefined rules file such that the rules are applied to the object. The library must not delay once a file is uploaded, this means the time it takes to respond to a uploaded file should be proportional to the files size.

7.1.2 Maintainability

The library's code must be well documented, both by means of in-code comments and external documentation, to aid in maintaining the system. A user manual should also be provided to make it easier to understand and ultimately maintain the library.

7.1.3 Portability

This library should be accessible across all major internet browsers that support WebGL. These include Chrome, Firefox etc. The library should also be able to be imported to any website.

7.1.4 Scalibilty

The system should be able to handle the majority of web browsers and file sizes, by file size we mean that the system should not necessarily struggle or outright reject a file because it was too large. Realistically we cannot handle all file sizes, but the system should aim to handle a large as possible file size.

Additionally the library should be able to work with any other webpage as a simple imported library, therefore users should be able to alter positions and perhaps even sizes of the dom elements used to upload, download and display the objects.

7.1.5 Usability

The Voxel library should be easy to operate and understand. The core functions of the system shouldn't take the user more than a minute to access and understand. A user manual should be provided to aid in the use of the library and to explain some advanced functions or settings.

This library should be an open source project and so the code will be freely available to anyone visiting the Github repository. The actual web interface hosted on Github will act as a demo of the library and should be protected, the

only editing that a user should be able to do is uploading of objects and editing their own rules file.