NTNU
Kunnskap for en bedre verden

Open source JavaScript voxelization engine, with complementary software

André Storhuag

May 2020

PROJECT / BACHELOR'S THESIS

Department of ICT and Natural Sciences

Norwegian University of Science and Technology

Supervisor 1: Ricardo Da Silva Torres

Supervisor 2: Saleh Abdel-Afou Alaliyat

Preface

This bachelor thesis is written a student from Computer Engineering at NTNU Ålesund ....

This type of technology....

What intrigued us ...

Acknowledgement

We would like to thank......

- Our mentors....
- Family and friends...
- ....

Abstract

This report concerns the development of ....

The purpose of this project ....

Glossary

Content Delivery Network Deliver static, cached content from a network of servers across the globe.

Open-Source Software Software that is free to use....

Polyfill Code for providing modern functionality on older browsers that do not support it natively.

voxelization Process for transforming a polygon mesh into voxels.

SemVer Versioning scheme.

Notation

K_p Proportional term of a PID controller

K_i Integral term of a PID controller

K_d Derivative term of a PID controller

Kg System International unit for Kilogram

ACK acknowledge message

Abbreviations

API Application Programming Interface

GUI Graphical User Interface

OSS Open-Source software

CDN Content Delivery Network.

JSX JavaScript XML

JSON JavaScript Object Notation

NPM Node Package Manager

VCS Version Control System

RTF Rich Text Format

JS JavaScript

ES ECMAScript

ES5 ECMAScript5

ES6 ECMAScript6

CSS Cascading Style Sheets

AMD Asynchronous Module Definition

WebGl Web Graphics Library

JIT Just In Time

UUID Universally unique identifier

IO Input Output

IPC Inter-process Communication

List of Figures

List of Tables

List of Program Codes

Contents

Chapter 1

Introduction

1.1 Background

Since the introduction of the relatively new technology WebGL, users has been able to significantly expand the graphical user-experience for the large user mass of the web. It has allowed for almost desktop like graphics performance. WebGl has especially expanded the abilities regarding the creation of web browser games and simulations. Creating high performance graphics applications in the browser has never been easier. It has opened up for a sea of possibilities where only your imagination is the limit.

In a lot of simulation software, and even some games, volumetric information plays a crucial part. With everything from fluid dynamics to voxel games like Minecraft, volumetric information is a key component. One way to acquire such volumetric data is through voxelizing a 3D model (polygon mesh). However, to the best of my knowledge, it does not exist any easy-to-use open-source voxelization software written in JavaScript. In order to obtain such volumetric data, developers are therefore forced to go through a tedious preprocessing steps, often involving old and complex, har to use, platform specific tools.

This was a problem I encountered myself a year ago in 2019. In connection with an assignment in a simulation course at the Norwegian University of Science and Technology (NTNU), I needed to be able to easily generate some volumetric data based on 3D models. I was using web-technologies, so I was looking for a simple solution in plain JavaScript. However, I was not able to find such a solution. I therefore decided to make one myself. The result was an open-source voxelization project, written entirely in JavaScript. It was named Voxelizer.

However, the Voxelizer project carries strong signs of the limited amount of time allocated for creating the software. Due to it's vast range of applicability, improving and expanding the capabilities of the project could therefore serve to be a valuable open-source asset to the web based game- and simulation-development ecosystem, providing easy access to voxelization.

1.2 Problem Formulation

Problems to be addressed

There exists an open-source JavaScript voxelization project for voxelizing 3D models. The software faces several issues and is lacking important features. It does not produce accurate and representative results. The output sometimes contains holes and a lot of artifacts. Importing and exporting support is extremely limited. Documentation is lacking, and the coding is of poor quality. The project needs to be professionalized, and made easy to both use and maintain. A more complete description of the problems Voxelizer v0.1.3 faces are described in Section 2.14.

Packaging and publication of new releases, as well as documentation, are tedious and manual procedures. This workflow is prone to human errors, potentially introducing critical bugs. These processes could be automated with modern continuous integration and continuous deployment tools, effectively eliminating these vulnerabilities.

1.3 Objectives

The main goals of this thesis is to improve and extend the open-source JavaScript Voxelizer engine, turning it into a maintainable and high-quality open-source project. A second goal will be to develop complimentary software for the Voxelizer project. This will be in the form of a cross platform desktop application and CLI, based on the Voxelizer engine, making it easy to voxelize 3D models.

In order to ensure the maintainability of the various software projects, automation is a critical component. Therefore a third goal will be to develop a GitHub Action for automating the API documentation generation process.

1.4 Scope

The main purpose of this project is to make it easy to conduct high quality voxelization of 3D models. Its scope is limited by the requirements specification defined in the Preliminary report. Complementary to these requirements, a backlog with user-stories has been created. See Appendix C.

It is important to note that the project does not primarily focus on speed of the voxelization algorithm. The targeted systems often run in an environment where resources are scarce. Performance is therefore of course important. However, usability is also extremely important. This thesis will mainly focus on providing easy access to high-quality voxelization, with reasonable performance. If speed is of the main concerns, it would be better to do the extra work of setting up a native solution (for example binvox [1]), often written in C/C++.

1.5 Systems overview

1.5.1 Voxel systems

The diagram in figure 1.1 shows how the different software repositories regarding voxelization interconnects. The green boxes represents the main software projects developed in conjunction with this thesis. During development, some components were generalized and extracted into a separate repository. This side project is represented by the blue box. The white box represents a third party library.

1.5.2 Automation systems

Figure 1.2 shows a diagram of the various automation repositories. This is mainly GitHub Actions, published to the GitHub Marketplace. The yellow box represents a main project. Throughout the project, it also became clear that some supportive actions needed to be created. These side projects are represented by the blue boxes. The white box represents third party software.

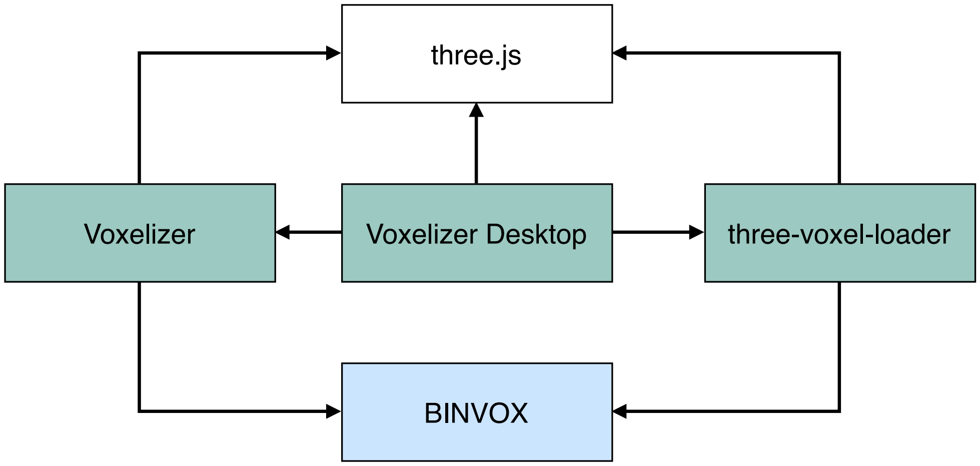


Figure 1.1: Voxel systems overview.

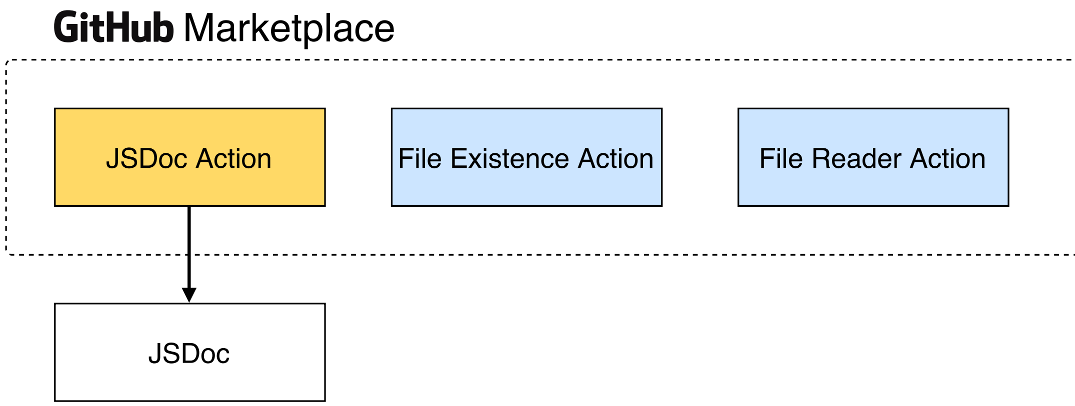


Figure 1.2: Automation systems overview.

1.6 Outline

The rest of the report is structured as follows.

Chapter 2 - Theory: Chapter two gives an introduction to the theoretical background that lies the foundation of this thesis.

Chapter 3 - Method: Contains a description of the methodology and materials that were considered throughout the project.

Chapter 4 - Result: Contains a description of the completed work.

Chapter 5 - Discussion: Discusses the achieved results, the execution of methodologies and tools, in addition to encountered difficulties.

Chapter 6 - Conclusions: This chapter presents an overall conclusion of the project, reviewing the objectives and the progress made.

Chapter 2

Theoretical basis

2.1 Agile methods

2.1.1 Scrum

Scrum [2] is an agile methodology. It is a lightweight, iterative and incremental framework for managing complex work. Scrum is one of the most popular agile methodologies used in the software business. The framework is mainly intended for developing information systems. Scrum defines several roles and different processes. Table 2.1 lists the various roles. An illustration of the Scrum process is shown in Figure 2.1.

At the hart of Scrum, there is a development team. This team is comprised of around three to nine people. The team collectively breaks down project tasks (user-stories) from the backlog into projects. These are then to be completed within a given time frame, so-called "sprints". A sprint usually lasts for about two weeks, to a month. Scrum is also highly focused in the communication between the involved persons, defining several short planning- and review-meetings. Table 2.2 lists a more detailed description of the various processes in Scrum.

Table 2.1: Roles in Scrum.

Role	**Description**
Product owner	Responsible for representing the stakeholders interests, and ensuring the product success.
Development team	The persons actually implementing the project tasks. They are also responsible for setting up the sprints and having daily stand up meetings.
Scrum Master	Person within the agile development team. The Scrum Master is to serve as a facilitator for the development team. A good Scrum Master should make himself/herself superfluous.

Table 2.2: Scrum processes.

Process	**Description**
Sprint planning	The team selects user-stories from the product backlog. Normally, story points are assigned to the user stories, based on the effort needed to implement it.
Sprint	The actual implementation of the selected user stories. This should result in the next increment (or version) of the product.
Stand-up meeting	Each day, the team then has a 10-15 minute long "stand-up" meeting. This gives the team an opportunity to plan for the day, as well as catching up on the progress done by the other team members.
Sprint review	An evaluation process of what was and what was not finished in the last sprint. The results of the sprint are also presented to the product owner and the various stakeholders.
Sprint retrospect	The team will have a meeting for assessing the completed sprint. This is an important step, as this presents an opportunity to find out how to improve the process of the next sprint.

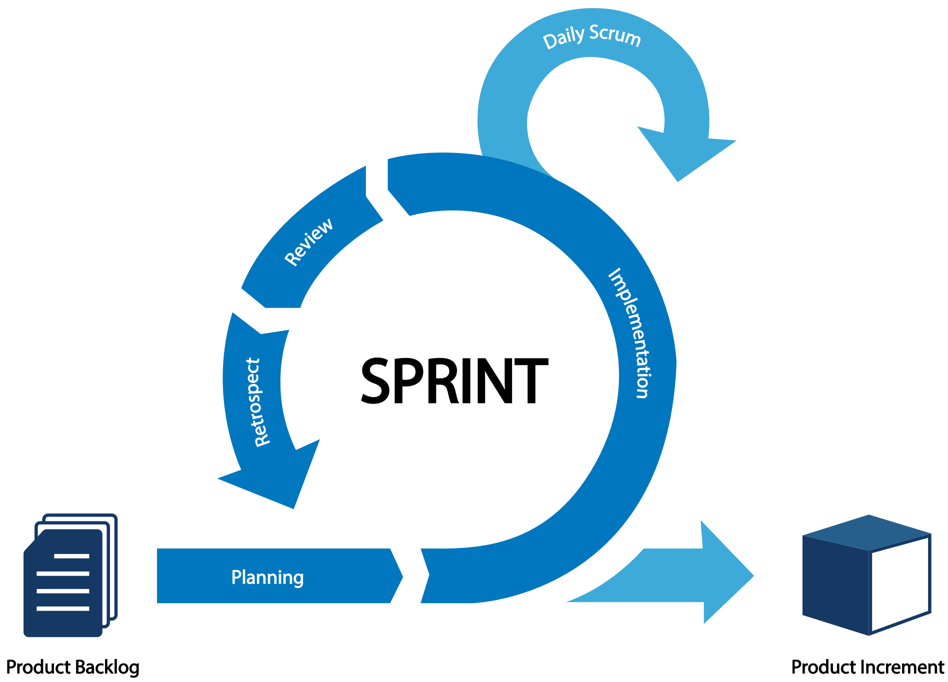


Figure 2.1: Scrum workflow.

2.1.2 Kanban

Kanban [3] is a lean software development methodology, based on the lean methodologies. These have been highly successfully in manufacturing processes. At the center of Kanban is a just-in-time (JIT) process. No new tasks are to be started unless there is a need for it. Further, once a capacity limit has been reached, no further task can be started on. When a task has been resolved, a new one can be started. This can simply be described as a pulling workflow. Kanban also often focuses on the visualization of tasks. This is normally done with a Kanban Board. In contrast to Scrum, Kanban allows for the software to be developed in one large development cycle. It is much more flexible, and does not define any roles.

2.2 Git

Git [4] is a type of distributed version control system (VCS), originally created by Linus Torvalds in 2005. Git is free and open-source, and is today the most popularly used VCS. It is fast and efficient, able to handle everything from small hobby projects to giant projects like the Linux kernel. Every Git directory is a complete repository with history and full version-tracking abilities. It does not need access to internet, nor a central server in order to work.

2.2.1 GitFlow

GitFlow is a popular branching model for Git, created by Vincent Driessen [5]. Figure 2.2 displays an example git history, adhering to the GitFlow branching style. GitFlow truly excels in parallel development. It is extremely well suited for collaboration and scales well. It also provide an efficient and predictable merging flow, making it easy to customize workflows for various needs.

Development of new features are done in **feature branches**. These branch off from the **development branch**, often named **develop**, reflecting the current state of "development". When a feature is done, it is merged back into the **development branch**. When it is time for a new release, a **release branch** is created based on the **development branch**. In this branch, finishing touches can be made, like bumping up the version numbers, etc. When approved, the **release branch** is then merged into a **master branch**, and also back into the **development branch**. The **master branch** only contains released code. In the event of an emergency, a **hotfix branch** can be used. This provides a shortcut for implementing critical fixes. A **hotfix branch** branches directly from master. When finished, the **hotfix** is merged into both **master** and **develop**.

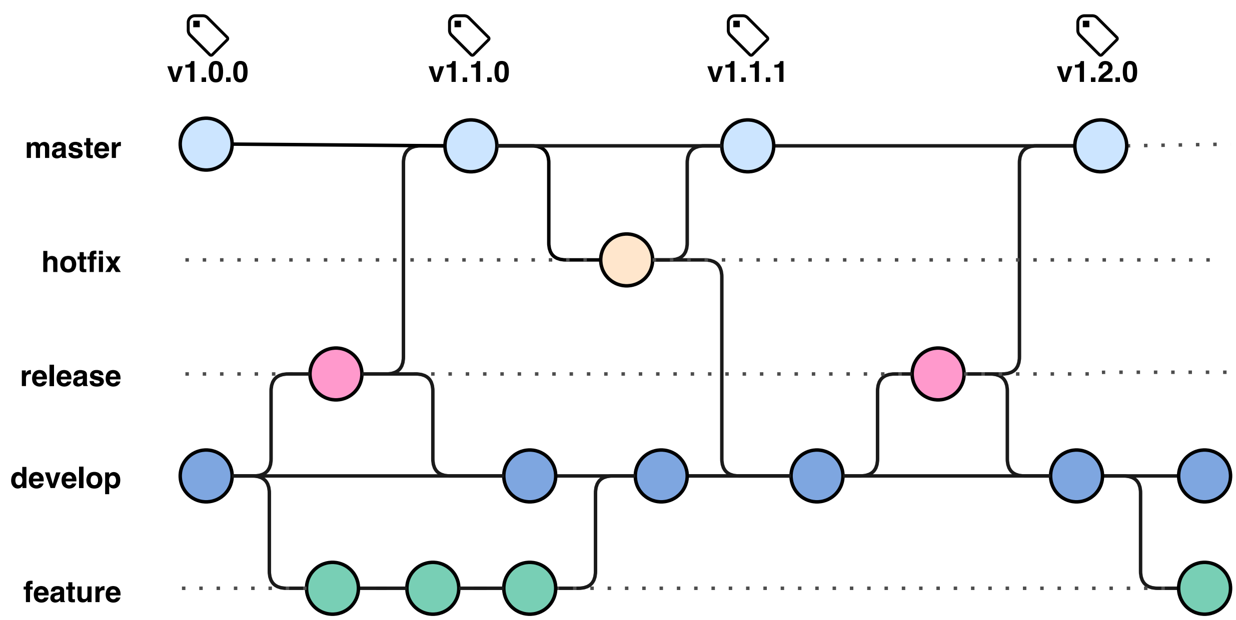


Figure 2.2: GitFlow branching model example.

2.3 GitHub

GitHub is primarily a hosting service for Git repositories. The company was acquired by Microsoft in 2019. In addition to repository hosting, GitHub provides a range of different services through it's web-based GUI. This includes both wikis, access controls, simple task management tools, statistics, automation capabilities and websites hosting.

2.3.1 GitHub Actions

GitHub Actions [6] is a fairly very new service provided by GitHub. It enables users to automate software workflows, effectively providing a high quality CI/CD for free [1]. It is also possible to set up self-hosted runners. GitHub Actions makes it possible to both build, test and deploy code directly from within GitHub.

To setup and configure an automated process, so-called workflows needs to be defined. These are made up of one or more jobs. The actual workflow has to be defined in a YAML file. This file needs to be created and placed in a repository hosted on GitHub. Consult the documentation available at GitHub for the appropriate workflow syntax for creating such YAML files [7].

2.3.2 GitHub Pages

Github provides its users with a public webpage hosting service. This is named GitHub Pages [8]. User are able to serve static websites directly from their repository hosted on GitHub. Normally, a GitHub Pages site is published by pushing static files to a specific branch named *gh-pages*.

2.4 HyperText Markup Language (HTML)

HTML is a markup language, originally defined by Tim Berners-Lee and Robert Caillau in 1989. It is primarily used for documents on the web, intended to be displayed in a web-browser. It is used for structuring and formatting information. HTML can be used in conjunction with other

[1] GitHub Actions usage is completely free for public repositories. For private repositories, depending on the subscription plan, some thousand minutes of free usage is provided each month.

web technologies such as Cascading Style Sheets (CSS) and scripting languages like JavaScript, in order to either style, or dynamically change and alter the contents of a web page. The currently latest release of the language is HTML5.

2.5 Cascading Style Sheets (CSS)

CSS is short for Cascading Style Sheets. CSS is a programming language in order to describe how HTML elements in a HTML file are to be rendered. Everything from the boldness of a headline, to the background of the entire page.

2.6 JavaScript

JavaScript is a lightweight interpreted programming language. The language is prototype-based, a type of object-oriented programming where properties and methods are added to an instance of an implicitly defined class [9]. JavaScript does not provide any type-checking.

JavaScript was developed by Brendan Eichand and released in 1995 [10]. Initially, it was designed to be a small scripting language for enabling interaction with web pages. A standardization effort of JavaScript, led by Ecma International [11], lead to the ECMAScript specification that the modern JavaScript language conforms to. Since then, the language has evolved rapidly and gained massive in popularity. It has become the de-facto language for adding dynamic behavior to HTML. As of may 2020, JavaScript is among the top ten programming languages according to the TIOBE index [12].

Originally, JavaScript engines were mainly used in browser environments. However, with the development of Node.js in early 2009, this was dramatically changed. Node.js provides a runtime environment for executing JavaScript outside of a web browser. It set the stage for server-side JavaScript programs. In January 2010, a package manager named npm [13] was released for Node.js. This made it easy for developers to share and reuse source code.

The size of JavaScript programs has increased massively in size. With increased size, so does the complexity of the code. However, JavaScript had very limited functionality in terms of splitting a program up in smaller modules for use with the browser. Maintaining large codebases was a nightmare. It therefore became apparent that a way of breaking down a JavaScript program into smaller modules was needed. Several open-source module systems were therefore developed by the community in order to tackle this problem.

2.6.1 Module systems

- **CommonJS** is a module specification meant for JavaScript outside the browser. It is mainly used in Node.js, and hence it is one of the most popularly used module definitions. The modules are mainly imported and exported with the keywords "require" and "module.exports".

- **Asynchronous module definition**, or more commonly known as AMD, is a JavaScript module definition intended for the browser. It defines an API for defining code as modules, including their dependencies. AMD also has the capability of loading modules asynchronously. The most popular AMD module loader is named RequireJS [14].

- **Universal Module Definition**, abbreviated UMD, is a module definition wrapper to be able to use various module systems [15]. Be it in the browser or in Node.js. It is compatible with both CommonJS and AMD.

- **JavaScript modules**, or ES Modules, are a language native module system, introduced with ECMAScript 2015 (ES6) in 2015. The implementation is relatively new, so a lot of libraries, frameworks and packages does not support this yet. Still, most browsers have already implemented support for this [16].

2.6.2 Transpilation

Due to the many versions of JavaScript, or more specifically ECMAScript, like ES5, ES6 and ES7, compatibility is an issue. Not all browsers and environments support the latest ECMAScript versions. Tools like Babel [17] has been developed in order to transpile JavaScript to a specific version. The most common transpilation target, supported by all the major browsers is ECMAScript 5 (ES5).

2.6.3 Bundling

JavaScript bundling is an optimization technique to combine separate resource files into one file. This is done in order to reduce the number of HTTP requests required for a page to load. Several bundlers are able to do so called tree-shaking, dramatically reducing the size of the finished bundle. In addition to the performance gain, bundling is also often done in order to develop a JavaScript application in separate files, effectively employing a form of module system. The bundlers often use one or more of the popular module systems like UMD and ES modules. There are three main actors in terms of bundling JavaScript.

- **Webpack** [18] is a module bundler for JavaScript. However, it is also able to transform front-end assets like HTML, CSS, and images. Webpack is mostly used for bundling JS applications, and is highly extendible. It also provides a way to bundle an application for Node.js to be used in the Browser [2]. However, as of may 2020, it does not support exporting ES Modules.

- **Rollup** [19] is a module bundler primarily focusing on JavaScript libraries. It has a lot of similarities with Webpack. However, it is a bit lighter and provides exporting support for ES Modules.

- **Browserify** [20] is a lightweight module bundler for enabling the use of CommonJS syntax in the Browser.

2.7 TypeScript

TypeScript [21] is a typed superset of JavaScript that compiles to plain JavaScript. It is open source, and primarily developed and maintained by Microsoft [22]. TypeScript provides optional static typing to the JavaScript language. The TypeScript compile also includes support for the latest ECMAScript features.

[2] This requires that no Node.js specific APIs are used. Alternatively, polyfills could be supplied.

2.8 JavaScript Object Notation (JSON)

JavaScript Object Notation, better known as JSON, is a lightweight data interchange format. It is easy for both humans and machines to read and write. The data is stored as attribute–value pairs. An example is shown in Program Code 2.1.

Program Code 2.1: Example JSON data

```
{
    "name": "A. Storhaug",
    "age": 22,
    "email": "andr3.storhaug@gmail.com",
    "url": "https://github.com/andstor"
}
```

2.9 JSDoc

JSDoc is markup language for annotating JavaScript source code files. The JSDoc specification was released in 1999. Today it has become the de-facto JavaScript documentation language. It is for example used in projects like the Google Closure Compiler [23] by Google. Since JavaScript has no type-checking, JSDoc is able to patch some of this inconvenience. Figure 2.2 shows an example of JSDoc code, describing a soda bottle class implementation. By using various tools, one is able to generate documentation in formats like HTML and RTF. JSDoc 3 is the current version of the original companion documentation generation tool for JSDoc. JSDoc 3 [24], also referred to as just JSDoc, is the most used tool for programmatically generating JavaScript documentation. It has a wast feature set, even allowing users to create customized themes, known as templates. Currently, JSDoc is used by more than 38.800 public projects [25], and has over 10.600 stars on GitHub [26].

Program Code 2.2: JSDoc example code

```
/**
 * Represents a soda bottle.
 * @constructor
 * @param {string} brand The brand of the soda.
```

```
 * @param {number} size The size of the soda in deciliters.
 */
function SodaBottle(brand, size) {
}
```

2.10 Tools and libraries

2.10.1 WebGL

WebGL (Web Graphics Library) [27] is a JavaScript API for rendering interactive high-performance 3D and 2D graphics in a web browser. WebGL uses OpenGl ES [28], a subset of OpenGL. WebGL makes it possible to take advantage of hardware graphics acceleration provided by the user's device. For actually displaying the graphics in the browser, HTML5 <canvas> elements are used.

2.10.2 three.js

three.js[29] is a cross-browser JavaScript library for creating and displaying 3D computer graphics in a web browser. It is open-source and licensed under the MIT license. three.js uses WebGL under the hood. The library abstracts away a lot of tedious manual labour, like the setup of WebGL, construction of vertices, faces, etc. three.js provides the user with an easy API for directly constructing three-dimensional objects like boxes, spheres and toruses, as well as easy camera controls. The library also includes a vast set of shaders, making it very simple to make use of high quality materials. three.js is one of the most popular 3D graphics JavaScript library for use in the browser, as can be seen on its GitHub repo [29].

2.10.3 ndarray

ndarray is an open-source JavaScript package providing modular multidimensional arrays, written by Mikola Lysenko in 2013. In short, ndarray implement a higher dimensional views of 1D arrays. The 1D array can either be a normal JavaScript Array, or a JavaScript typed array. MDN defines typed arrays as "array-like objects that provide a mechanism for reading and writing raw binary data in memory buffers." (Mozilla Developer Network [30]). Mainly, they are used for maximizing efficiency and reducing memory footprint. However, typed arrays are normally quite difficult to work with in JavaScript. Multidimensional typed arrays even more so. ndarray provides a simple but powerful API, making use of multidimensional typed arrays easy. strided arrays??

2.10.4 Electron

Electron [31] is an open-source framework developed and maintained by GitHub [32]. It allows one to build cross-platform desktop applications with JavaScript, HTML and CSS. Electron is used by thousands of people, and apps like Visual Studio Code [33], Facebook Messenger [34] and Microsoft Teams [35] are all made with Electron.

In short, the Electron Application architecture is as follows. An Electron application starts by running a package.json's main script. As shown in Figure 2.3, this creates a process that is called the main process. From within this process, a GUI can be displayed by creating web pages. Electron uses Chromium for creating web pages Google [36]. This means that Chromiums multi-process architecture is also available. Every generated web page is therefore run in its own process. These processes are called renderer processes. Communication between the main process and a rendering process is done using inter-process communication, or IPC. Electron also makes it possible to use Node.js APIs, effectively allowing lower level operating system interactions.

2.10.5 React

React [37] is a JavaScript library for building user interfaces, created by Facebook [38]. It is based around components, where each component manage its own state. These are then composed together, enabling the creation of intricate and complex UIs. The library also implements its own syntax extension to JavaScript. It is named JavaScript XML, or the more popular used term - JSX. In addition to this, there exists a vast ecosystem of plugins for React, greatly simplifying the implementation of everything from localization to state management.

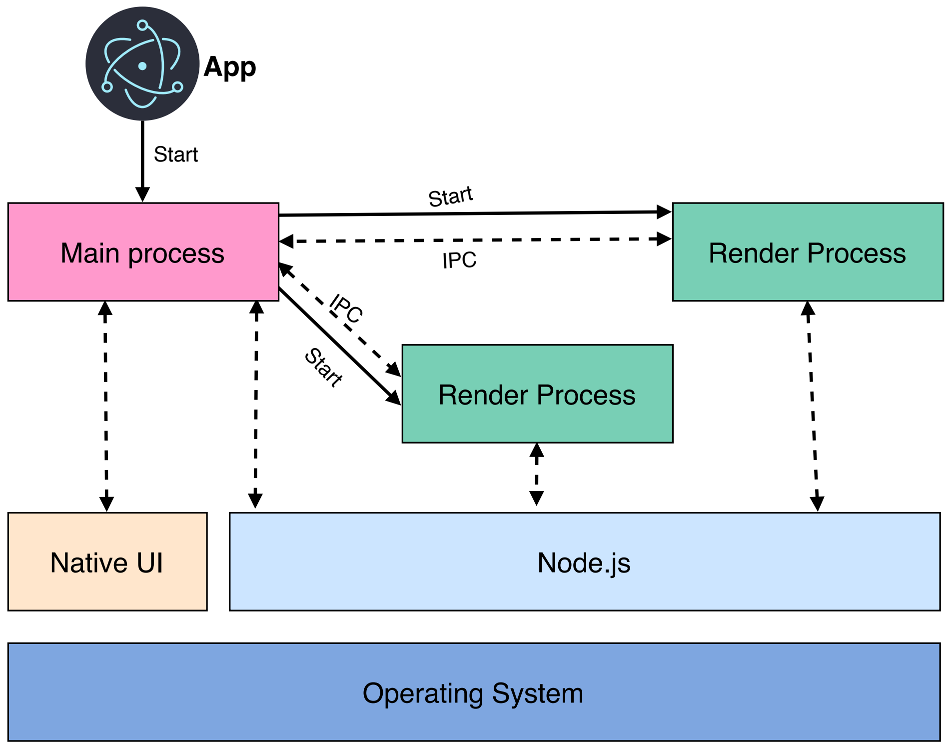


Figure 2.3: Electron architecture.

2.10.6 Semmle LGTM

LGTM by Semmle [39] is a web service providing code security analysis. The service is free for open-source projects. LGTM integrates with sites like GitHub and Bitbucket, and is able to analyze projects written in Java, Python, JavaScript, TypeScript, C#, Go, C and C++. It seeks out to combat the manual process of finding vulnerabilities. By catching them at an early stage, one can prevent vulnerabilities from reaching production. LGTM is based on large community of top security researchers, making it possible to help developers ship secure code [40].

2.10.7 Coveralls

Coveralls [41] is a web service for code testing coverage. It enables one to track a projects code coverage over time, providing valuable insight in a projects testing suite. Coveralls also features close integration with GitHub, enabling pull request coverage reviews.

2.11 3D computer graphics

3D computer graphics refers to three-dimensional representation of geometric data in computers, normally to be rendered into a two-dimensional image. The finalized render may be saved or displayed on a screen in realtime. The geometric data is usually a 3D model, stored in an appropriate file format. The most basic polygon primitives in computer graphics includes vertices, edges and faces. A vertex is simply point in space. An edge is a connection between two vertices. A face is a closed set of edges. Figure 2.4 shows a yellow triangle with the appropriate vertex, edge and face labels. These primitives together defines a polyhedral. Polyhedrons can then be further grouped together into a mesh. The most common type of polygon mesh is a triangular mesh. This is a mesh comprised of only triangles. Figure 2.4 shows an illustration of a section of a triangular mesh.

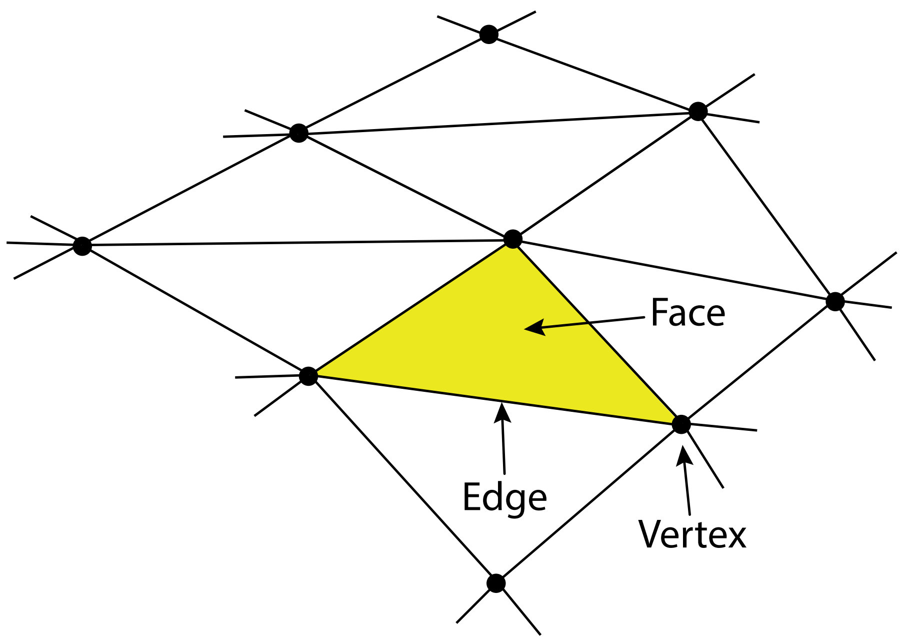


Figure 2.4: Triangular mesh

2.11.1 Texture maps

A texture map is an image which is applied, or mapped, onto the surface of a geometry. The images is often in the form of a bitmap image or a procedural texture. Texture mapping, or

UV mapping, is the process of projecting an actual 2D image onto a 3D model. The technique was initially developed by Edwin Catmull in 1974 [42]. UVs are two-dimensional texture coordinates, assigned to every vertex in a polygon. They are essential in terms of describing how an image gets applied onto a geometry. Figure 2.5 shows an illustrative example of how a 2D image gets "wrapped" around a 3D model. A lot of 3D modeling software are able to do the UV unwrapping automatically, for example Blender [43]. It is also possible to map a finalized render into a surface texture, a process known as baking [44]. This is primarily used as an optimization technique.

Figure 2.5: Texture mapping illustration.

2.11.2 Ray casting

Ray casting is the concept of use of ray–surface intersection tests to solve a variety of problems in 3D computer graphics and computational geometry. The first use of the term ray casting was made by Scott Roth, in a paper from 1982 titled "Ray casting for modeling solids" [45]. Raycasting is demonstrated in Figure 2.6. A ray is directed towards an object. If it crosses a face, an intersection is registered.

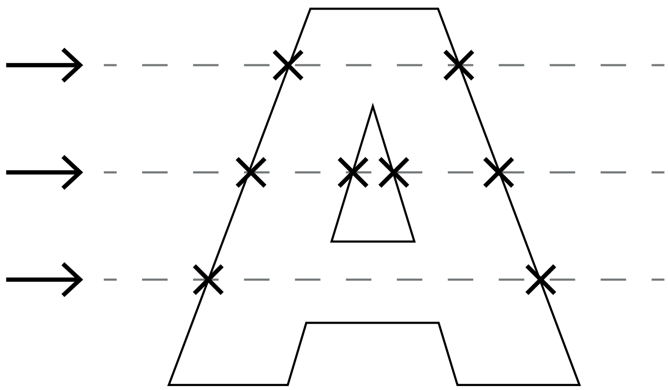


Figure 2.6: Raycasting intersections example.

2.12 Acceleration data structures

2.12.1 Octrees

An octree is a type of tree structure. Each internal node of an octree has exactly eight children. An octree is most commonly used for partitioning three-dimensional space. This is done by recursively subdividing the space into eight octants. Note that depending on the number of recursive subdivisions, an octree may contain multiple objects in its leaf nodes. Figure 2.7 shows an example of an octree with three levels. Octrees are very commonly used in 3D computer graphics. Another common use case of octrees is for storing voxel data.

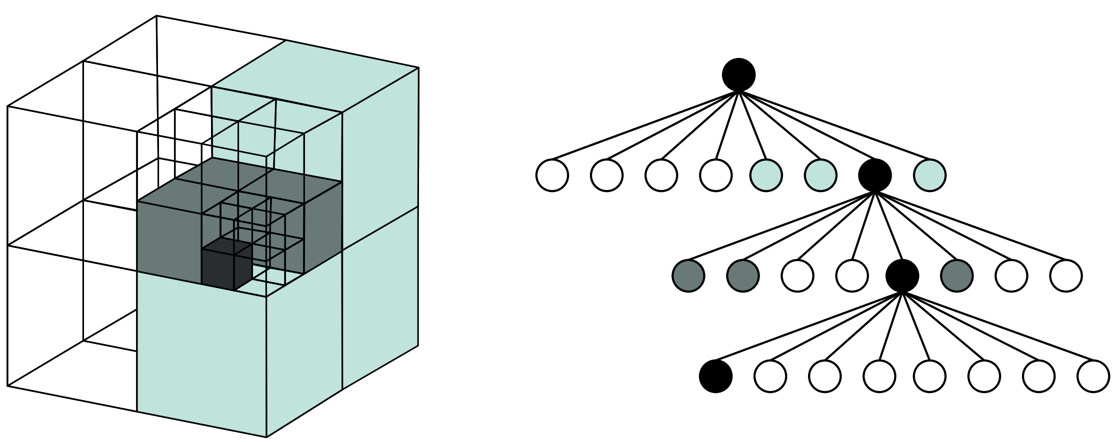


Figure 2.7: Example of an octree with three levels.

2.12.2 Bounding volume hierarchy

A bounding volume hierarchy, abbreviated BVH, is a tree structure on a set of geometric objects. A BVH construction algorithm partitions the actual objects. The objects are wrapped in a so-called bounding volume, forming the leaves of the tree. These are then grouped together into a larger bounding volume. This process is then repeated in a recursive manner. The result is a tree structure with one single bounding volume as the root node. An example of a BVH is shown in Figure 2.8. BVH are often used for accelerating collision detection and raytracing.

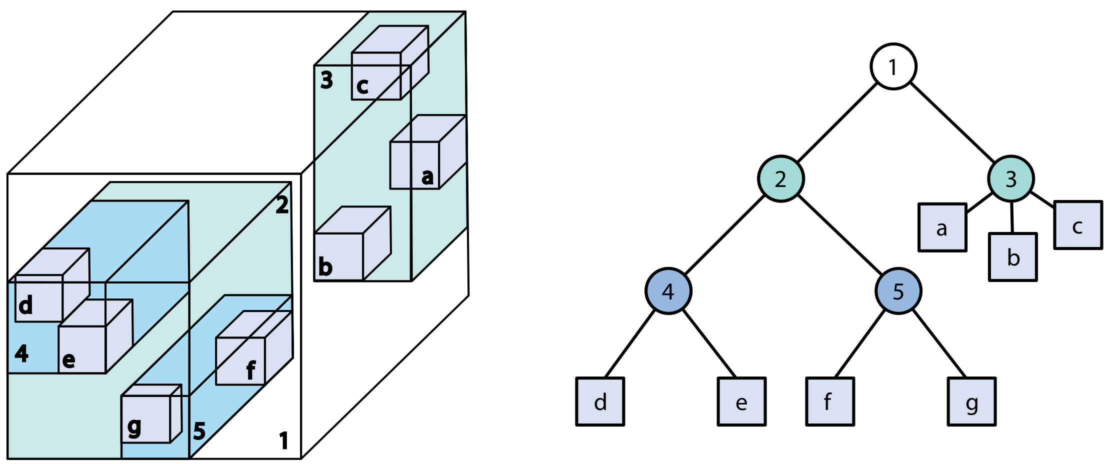


Figure 2.8: Example of an BVH. Replica of figure from MacDonald [46].

2.13 Voxel

A voxel is the three-dimensional analogue of a pixel [47]. It represents a single data point in a regularly spaced three-dimensional grid. Figure 2.9 shows an illustration of three voxels, where one of the voxels are marked with blue color. A very common use of voxels are in medical imaging, for example datasets produced by a CT scan. Other areas where voxels are commonly used includes simulations and for representing terrain in games.

2.14 Voxelizer v0.1.3

Voxelizer v0.1.3 [48] is a JavaScript engine (or library) for conducting voxelization of 3D models. It was written by me, André Storhaug, in 2019. Version 0.1.3 features a relatively simple voxeliza-

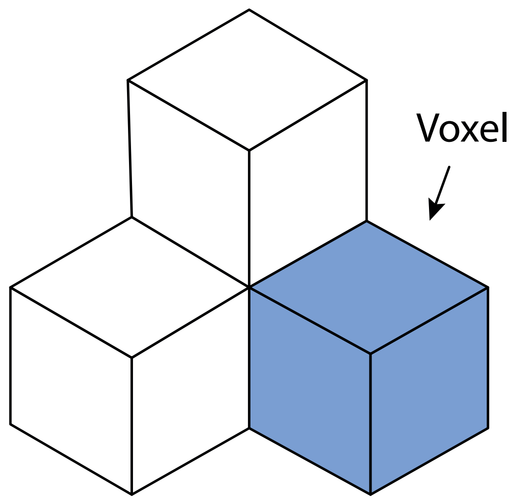
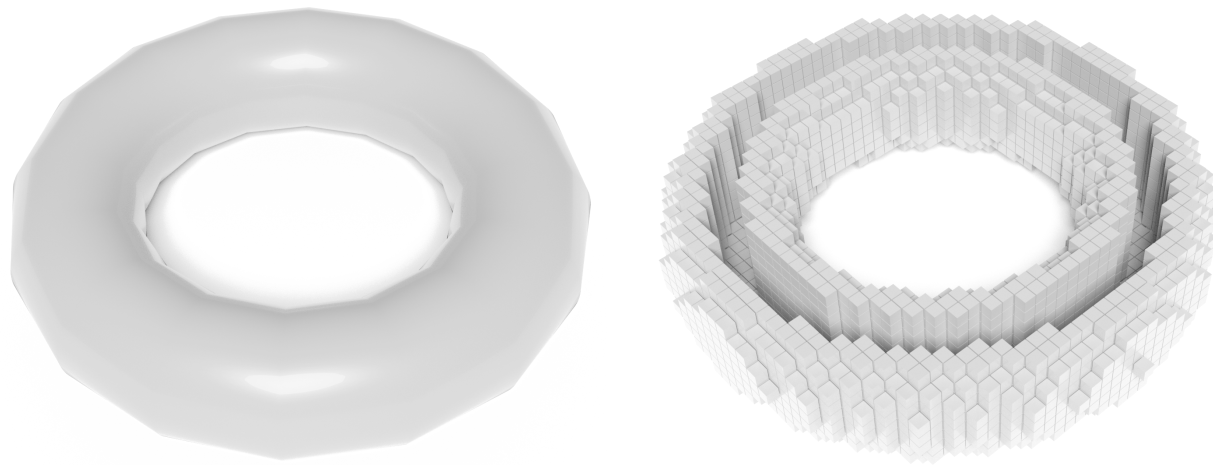


Figure 2.9: Three voxels.

tion algorithm which is based on raycasting. The sampling algorithm tries to produce a filled volume-representation of a supplied 3D model. It samples the front and the back of the model, combines the two results together and tries to fill the in-between gap. The 3D model loading capabilities of the program is limited to plain OBJ files. In terms of exporting, the software is able to output a 3D JavaScript array (nested arrays).

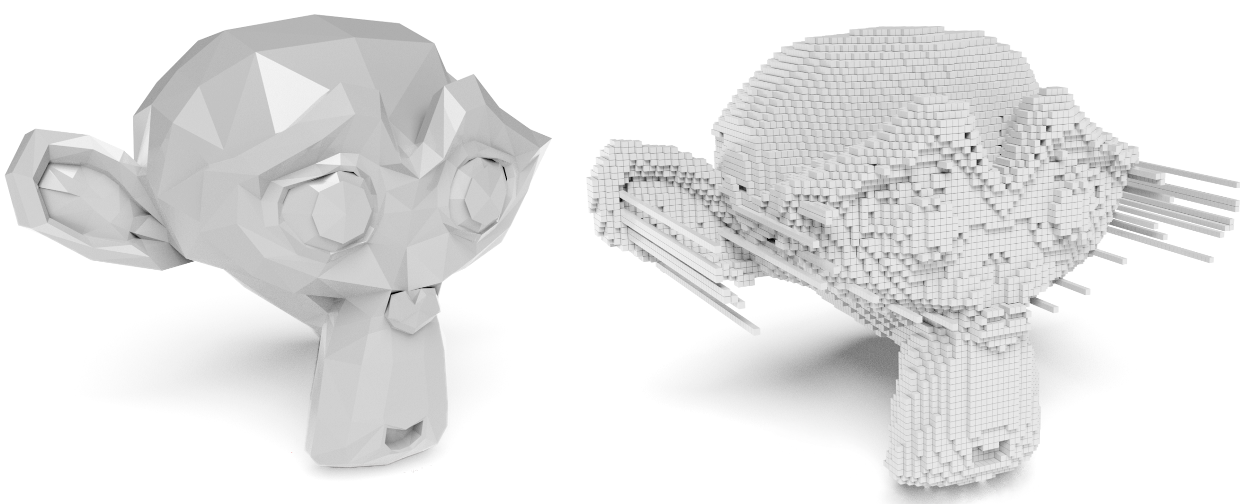
The engine is using ES6 features, hence it is transpiled with Babel (see Section 2.6.2). However, is not bundled. It is therefore not possible to use the program out of the box in a browser. One is limited to Node.js, or setting up a build system involving a module bundler like Webpack or Rollup, as described in Section 2.6.3. The source code is messy, and it is very hard to extend functionality. Especially due to a severe lack of documentation.

Voxelizer v0.1.3 produces unsatisfactory voxelization results. Firstly, several of the voxelizations contains holes. This can be clearly seen in Figure 2.10. A voxelization with holes often renders the voxelization useless. Secondly, a lot of artifacts are often generated, severely degrading the results. This is shown in Figure 2.11, where long strains of voxels appear in the front of the model. This is especially pronounced around the ears of the monkey. Thirdly, the software is only able to produce a filled voxelization result. This is mainly due to the fact that the model is only sampled from the front and back. The other sides of the model are not taken into account. This means that shell voxelization is not an option, and details may be lost.



(a) Original torus mesh. (b) Voxelized result.

Figure 2.10: Voxelization of a torus with Voxelizer v0.1.3. The voxelization is done with a resolution of 40.



(a) Original monkey mesh. (b) Voxelized result.

Figure 2.11: Voxelization of a monkey with Voxelizer v0.1.3. The voxelization is done with a resolution of 100.

Chapter 3

Materials and methods

3.1 Tools and libraries

3.1.1 JavaScript

JavaScript has mainly been chosen as the base implementation language for all the projects because of its cross platform compatibility and popularity.

3.1.2 npm

npm is used as the main package publishing- and installation-tool. Also, all published packages are published to the npm package registry. Alternatively, the new GitHub package registry could be used instead. However, npm is the both the most stable and popular JavaScript package registry available.

3.1.3 GitHub Actions

For automation, several platforms are available. One of the more popular ones includes Travis CI [49]. However, GitHub Actions is used instead. This is because GitHub Actions are deeply integrated into GitHub. It also has a unique concept of Actions, as described in Section 2.3.1.

3.1.4 Build tools

Several build tools are used. This includes:

- **Rollup** for bundling the three-voxel-loader and BINVOX projects.
- **Webpack** for bundling the Voxelizer engine.
- **Babel** for transpiring JavaScript to ES5.
- **electron-builder** for building and preparing Electron apps for distribution.

3.1.5 Third party libraries

Several third party libraries and frameworks are used. A short summary of why these are put to use is now provided.

- **three.js** provides a simple and powerful 3D graphics library. It also includes raycasting functionality.
- **sparse-octree** is used for generating octrees.
- **three-bvh-mesh** is used for improving the raycasting functionality of three.js.
- **format-vox** provides tools for parsing VOX files.
- **dat.gui** makes it easy to create input controls.
- **ndarray** provides multidimensional views of 1D typed arrays.
- **Jest** is used as testing framework.

3.2 Working methodology

3.2.1 Scrum

Even though this is a one man project, i have tried to adapt the scrum methodology. The main changes done are regarding the various meetings defined by Scrum, as described in Section

2.1.1. Sprint planning has been conducted normally. User stories are selected from the backlog, and story points are assigned. The sprint is then executed. Instead of daily stand-up meetings with a team, I have have set aside 10-15 minutes for planning every day. After a sprint is finished, a sprint review is done. Then, a sprint retrospect is conducted in order to improve the next sprint. This includes both investigating sprint burn-down charts and other statistics. These charts are available in Appendix **??**.

3.2.2 GitFlow

The branching strategy used during development is GitFlow, as described in Section 2.2.1. This creates a consistent development and merging flow, making it easy to implement automation in the various project. Since GitFlow is very popular, a lot of potential contributors are already familiar with the strategy. Providing familiar guidelines helps in keeping the projects tidy and secure.

3.2.3 Semantic versioning

All the created projects are enforcing Semantic Versioning. This makes the use of the software predictable in terms of compatibility and potential breaking changes.

3.3 three-voxel-loader

NEED UML DIAGRAMS

The main goal for the three-voxel-loader is to generate a 3D mesh based on voxel data. The next subsections will present a walkthrough of how the plugin is implemented, including the various design choices made.

3.3.1 Internal data structure

For the plugin to be able support loading of various voxel data formats, an internal data structure is used. This serves as an interface for the processing of various voxel data internally in the plugin. The chosen data structure is an octree, as described in Section 2.12.1. The actual implementation of the octree is done with the sparse-octree library by Raoul van Rüschen [50]. There are several reasons behind the choice of using an octree.

Voxel data normally consists of very large amounts of data. One of the main limitations for how big a dataset can be, is the amount of available computer memory (RAM). The memory footprint of the plugin is therefore a big concern, especially since the targeted runtime environments are often placing further restrictions on the available memory resources. Voxel data normally contains very large amounts of empty space, or "air". This data is not needed for generating the polygon mesh. Only the data about the actual voxels and their locations are of interest. The location of the voxels are normally not stored explicitly, but rather derived by their relative location to neighboring voxel cells. An octree is especially well suited for this purpose. Since an octree is based on partitioning of space, large amounts of this empty space in the voxel data can be discarded. This works especially well if the voxel data is clustered.

Octrees also makes it easy to implement a Level of Detail (LOD) mechanism. By determining the desired depth of the octree, one are able to simplify the detail of the voxel data. This is very valuable, as generating mesh geometry for every voxel is placing high stress on the available hardware resources. Being able to control the LOD, this can be very effective in terms of simplifying the resulting mesh.

3.3.2 Loading voxel data

In order to actually load the voxel data (by generating an octree), several loader classes have been created. These classes all extend the *loader* class defined in three.js, providing both consistency and tight integration with library. It also ensures that extending the support for more file loaders in the future is easy. Finally, a factory pattern has been implemented for getting and instantiating the desired voxel loader. This makes it able to define an easy-to-use API, where the user only needs to supply the actual voxel data and the corresponding format.

The currently implemented loaders supports several file formats, including XML, VOX and BINVOX. It is also possible to import plain 3D arrays. Several of the loaders also supports color of the voxels. Following is a brief description of these loaders.

- **XML** - XML is an incredible versatile file format. For implementing the XML loading, the native JavaScript DOM parser is used for the actual parsing of the XML data. The format supports color data. The required format of the XML document structure is described on the GitHub wiki page for the plugin. PROVIDE LINK HERE!!!!!

- **VOX** - VOX files are loaded with a third party package named *format-vox* [51]. The VOX file format is provided by MagicaVoxel, a popular voxel graphics editor. VOX files supports color data.

- **BINVOX** - BINVOX [52] is one of the more popular voxel data file formats. BINVOX is the file format used by the binvox [1] voxelization software. A separate repository named binvox [53] has been created for handling BINVOX files. See Section 3.5

- **3D array** - The 3D array loader is implemented by simply iterating the multidimensional array. For loading color data, a 4D array with RGB values has to be supplied along the 3D array.

3.3.3 Visualization

The most intuitive way to visualize a voxel is in the form of a cube. BoxBufferGeometry from three.js has therefore been used to generate the individual 3D visualization of the voxels. However, one BoxBuffer geometry consists of no less than twelve triangles. The number of triangles generated for visualizing the mesh is therefore twelve times the number of voxels. One of the more time consuming operations in terms of actually displaying the 3D graphics, is the number of draw calls made to the graphics API. In order to limit this, all the generated box meshes are merged into one big mesh. For actually coloring the voxels, color is applied to the vertices of the generated box meshes.

3.3.4 Debugging

For actually developing and testing the plugin, a HTML page was created. The page includes basic setup of three.js, alongside various input controls for inspecting and testing the three-voxel-loader plugin. The input controls are provided by a lightweight JavaScript controller library named dat.gui [54]. In the end, the debugging solution were polished and deployed to GitHub Pages, serving as an example for the various functionality the plugin provides.

LINK TO GITHUB PAGES EXAMPLE

3.3.5 Building

The plugin is bundled with Rollup. This produces excellent bundles, with support for both UMD and ES Modules. See Section 2.6.3 for more details on Rollup, and Section 2.6.1 for UMD and ES Modules. three-voxel-loader makes use of ES6 features. All source files are therefore transpiled to ES5 with babel.

3.4 Voxelizer

Short intro Three.js provides raycasting out of the box. However, the raycasting method implemented It would be more efficient to implement the raycasting directly without the use of three.js. However, this would be laboursome. However, the main reason for using the three.js library is based on the popularity of the project. the ecosystem of three.js is vast, providing everything from file loaders to ... This, combined with excellent documentation, should make it very easy to produce a 3D model in three.js. This sets the stage for further processing of the 3D models with the Voxelizer engine.

3.4.1 Systems overview

UML diagram of system here

The system is broken down in several modules (folders). This includes:

- **core** - The core module contains core APIs. This provides the main user API for conducting the voxelization.
- **algorithms** - The algorithm module defines the algorithm system. This is in charge for the actual sampling of the 3D models.

- **color** - The coloring system is found in the color module. This manages the color extraction for supporting color voxelization.
- **volume** - The volume module mainly contains a wrapper class for providing a consistent interface for interacting with volumes throughout the application.
- **exporters** - The exporters module is made up of various exporter classes. These enables the engine to export the voxel data into many different formats.
- **utils** - This module contains various utility functions.

3.4.2 Algorithm system

Voxelizer implements an algorithm system that makes it easy to extend the voxelization support. Be it new algorithm types or options. The system mainly consists of two base abstract algorithm classes. One for plain voxelization, and one which is colorable. By simply extending the appropriate base class, a new algorithm can be defined. Further, a factory pattern has been implemented for selecting the appropriate algorithm.

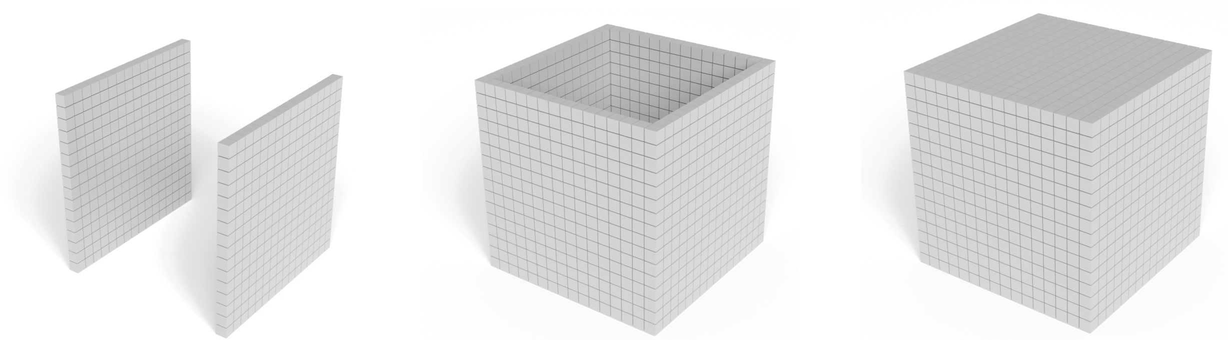
Since the generated voxel data can be huge, an efficient internal data structure needs to be used. Here is two main concerns to take into account. The first is the memory footprint. In order to be able to do high resolution voxelizations, a limiting factor is the amount of available system memory. A second and bigger concern is speed. The JavaScript engines are able to do quite a lot of optimization. By using the JavaScript language in clever ways, quite high processing speeds can be achieved.

The old Voxelizer v0.1.3 used normal JavaScript arrays which was nested. These arrays grows and shrink dynamically, potentially resulting in slow performance. Although, the JavaScript engines are often able to optimize the execution quite a bit, resulting in decent speeds. However, in order to comply with both the memory and performance requirements, typed arrays have been used instead. More specifically, only one large one-dimensional typed array is used. This is a more low level data type than normal arrays, providing mechanism for reading and writing raw binary data in memory buffers. In order to support multidimensionality, the efficient third party library ndarray is used. See Section 2.10.3 for more details on ndarray.

3.4.3 Raycasting algorithm

The new and upgraded voxelization algorithm is mainly based on raycasting (see Section 2.11.2). Following is a description of how the algorithm works.

First, several preparations are made on the mesh. Firstly, the mesh is centered at the origin. Then the mesh is manipulated to include both front and back faces. This ensures that raycasting against the mesh will result in a hit against both front and back sides of the model. This means that it is not needed to raycast from all 6 sides of the mesh. Hence, the algorithm samples the mesh from the front, left and top sides. These results are then merged together, and returned to the user in form of an ndarray. Figure 3.1 illustrates how these three results are merged, resulting in a complete surface shell representation.



(a) Front side sample.

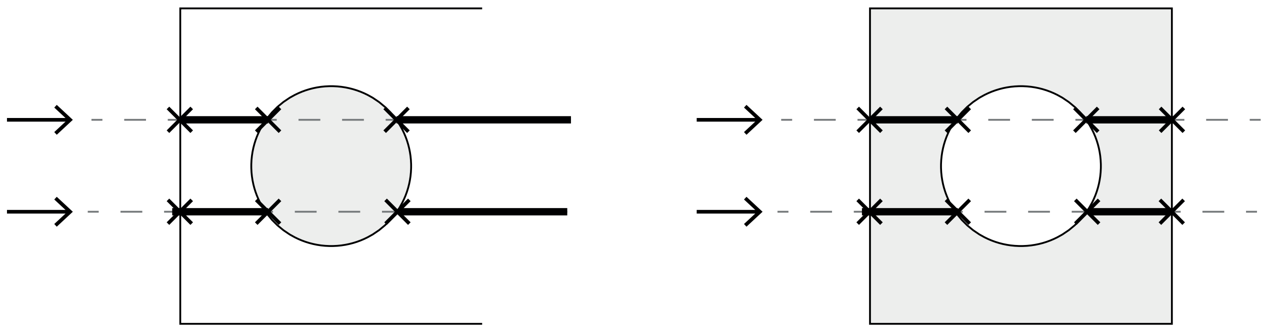
(b) Front and left side samples merged together.

(c) Front, left and top side samples merged together.

Figure 3.1: Merging of voxel samplings.

The algorithm also has an option for producing filled, or solid, results. This is achieved by interpreting the first raycast intersect as the surface of the object. From this point will everything be considered "inside" the object. When a second intersect is detected, the state is changed to be "outside" the object. A new hit would indicate "inside", and so on. This works very well with a watertight 3D model, as can be seen from figure 3.2a. However, when trying to fill an object which is not watertight, this can result in severe inaccuracies. This can be seen in figure 3.2b

(Remove watertight model?)



(a) Non-watertight 3D model cross section.

(b) Watertight 3D model cross section.

Figure 3.2: Solid (voxelization) filling of 3D model cross section.

3.4.4 three.js optimization

The raycasting functionality, used by the raycasting algorithm described in Section 3.4.3, is supplied by the three.js library. The library provides a throughly tested and accurate raycasting solution. However, it is CPU bound and iterates every face of the mesh. This gives each raycasting operation a time complexity of O(n). If the 3D mesh is highly detailed, containing a large amount of polygons, the raycasting will take a long time to perform. After a careful assessment of potential solutions, the three.js plugin named three-bvh-mesh [55] was used to improve this. This plugin provides a BVH implementation in order to speed up the raycasting against three.js meshes. By using this plugin, the time complexity for a single raycasting operation decreases from O(n) to O(log n). Figure 3.3 shows an example visualization of a BVH tree applied to a 3D model, which is generated by the three-bvh-mesh plugin.

3.4.5 Color system

The color system of Voxelizer handles the storing and extraction of color from polygon meshes. This enables the engine to extract color information from the 3D mesh, and apply it to the voxels. The system mainly comprises of a ColorExtractor class and a TextureHandler class. The TextureHandler class stores a copy of all the texture maps associated with a 3D object. These are stored in a hashmap, using the texture maps UUID as key. Further, it is able to look up a UV coordinate of a texture map in its store, retrieving the corresponding color. See Section 2.11.1 for more details on texture maps and UV coordinates. The ColorExtractor class provides various methods for extracting colors from from an raycast intersect. It uses the TextureHandler as

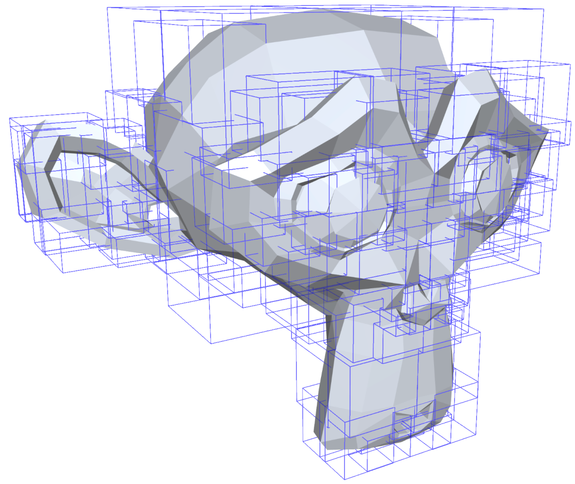


Figure 3.3: Visualization of BVH applied to 3D model.

internally texture storage, for fast lookup of texture colors.

During raycasting with the raycasting algorithm described in Section 3.4.3, each intersect produces an UV coordinate of the associated texture map, along with its UUID. This info is then fed to a ColorExtractor class, returning the appropriate RGB color of the intersect. This color data is then stored in a four-dimensional view of a large ndarray. For maximum efficiency and safety, actual datatype supplied to ndarray is an Uint8ClampedArray [56] (typed array).

3.4.6 Loading

The Voxelizer library/engine previously made use of a wrapper OBJ loader. This resulted in very limiting compatibility with other file formats. This support has been dropped in favor of the new ES6 JS loader modules introduced by three.js. three.js supports around 40 different file formats for loading 3D models. All three.js objects inherits from a base class named Object3D. This includes meshes. By ensuring compatibility with three.js meshes, any loader compatible with three.js can be used.

3.4.7 Exporting

As described in Section 3.4.1, the engine normally outputs ndarrays with color and voxel information, wrapped in a Volume class. However, the engine also comes with several exporter classes. This includes:

- **XML** - XML provides a versatile and flexible file format. The native JavaScript DOM parser is used for the actual parsing of the XML data. The outputted format of the XML document structure is described on the GitHub wiki page for the engine. PROVIDE LINK HERE !!!
- **BINVOX** - BINVOX [52] is one of the more popular voxel data file formats. BINVOX is the file format used by the binvox [1] voxelization software. A separate repository named binvox [53] has been created for handling BINVOX files. See Section 3.5.
- **3D array** - An array exporter is implemented for exporting the voxel data as a normal nested JavaScript array. If the export includes color data, this is exported as a 4D JavaScript array.

3.4.8 Testing

Several unit tests are created for testing the different parts of the voxelization system. This ensures correct operation of the voxelization process, and protect against introducing new bugs. Jest [57] has been chosen as the testing framework provider. Jest also provides coverage reports. These are very valuable in terms of analyzing what parts of the system is and is not tested.

3.4.9 Migration

The previous version of Voxelizer was version v0.1.3. Following Semantic Versioning [58], or SemVer, the old version of Voxelizer is defined as still in Beta. Introducing a new Major version of the library with breaking functionality is therefore no problem. Still, a very simple migration guide is provided on the Wiki of the Voxelizer engine repository on GitHub LINK HERE!!!!.

3.4.10 Debugging and Profiling

During development, the three-voxel-loader plugin was used to visualize the actual voxel outputs. This made it easy to visually inspect the results of the voxelization algorithm. A similar solution to the debugging setup used for the three-voxel-loader plugin was used. Likewise, this also resulted in an example usage of the engine, and is deployed to GitHub Pages. LINK TO VOXELIZER EXAMPLE PAGE.

For assessing the memory consumption and speed of the engine, the performance tool [59] in the Google Chrome Developer Tools is used. This helped removing some CPU-heavy bugs, memory issues and other performance bottlenecks. Figure 3.4 shows an image of the performance tool.

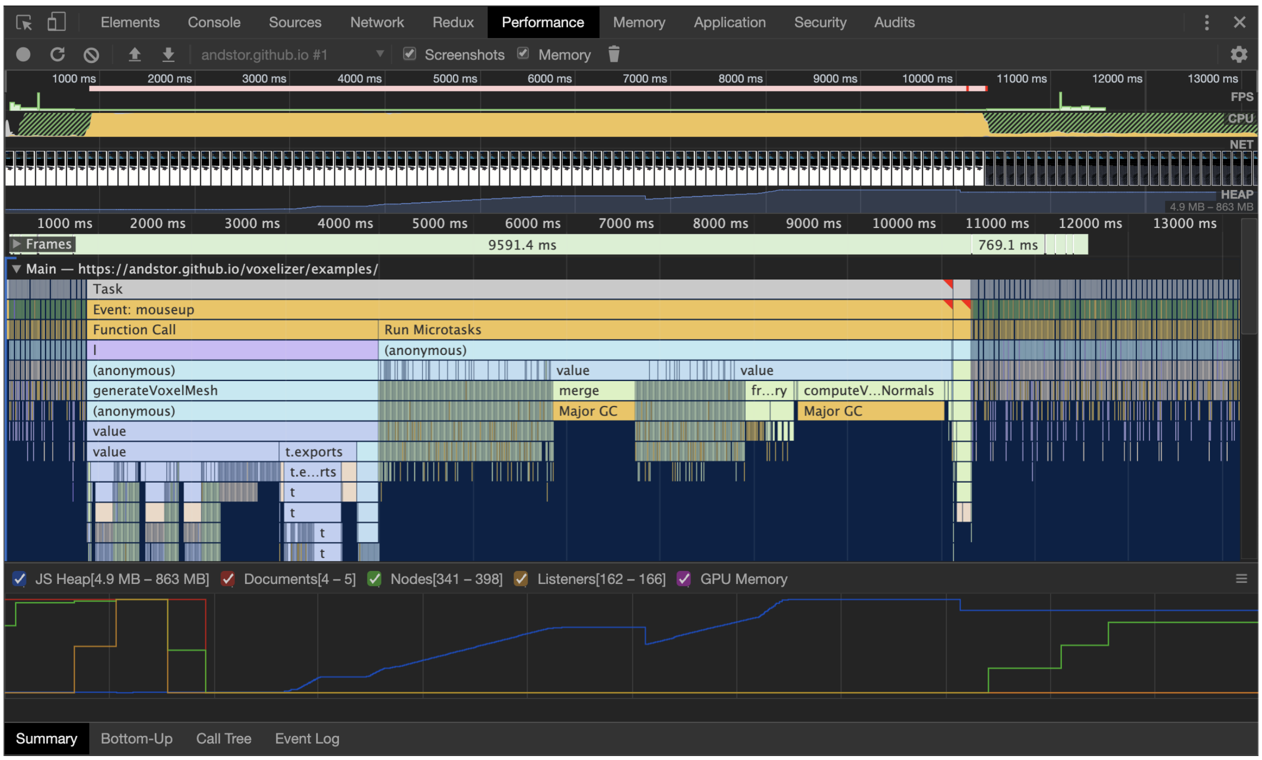


Figure 3.4: Screenshot of performance profiling with Google Chrome Developer Tools.

3.4.11 Building

The engine is bundled with Webpack. This gives great control over the building process. See Section 2.6.3 for more details on Webpack. Voxelizer also makes use of ES6 features, so all source files are also transpiled to ES5 with babel. The main output of the library is UMD, as described in Section 2.6.1. This makes the engine compatible with a range of other module systems. However, since Webpack does not support ES Modules right out of the box, a better alternative would be to use Rollup 2.6.3. Unfortunately, this was not possible due to a circular dependency in one of the dependencies of Voxelizer. Therefore, in order to use Voxelizer as a native ES Module, a project also needs to use a module loader like Webpack.

3.5 BINVOX

A separate open-source repo for building and parsing BINVOX file formats were created during refactoring of the Voxelizer engine and the three-voxel-loader plugin. It is named binvox and licensed under the MIT license. The BINVOX file format consists of a header in plain ASCII, followed by binary data. The binary data is compressed using run-length encoding. From the BINVOX specification: “The binary data consists of pairs of bytes. The first byte of each pair is the value byte and is either 0 or 1 (1 signifies the presence of a voxel). The second byte is the count byte and specifies how many times the preceding voxel value should be repeated (so obviously the minimum count is 1, and the maximum is 255).” (Min [52]).

The new binvox package provides tools to both parse and construct files according to the BINVOX file specification [52]. Parsing is done on the individual set of two bits, uncompressing the data and storing it in a JSON format. An example of the parsed JSON result can be seen in Program Code 3.1. Similarly, a user can supply the same JSON data structure for constructing a BINVOX file resource. Hence, parsing and building commute.

Program Code 3.1: Example BINVOX data in JSON format

```
{
    "dimension": {
         "depth": 32, "width": 32, "height": 32
    },
    "translate": {
        "depth": 11.81, "width": 21.39, "height": -1.69
    },
    "scale": 30.206,
    "voxels": [
        { "x": 0, "y": 2, "z": 3 },
        { "x": 0, "y": 3, "z": 3 },
        { "x": 0, "y": 4, "z": 3 },
        ...,
```

```
    ]
}
```

3.6 Voxelizer Desktop

Voxelizer Desktop is an desktop application for voxelizing 3D models. The next few sections goes through how the application is developed.

3.6.1 Design

Wireframes diagrams were created for planning the application's GUI. Figure 3.5 shows a wireframe diagram over the start screen. Here, a drag and drop interface is presented. The user should be able to just drag and drop the files he/she wants to voxelize into the application. Then, the application will need to load the 3D file(s). Figure 3.6 shows how this should be conveyed to the user by using some form of loading graphic. Finally, Figure 3.7 shows the main screen. This interface is divided in two parts. In the lower half of the application window, various controls are presented. This is further divided into a *Settings* section, and a *Exporting* section. The settings section provides controls for the actual voxelization settings. This includes settings for resolution, coloring and solid/shell. The exporting section provides a UI for selecting exporting type and saving it as a file. In the upper half, an interactive widow should display the voxelized result of the voxelization.

3.6.2 Implementation

The Voxelizer Desktop application is built with the Electron framework, described in Section 2.10.4. Alongside the bare Electron framework, the electron-builder [60] package is used. This package provides a complete solution for packaging and building a distribution ready Electron app for macOS, Windows and Linux. It also provides "auto update" out of the box. For creating the actual GUI, the React library is used. The create-react-app [61] is used for bootstrapping the react project.

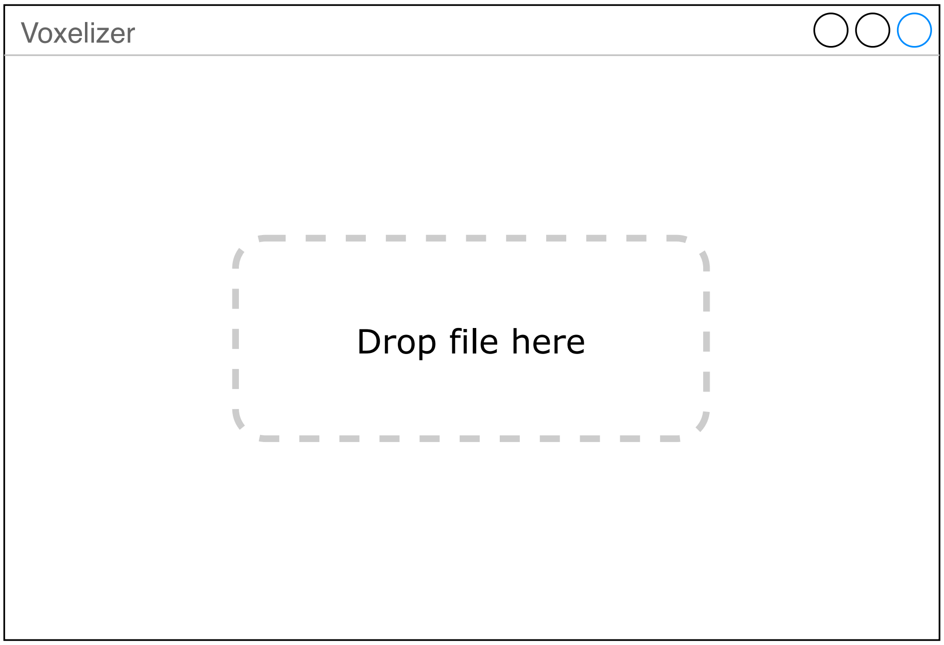


Figure 3.5: Wireframe diagram of drag and drop start screen.

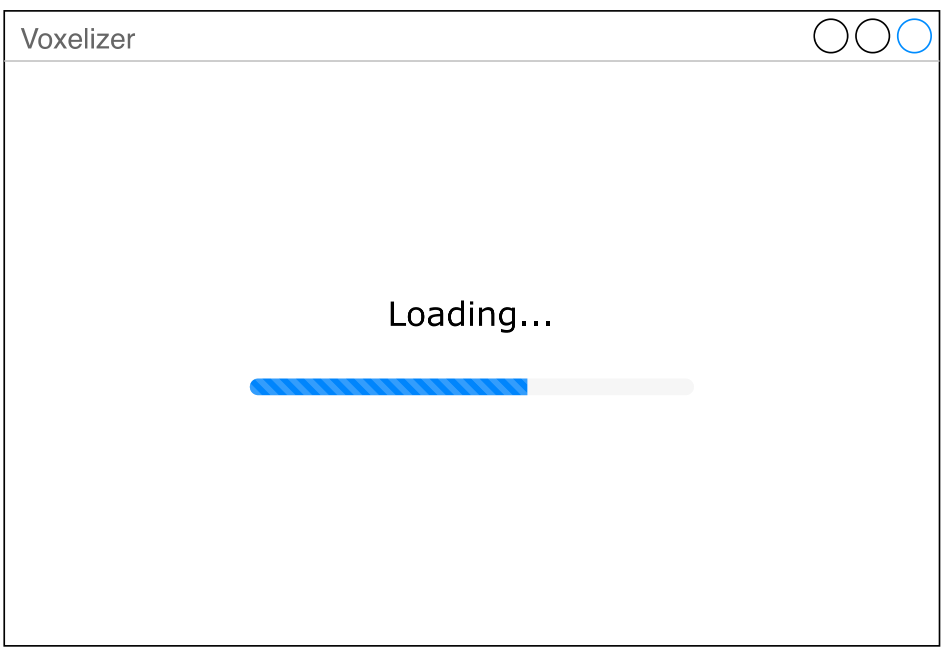


Figure 3.6: Wireframe diagram of loading screen.

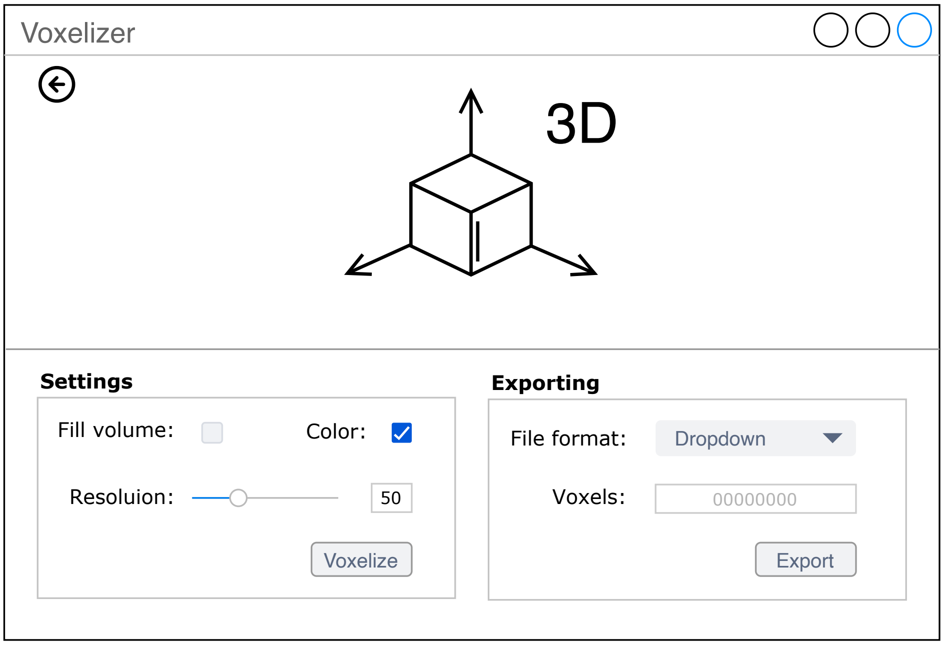


Figure 3.7: Wireframe diagram of the main screen.

For providing the voxelization, the Voxelizer engine is put to use. For visualization of the voxelized result, the three-voxel-loader plugin is used to generate a 3D mesh. This is then passed on to three.js.

In order to make the application user-friendly, a translation system is also implemented. This system is mainly backed by the react-intl package [62]. During application startup, the language setting (locale) on the users computer is read in the application's main process. This is then passed on to the instantiated renderer process through IPC. Based on this locale, the GUI is displayed in the appropriate language is presented.

3.7 JSDoc Action

In the next couple of subsections, the implementation and design decisions of the JSDoc Action will be presented.

3.7.1 Implementation

For creating a GitHub Action, several mandatory files and configurations has to be created. This documentation is available at GitHub. GitHub Actions provides two main types of action. One is JavaScript based. The other is based on Docker. They both have their advantages and disadvantages. A Docker container action provides an isolated environment, providing extremely flexible and stabile solutions. However, it is only possible to run on Linux. On the other hand, a JavaScript GitHub Action can be run directly on a runner machine. This makes it a lot faster than a Docker based Action. This is because of latency due to retrieve and build the container. A JavaScript action is also cross platform compatible, meaning it can run on both a Linux, Windows or MacOS operating system. In order to provide a fast and cross compatible solution, the JavaScript action type is chosen for the JSDoc Action.

The JSDoc Action is made up of mainly two parts. One is the template installation system, and the other is the actual execution of the JSDoc tool. For providing maximum flexibility, all functions of the JSDoc tool is made available as input configurations through the GitHub Actions Workflow API. The action is also able to use a JSDoc configuration file [63]. If a user provides a template to be installed, the action will first install this. This is done with the help of the node package manager (npm). The supplied template can be everything from a GitHub repository, to an npm package. The installed template files are then processed. The action does all of its IO operations asynchronously, ensuring fast execution speed of the action. When finished, a JSDoc CLI command is then formed based on the various user inputs and (if provided) config file. This command is then executed, effectively telling JSDoc to generate the documentation. The result is a user defined output folder with the generated API documentation.

3.7.2 Usage

For actually using the action in a Workflow, the action makes a couple of assumptions. Firstly, the actual source files to generated documentation from needs to be supplied. This is normally solved by using the Checkout Action [64] made by GitHub. Secondly, the JSDoc Action only generates an output directory with files. This means that nearly any deployment action can be used for upload the files the desired service, for example GitHub pages. The deployment action supplied as example in the README.md file of the repository is named GitHub Pages Action [65].

3.7.3 Feedback

The JSDoc Action generated quite a lot of feedback from eager users of the action. Several wanted to test the action, and multiple issues were filed in the issue tracker on GitHub [66]. See for example issue #20 [67]. Since maintenance is essential to the success of an open-source project, all feedback were responded to and handled accordingly. Alongside the development of the other main projects, many bug-fixes were made to the JSDoc Action. Eventually, all issues were resolved, resulting in several happy users.

3.8 Automation

Automation is an important part for ensuring both efficiency and security. The next subsections will contain a description of how the various open-source systems developed in connection with this thesis have been automated.

3.8.1 JavaScript package workflows

GitHub provides CI and CD as a part of its GitHub Actions system. This system has been put to use, creating several workflows in order to automate various tasks like testing, building, documentation generation and publishing. The workflows have been made so that they should work out of the box for similar projects. See Section 2.3.1 for a description of GitHub Actions. Do note that the default behavior of Workflows is to terminate if any errors are encountered.

Figure 3.8 shows a simplified diagram of the CI/CD automation pipelines created for the JavaScript packages which are to run on new contributions to the codebase. This system mainly consists of two workflows. One for building the package, and one for generating and publishing API documentation. A simple step by step walkthrough of the system is now presented.

1. **Pull request** - The pipelines are mainly triggered by a pull request. This starts up both a security analysis by LGTM and a build workflow.

2. **Security analysis** - As described in Section 2.10.6, LGTM provides a security analysis. If any vulnerabilities are found, the pull request fails.

3. **Build workflow** - The build workflow has four steps.
    (a) **Checkout** - First, the repository in which the automation is done on is cloned.
    (b) **Build** - Then, the JavaScript project is built.
    (c) **Test** - If the project provides tests, these are also run.
    (d) **Coverage** - Coverage reports are created with Jest.
4. **Coveralls** - The coverage report is then published to Coveralls.io. See Section 2.10.7 for details on Coveralls.

If the workflow finishes, the security analysis comes out clear, and the coverage percentage is not decreased, the pull request is approved for merging. When a user with the appropriate privileges approves the pull request, the code is merged into the base branch. If the base branch is the master branch, a second workflow is started. This is a workflow for generating/updating the API documentation.

1. **Checkout** - The source repo is cloned once again.
2. **JSDoc Action** - The JSDoc Action is then used for generating JSDoc API documentation.
3. **GitHub Pages** - The outputted documentation is then finally deployed to GitHub Pages with an action called [65].

In order to automate the release process of a JavaScript package, a third workflow is used. This is shown in Figure 3.9. It mainly involves packaging and publication of the software to the npm registry [68]. The workflow functions like this:

1. **Trigger release** - First, a user with the appropriate privileges needs to create a release manually on GitHub. This generates a git tag. Further, the release starts up the package workflow.
2. **Package workflow**
3. (a) **Checkout** - The source repo is cloned once again.
    (b) **Build** - The JavaScript project is then built.

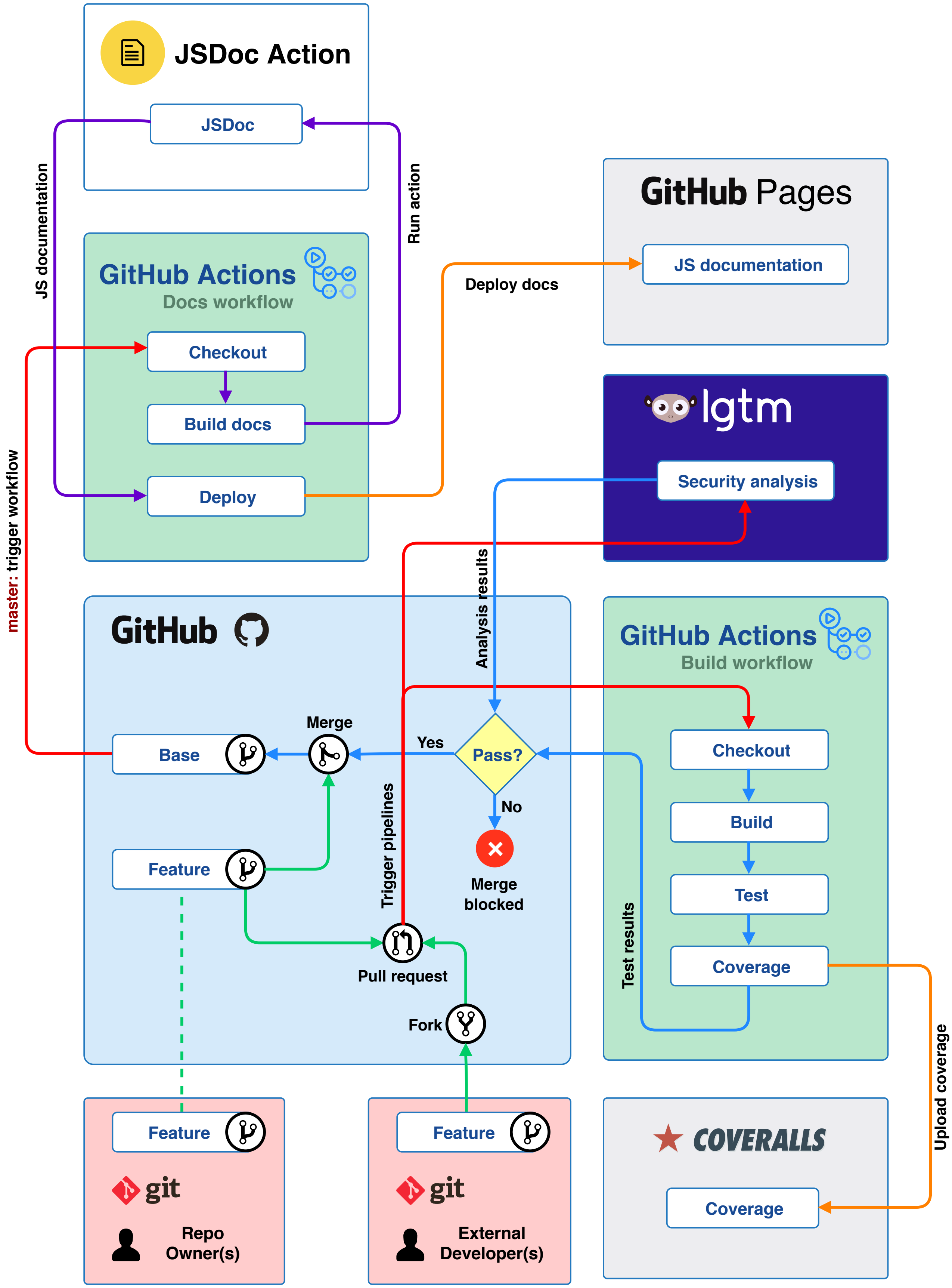


Figure 3.8: CI/CD pipelines

(c) **Test** - If the project provides tests, these are also run.

(d) **Package** - If all the above tests are completed without errors, the project is then packaged.

4. **Publish package** - If the package workflow completes, the new package is then published to the npm registry.

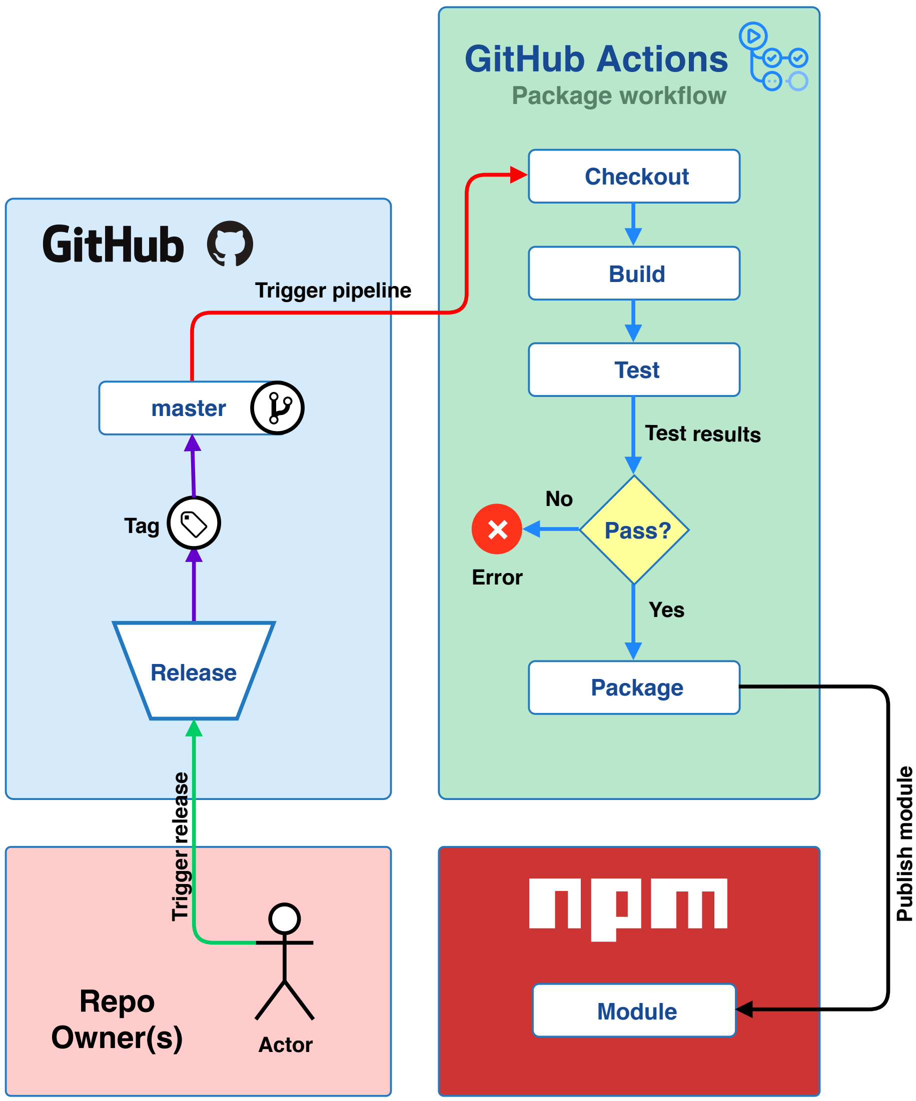


Figure 3.9: Automation of release publishing process.

3.8.2 GitHub Action version tagging

GitHub provides guidelines in terms of versioning and tagging of a GitHub Actions [69]. According to Github, releases should be following Semantic Versioning. When a new release is made, the major tag (v1, v2, etc.) should be moved to point on the Git ref of the current release. This process is tedious. A workflow has therefore been created for automating this release process. Figure 3.10 provides an illustration of the moving of Git tags. An action named actions-tagger [70] already provides support for this tagging scheme.

Following Auto major version tagging update

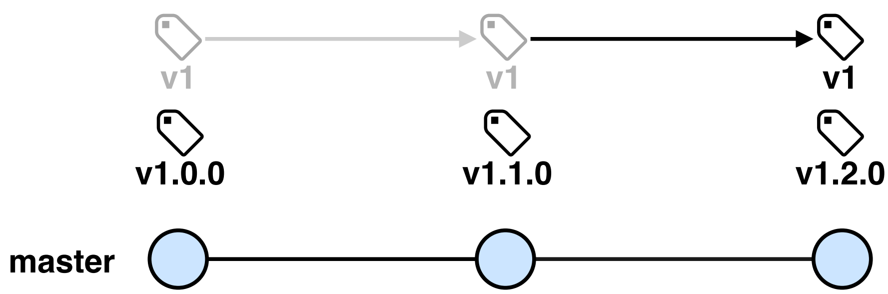


Figure 3.10: Automatic updating of major version tag.

3.9 file-existence-action

In order to be able to produce the general purpose workflows described in Section 3.8.1, it became apparent that i needed to be able to check if a file existed. Normally, a coverage script is defined in the projects package.json file. This runs all the tests and produces a coverage report file. However, if no such script is provided, the workflow will run the tests manually instead, and no file will be generated. If no such file is generated, this needs to be detected by the workflow, in order to avoid running the upload to Coveralls.io step. This resulted in a relatively small GitHub Action, named File Existence. The action is written in TypeScript, and based on a template provided by GitHub. The action is able to check one or more paths for the existence of a file. The boolean result is then available to the following steps in the workflow.

3.10 file-reader-action

Following the problems described in Section 3.9 above, another problem arises if the coverage script is defined, but no tests are created. If this is the case, an empty coverage file is created. Trying to upload this to Coveralls.io results in an error. The need for reading the contents of a file was therefore necessary, in order to check if the coverage file is empty. If it is empty, the Coveralls step should not be executed. The solution was to create a small GitHub Action, named File Reader. The action is written in TypeScript, and based on a template provided by GitHub. The action is able to read the contents of a file path supplied by the user. THe output is then available to the consequently workflow steps.

Chapter 4

Result

Brief assessment of the overall goals (or recap?) What to be discussed in this chapter... This section presents a walk through of the results...

4.1 three-voxel-loader

Include picture of voxelized chicken here

4.1.1 Level Of Detail

Include picture of torus with low LOD

4.1.2 Performance

Not optimized, a lot of unnecessary voxel geometry rendered when only outer mesh is visible.

4.1.3 Loading support

- XML - VOX - BINVOX -> Separate repo! - 3D array

4.1.4 Example

An example is available on GitHub pages (repo)

4.2 Voxelizer

General overview of the engine.... Completely redesigned.. Features: Extendible ...

4.2.1 Voxelization

side by side image of anvil and the anvil voxelized.

As you can see from the figure xxx, the system support coloring.....

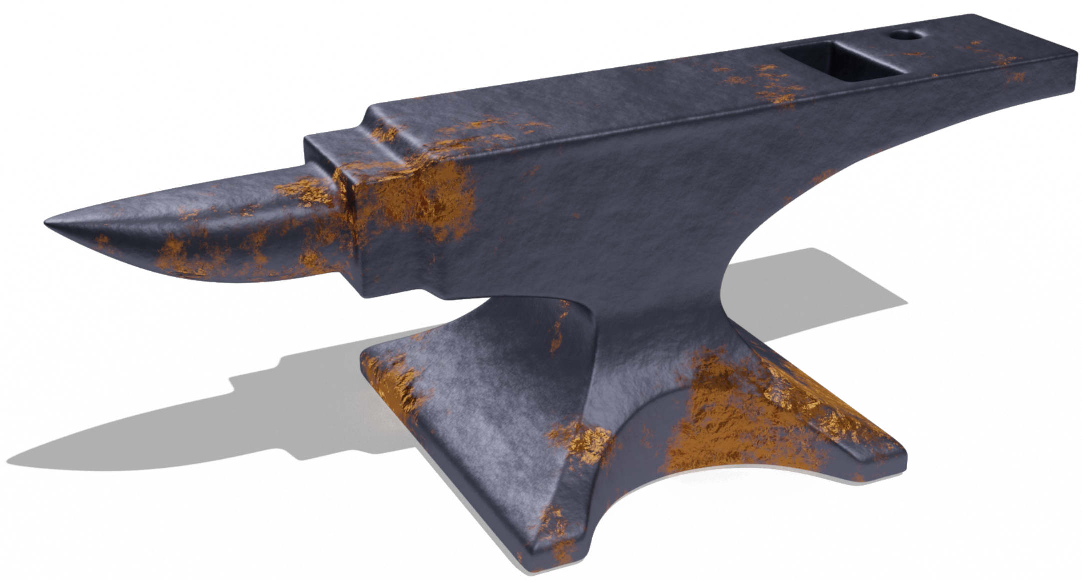


Figure 4.1: Render of anvil 3D model.

4.2.2 Visual inspection

No holes, accurate representation Compare to requirement specification.

4.2.3 Performance

Do some tests and present them in a table. Compare to requirement specification acceptance criteria.

4.2.4 Importing

This will be left up to the user. Supports all three.js meshes....

4.2.5 Exporting

Several formats... - XML - BINVOX - 3D array - ndarray

4.2.6 Usage example

Example of how to use the library with code!!!! HERE!!!!

```
// Import via ES6 modules
import * as THREE from 'three';
import {Sampler, XMLExporter} from 'voxelizer';

// Generate a yellow torus mesh to voxelize.
let geometry = new THREE.TorusGeometry( 10, 3, 16, 100 );
let material = new THREE.MeshBasicMaterial( { color: 0xffff00 } );
let torus = new THREE.Mesh( geometry, material );

// Setup Voxelizer.
let options = {
  fill: false,
  color: true
};
const sampler = new Sampler('raycast', options);

// Voxelize torus.
const resolution = 10;
let data = sampler.sample(torus, resolution);

// Export result to XML.
const exporter = new XMLExporter()
exporter.parse(data, function (xml) {
  console.log(xml)
});
```

4.3 Voxelizer Desktop

TODO Cross compatibility Speed Simple to use Secure

4.4 JSDoc Action

Easy automation of JSDoc generation and publication. Included in MAIN JSDOC REPO!!! 10.000 stars and 38.000 users!!!

Supports templates and all that JSDoc3 natively supports. Can use other actions to deploy docs to arbitrary service.

4.5 Supportive projets

4.5.1 BINVOX

JS module Binary format... Parses BINVOX files into JSON. Builds BINVOX from JSON. Works according to the official specification Cross platform support (Node.js and Browser), ES6 and UMD.

4.5.2 file-existence-action

Simple GitHub action to check for the existence of files.

4.5.3 file-reader-action

Simple GitHub action to read the contents of a file.

4.6 Automation

More or less fully automated all build and release/publishing steps.

4.6.1 JavaScript modules

- Building - Testing - coverage upload to coveralls - Security analysis with LGTM - JavaScript documentation Generation and deployment of the docs to GH pages. - Publishing module to NPM.

4.6.2 GitHub Actions

- automation of "node-modules" installation setup - automatic update of major version tags according to guidelines recommended by GitHub.

4.6.3 LaTeX automation?

Should i include this?

4.7 Open-source community

Several of the projects has already gained interest by the public. Promotion og JSDoc Action in the main repo with 38 thousand users.

4.7.1 Statistics

Number of stars, weekly downloads, etc

4.7.2 Feedback

Several filed issues (which has been resolved). Feedback from the users, etc... Feature requests...

4.8 Requirements specification

The different projects have addressed many of the user-stories defined in the backlog. This can bee seen in the backlog in Appendix C. 3-4 sentences....

Chapter 5

Discussion

5.1 Completness compared to requirements specification

The system is overall considered a sucess. All primary objectives are completed and

5.2 Test results

5.2.1 Result 1

The ...

5.3 Future work

5.3.1 three-voxel-loader

Performance

The three-voxel-loader plugin generates a cube buffergeometry for every voxel. Even for voxels that are not visible from outside the model. When loading a large and filled voxel model, this results in an enourmous number of faces being rendered, putting a heavy load on the hardware. A future improvement could be to only render a shell geometry based on the voxel "cubes". This would dramatically reduce the number of triangles needed to render the voxel model.

5.3.2 Voxelizer

Currently, raw usage of the engine in a browser, the program will run on the main thread. This will in turn freez the GUI. It should be possible to run the engine in a WebWorker, hence leveraging multithreading. However, future work could look into implementing support for webworkers directly into the Voxelizer engine. This way, the heavy voxelization calculations could be split up into chuncs and voxelized in parallel.

5.3.3 isosurface extraction

Isosurface extraction is the exact opposite process of the voxelization process. It tries to approximate a 3D mesh based on voxel data. Implementing this possibility into the projects could be of great value.

Chapter 6

Conclusions

The ....

6.1 Further work

In the .....

- Implement ....
- Expand .....
- Upgrade ......

Bibliography

[1] P. Min. (Jan. 2004 - 2020). "Binvox," [Online]. Available: http://www.patrickmin.com/binvox (visited on 05/06/2020).

[2] Scrum.org. (2020). "Scrum," [Online]. Available: http://www.scrum.org/ (visited on 05/06/2020).

[3] Atlassian. (2020). "What is kanban?" [Online]. Available: https://www.atlassian.com/agile/kanban (visited on 05/06/2020).

[4] Git. (). "About git," [Online]. Available: https://git-scm.com/about.

[5] V. Driessen. (Jan. 2010). "A successful git branching model," [Online]. Available: https://nvie.com/posts/a-successful-git-branching-model/ (visited on 04/29/2020).

[6] GitHub. (2020). "Github actions," [Online]. Available: https://github.com/features/actions.

[7] ——, (2020). "Workflow syntax for github actions," [Online]. Available: https://help.github.com/en/actions/reference/workflow-syntax-for-github-actions (visited on 05/07/2020).

[8] ——, (2020). "Github pages," [Online]. Available: https://pages.github.com (visited on 05/02/2020).

[9] M. D. Network. (Mar. 2019). "Prototype-based programming," [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Prototype-based_programming (visited on 05/04/2020).

[10] Netscape. (Dec. 1995). “Netscape and sun announce javascript,” [Online]. Available: https://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html (visited on 05/03/2020).

[11] E. International. (2020). “Ecma international,” [Online]. Available: http://www.ecma-international.org (visited on 05/03/2020).

[12] TIOBE Software. (Mar. 2020). “TIOBE index for may 2020,” [Online]. Available: https://www.tiobe.com/tiobe-index/ (visited on 05/03/2020).

[13] npm. (2020). “Npm,” [Online]. Available: https://www.npmjs.com (visited on 05/03/2020).

[14] RequireJS. (). “Requirejs,” [Online]. Available: https://requirejs.org (visited on 05/04/2020).

[15] UMD. (Oct. 2017). “UMD,” [Online]. Available: https://github.com/umdjs/umd (visited on 05/04/2020).

[16] Mozilla Developer Network. (Apr. 2020). “Browser support,” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules (visited on 05/04/2020).

[17] Babel. (2020). “Babel is a javascript compiler. babel is a javascript compiler. babel is a javascript compiler,” [Online]. Available: https://babeljs.io (visited on 05/04/2020).

[18] Webpack. (2020). “Webpack,” [Online]. Available: https://webpack.js.org (visited on 05/04/2020).

[19] Rollup. (2020). “Rollup,” [Online]. Available: https://rollupjs.org/guide/en/ (visited on 05/04/2020).

[20] Browserify. (2020). “Browserify,” [Online]. Available: http://browserify.org (visited on 05/04/2020).

[21] TypeScript. (2020). “Typescript,” [Online]. Available: https://www.typescriptlang.org (visited on 05/04/2020).

[22] Microsoft. (2020). “Microsoft,” [Online]. Available: https://www.microsoft.com (visited on 04/30/2020).

[23] Google. (May 2020). “Google closure compiler,” [Online]. Available: https://github.com/google/closure-compiler (visited on 05/04/2020).

[24] JSDoc. (May 2020). "Jsdoc," [Online]. Available: https://github.com/jsdoc/jsdoc (visited on 05/04/2020).

[25] GitHub. (May 2020). "Dependency graph," [Online]. Available: https://github.com/jsdoc/jsdoc/network/dependencies (visited on 05/04/2020).

[26] ——, (May 2020). "Stargazers," [Online]. Available: https://github.com/jsdoc/jsdoc/stargazers (visited on 05/04/2020).

[27] K. Group. (). "Webgl overview," [Online]. Available: https://www.khronos.org/webgl/.

[28] ——, (2020). "Opengl es," [Online]. Available: https://www.khronos.org/opengles/ (visited on 05/06/2020).

[29] three.js. (). "Three.js," [Online]. Available: https://threejs.org.

[30] Mozilla Developer Network. (Mar. 2020). "Javascript typed arrays," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays (visited on 05/04/2020).

[31] GitHub. (2020). "Electron," [Online]. Available: https://www.electronjs.org (visited on 05/02/2020).

[32] ——, (2020). "About github," [Online]. Available: https://github.com/about (visited on 01/25/2020).

[33] Microsoft. (2020). "Visual studio code," [Online]. Available: https://code.visualstudio.com (visited on 05/02/2020).

[34] Facebook. (2020). "Facebook messenger," [Online]. Available: https://www.messenger.com/desktop (visited on 05/02/2020).

[35] Microsoft. (2020). "Microsoft teams," [Online]. Available: https://www.messenger.com/desktop (visited on 05/02/2020).

[36] Google. (). "Chromium," [Online]. Available: https://www.chromium.org/Home (visited on 05/11/2020).

[37] Facebook. (2020). "React," [Online]. Available: https://reactjs.org (visited on 05/02/2020).

[38] ——, (). "Facebook," [Online]. Available: https://about.fb.com (visited on 05/02/2020).

[39] Semmle. (2020). “Lgtm,” [Online]. Available: https://lgtm.com (visited on 05/04/2020).

[40] ——, (2020). “Security at semmle,” [Online]. Available: https://semmle.com/security (visited on 05/04/2020).

[41] Coveralls. (). “Coveralls,” [Online]. Available: https://coveralls.io (visited on 05/02/2020).

[42] E. E. Catmull, “A subdivision algorithm for computer display of curved surfaces,” Ph.D. dissertation, University of Utah, 1974.

[43] B. Foundation. (). “About blender,” [Online]. Available: https://www.blender.org/about.

[44] ——, (2020). “Render baking,” [Online]. Available: https://docs.blender.org/manual/en/latest/render/cycles/baking.html (visited on 05/03/2020).

[45] S. D. Roth, “Ray casting for modeling solids,” *Computer Graphics and Image Processing*, vol. 18, no. 2, pp. 109–144, Feb. 1982.

[46] D. MacDonald, “Space subdivision algorithms for ray tracing,” M.S. thesis, University of Waterloo, waterloo, Ontario, 1988.

[47] Wikipedia. (2020). “Voxel,” [Online]. Available: https://en.wikipedia.org/wiki/Voxel (visited on 05/04/2020).

[48] A. Storhaug. (Apr. 2019). “Voxelizer v0.1.3,” [Online]. Available: https://github.com/andstor/voxelizer/tree/v0.1.3 (visited on 05/04/2020).

[49] Travis. (2020). “Travis ci,” [Online]. Available: https://travis-ci.org (visited on 05/10/2020).

[50] R. van Rüschen. (Jan. 2020). “Sparse octree,” [Online]. Available: https://github.com/vanruesc/sparse-octree (visited on 05/10/2020).

[51] D. Klein. (Aug. 2018). “Format-vox,” [Online]. Available: https://github.com/sh-dave/haxe-format-vox (visited on 05/08/2020).

[52] P. Min. (). “Binvox voxel file format specification,” [Online]. Available: https://www.patrickmin.com/binvox/binvox.html.

[53] A. Storhaug. (May 2020). "Binvox," [Online]. Available: https://github.com/andstor/binvox (visited on 05/08/2020).

[54] G. D. A. Team. (2020). "Dat.gui," [Online]. Available: https://github.com/dataarts/dat.gui (visited on 05/08/2020).

[55] G. Johnson. (Mar. 2020). "Three-mesh-bvh," [Online]. Available: https://github.com/gkjohnson/three-mesh-bvh (visited on 05/09/2020).

[56] Mozilla Developer Network. (2020). "Uint8clampedarray," [Online]. Available: Uint8ClampedArray (visited on 05/10/2020).

[57] Facebook. (). "Jest," [Online]. Available: https://jestjs.io (visited on 05/10/2020).

[58] T. Preston-Werner. (2020). "Semantic Versioning," [Online]. Available: https://semver.org (visited on 05/10/2020).

[59] K. Basques. (2019). "Get started with analyzing runtime performance," [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/evaluate-performance (visited on 05/10/2020).

[60] electron-builder. (2020). "Electron-builder," [Online]. Available: https://www.electron.build (visited on 05/11/2020).

[61] Facebook. (2020). "Create react app," [Online]. Available: https://github.com/facebookincubator/create-react-app (visited on 05/11/2020).

[62] formatjs. (2020). "React-intl," [Online]. Available: https://www.npmjs.com/package/react-intl (visited on 05/11/2020).

[63] J. 3. (2011-2017). "Configuring jsdoc with a configuration file," [Online]. Available: https://jsdoc.app/about-configuring-jsdoc.html (visited on 05/10/2020).

[64] GitHub. (May 2020). "Checkout v2," [Online]. Available: https://github.com/actions/checkout (visited on 05/10/2020).

[65] S. Ueda. (2020). "Github pages action," [Online]. Available: https://github.com/marketplace/actions/github-pages-action#table-of-contents (visited on 05/10/2020).

[66] A. Storhaug. (2020). "Jsdoc action issues," [Online]. Available: https://github.com/andstor/jsdoc-action/issues?q=is%3Aissue+ (visited on 05/10/2020).

[67] Androz2091. (Apr. 2020). “How to use a github repo as template?” [Online]. Available: https://github.com/andstor/jsdoc-action/issues/20 (visited on 05/10/2020).

[68] npm. (2020). “Npm registry,” [Online]. Available: https://www.npmjs.com (visited on 05/10/2020).

[69] GitHub. (2020). “Versioning your action,” [Online]. Available: https://help.github.com/en/actions/building-actions/about-actions#versioning-your-action (visited on 05/10/2020).

[70] smac89. (May 2020). “Actions tagger,” [Online]. Available: https://github.com/marketplace/actions/actions-tagger (visited on 05/10/2020).

Appendices

Appendix A

Preliminary report

Preliminary report
for bachelor's thesis

Kunnskap for en bedre verden

Title			
Open source JavaScript voxelization library, with complementary applications			

Candidate				
André Storhaug				
Date	Subject code	Subject		Document access
31.01.2020	IE303612	Bachelor's thesis		- Open
Study		Pages/Attachments		Bibl. nr.
Computer engineering		21/0		Not used

Supervisor(s)
Primary: Ricardo Da Silva Torres
Secondary: Saleh Abdel-Afou Alaliyat

Summary

This preliminary report concerns the plans and preparations for a bachelor's thesis at the Norwegian University of Science and Technology (NTNU).

The purpose of the thesis is to improve and further develop the already existing open source project named Voxelizer. Voxelizer is a JavaScript library for converting 3D models into a volumetric representation, a process known as voxelization. The library needs to be refined, professionalized and extended.

To ease the use of the library, a cross platform desktop application and a command line interface will be developed. In order to make the Voxelizer library and the complementary software easy to maintain, the projects will have a focus on automation. Especially, a GitHub action will be developed for automating the generation of JavaScript documentation.

Postal address	Visitation address	Phone	Fax	Bank account
NTNU i Ålesund	Larsgårdsvegen 2	70 16 12 00	70 16 13 00	7694 05 00636
N-6025 Ålesund	**Internet**	**E-mail**		**Foretaksregisteret**
Norway	www.ntnu.no	postmottak@ntnu.no		NO 947 767 880

This assignment is an exam submission done by a student at NTNU in Ålesund.

Contents

Chapter 1

Terminology

Concepts

Voxel Three-dimensional analogue of a pixel, representing a value on a regular grid in three-dimensional space.

Voxelization The process of converting a 3D model into voxels.

Library A collection of data and programming code that are used to develop software.

Cross-platform Computer software that can be run on multiple computing platforms.

Abbreviations

MDN Mozilla Developer Network

API Application Programming Interface

UML Unified Modeling Language

GUI Graphical User Interface

CLI Command Line Interface

Chapter 2

Introduction

This project aims to improving an already existing open source [27] library named Voxelizer [20]. It is a cross-platform library for conducting voxelization of 3D models, and is written in JavaScript.

The background for its creation was an assignment in a simulation course. The objective was to simulate diffusion using a cellular automaton. I wanted to do the simulation in the shape of a 3D object. Hence, I needed some way of constructing a volume representation out of a 3D model. Further, I also wanted to make the simulation with web technologies by making use of Three.js [21], an abstraction layer over WebGL [11].

To my surprise, I couldn't find any simple open source JavaScript solution for this. I therefore decided to make a solution myself. The result was the open source library "Voxelizer". However, due to time constraints, the current state of the library can only be considered a crude prototype. It has several issues, lacks important features and needs to be professionalized.

Alongside the library, a desktop program and a CLI interface will be developed. These will be making use of the Voxelizer library, and will greatly simplify the use of it.

Chapter 3

Project organization

3.1 Project group

Table 3.1: Students in the group

Name of student
André Storhaug

3.2 Steering group

The steering group will consist of Ricardo Da Silva Torres at NTNU in Ålesund, along with Saleh Abdel-Afou Alaliya at NTNU in Ålesund. Torres will act as supervisor and Alaliyat will be the co-supervisor.

Chapter 4

Attitudes

As a computer engineer, one is expected to behave responsible and professional. One should be curious of new technology and strive to provide the best solutions possible. Further, one should take proud in ones own work and feel a certain responsibility of the work that is done.

In a world that is becoming increasingly smaller, good collaboration is essential. In the role as a computer engineer, it is expected that one will come into contact and collaborate with persons from many different disciplines. It is therefore vital to have open mindsets and welcome new ideas. Discussions are to be expected. However, disagreements should be kept factual and handled respectfully.

The development of software is often a large part of the job as a computer engineer. Software has potential to affect human lives. Either directly or indirectly. The creation of software should therefore be rooted in strong ethics and respect for the users privacy. With this in mind, open source is a great way of achieving both transparency and openness. Today, everything revolves around profit. Companies are doing everything from charging huge amounts for proprietary software, to profiting on your personal information. However, open source has become a popular platform in which people can collaborate on software projects. By join forces and helping one another, one can achieve truly great things.

Chapter 5

Project description

5.1 Thesis problem - goals - purpose

5.1.1 Thesis problem

There exists an open source JavaScript library for conducting voxelization of 3D models. This library is called "Voxelizer". However, the library faces several issues and is lacking important features. It needs to be professionalized and made easy to both use and maintain.

5.1.2 Goals

This project has two main goals. The first goal is to improve and extend the open source Voxelizer library in such a way that it fulfills the requirements specified in the next section. The second goal is to develop a cross platform desktop application and a CLI for easy voxelization of 3D models, based the Voxelizer library.

In order to ensure maintainability of the various software projects, automation is critical. Therefore, a common subgoal will be to develop a GitHub action in order to automate documentation generation.

5.1.3 Purpose

The purpose of this project is to make it easy to conduct high quality voxelization of 3D models.

5.2 Requirements specification

The scope of this project is defined and limited by the requirements specification defined in following sections. In addition to this specification below, a backlog with user stories shall be created.

5.2.1 Voxelizer

Algorithms

The voxelisation algorithm should provide an accurate render of the original 3D model (polygon mesh [25]). The result should be geometrically representative witout distortions. No holes should be presen, unless dictated so by the given 3D model shape. Internal cavities and structures needs to be accuratly preserved. Lastly, there should be an absolute minimum of artifacts.

It should be possible to do two types of voxelization. One that is a shell voxelization, and another that is a filled volume version. The shell-type algorithm should only capture the surface of the 3D model. The filled-type algorithm needs to capture a complete volume representation of the 3D model.

One should be able to set the wanted resolution of the voxelization.

Input

The library should support a large variety of different input types. Both in terms of various file types and data structures. Support for popular file formats such as OBJ, STL and gLTF should be implemented.

Output

A diverse mixture of output types have to be supported. This includes relevant file formats and data structures. Some file formats for saving voxel data are VOX by MagicaVoxel [6], XML, BINVOX [14] and minecraft SCHEMATIC format. Relevant data structure exports include 3D arrays and octrees.

It should also be possible to export the voxelized result as normal 3D models. This could be file formats such as OBJ, STL and gLTF. Each voxel in the model should be represented as a cube.

Lastly, one could also support image export for each layer of the voxelized result. File format could for example be JPEG or PNG.

Coloring

The texture of a 3D model should carry over to surface voxels. This should be in the form of the most representative color.

Optimization

tree.js raycasting should be optimized. three.js raycasting is CPU based. It itterates each face in a 3D model, checking if the ray intersects a face or not. However, one can speed up the raycasting by employing a spatial index, for example with the help of an octree [24] or aabb tree .

5.2.2 three.js voxel loader

The three.js voxel loader module needs to be able to load voxel data into a three.js mesh [23]. The module should manage to load the voxel file formats and data structures that the voxelizer library supports exporting. This is voxel data stored in the form of a 3D array or an octree, or in a file format like VOX, XML BINVOX or SCHEMATIC.

It should be possible to customize the appearance of the loaded voxels. Both in terms of size, material and/or color.

5.2.3 Voxelizer Desktop

The Voxelizer Desktop shall be a corss-plattform [26] desktop application. It should work on both MacOS, Windows and Linux. The application should be able to voxelize a 3D models with the use of the Voxelizer library [20]. Also, it should automatically update itself when a new release of the application is published.

The application should provide intuitive GUI. It should be possible to view both the original

3D model and the voxelized result. Also, it should be possible to generate a 2D view of the cross-sections of the voxelized model.

5.2.4 Voxelizer CLI

The Voxelizer CLI should be a corss-plattform [26] CLI application. It should function on both MacOS, Windows and Linux. The application should be able to voxelize a 3D models with the use of the Voxelizer library [20].

5.2.5 JSDoc Action

The JSDoc GitHub Action should be an installable GitHub Action [9], available from the GitHub marketplace [2]. It should automate the process of generating JavaScript documentation with the help of JSDoc [12].

5.2.6 Automation

Automation should be used to ease maintenance of the various software projects. Firstly, JavaScript projects needs to have the documentation automatically generated with JSDoc [12]. Secondly, the process of publishing new versions should be automated to the greatest extent.

5.3 Methodology

The method that will be utilize in this project is the agile methodology Scrum [19]. Scrum is a very popular working methodology in the software development business. It uses an iterative and incremental approach, where each sprint gives an opportunity to improve the development process. Scrum organizes the work in sprints. This is a predefined period of time that is devoted to a set of very defined goals. The tasks to be done are often defined in a product backlog [18].

Scrum is mainly intended for teams. However, even though this is a one man project. The Scrum methodology will serve as a project framework for keeping up with progress, in addition to being able to adapt the project pace to the available working capacity. By breaking down the tasks to be done in sprints, this will help with organizing the work and steering the project in the right direction, allowing adjustmens along the way.

For this project, each sprint will be two weeks long. After each spring, a review of the completed sprint will be made. This will be an opportunity to reflect on the process, and see which goals were completed and which wasn't. Further, this review will be highly valuable for determining if adjustments should be made for the next sprint.

Scrum also seems to be a good fit because there will be a meeting with the supervisor every two weeks. By organizing the tasks to be done in two-week sprints, this will make the meetings with the supervisor more effective and relevant. New functionallity can be discussed and reviewed, in addition to planning ahead for the next two weeks.

5.4 Information gathering

The main source of information will come from various web resources. Everything from articles to code documentation will be needed for this project. The MDN Web Docs [15] will be an important source for JavaScript documentation. For Node.js related work, the Node.js API Docs [16] will be put to good use. Also, documentation from the third party library Three.js [22] will be essential. Further, Stack Overflow [17] will be a highly valued source of information due to its wast amount of questions and answers in a lot of topics.

5.5 Risk analysis

A qualitative approach will be used for assessing the risk of this project. A risk can be described as the likelihood of an event times the impact. A **MEDIUM** risk level will be accepted. The table 5.1 will be used in order to define the various risk levels.

Table 5.1: Risk level matrix.

		IMPACT			
		LOW	**MEDIUM**	**HIGH**	**VERY HIGH**
LIKELIHOOD	**VERY HIGH**	MEDIUM	HIGH	VERY HIGH	VERY HIGH
	HIGH	MEDIUM	HIGH	HIGH	VERY HIGH
	MEDIUM	LOW	MEDIUM	HIGH	HIGH
	LOW	LOW	LOW	MEDIUM	MEDIUM

In table 5.2 below, a risk assessment and risk control is conducted. The letter “L" stands for “Likelihood”, “I” for “Impact” and "R" for "Risk".

Table 5.2: Risk assessment table.

						Residual risk		
ID	**Description**	**L**	**I**	**R**	**Risk control**	**L**	**I**	**R**
R1	Services like GitHub, Jira and Confluence may go down, making various resources unavailable.	L	VH	H	Perform regular backups of important data.	L	M	M
R2	Sickness, resulting in inability to work.	M	M	M	Practicing good hygiene.	L	M	M
R3	Damaged equipment used for development.	L	VH	M	Exercise caution when handling important equipment.	L	H	M
R4	Lost or corrupt files due to system crash or failure.	M	VH	H	Perform regular backups of important data.	M	L	L
R5	Incompatibilities between technologies.	M	M	M	Properly assess the technology and plan ahead before starting development.	L	M	L
R6	Security vulnerability in package dependency.	VH	H	VH	Automatic package auditing and fixing provided by GitHub [10].	L	H	M

VH: VERY HIGH risk
H: HIGH risk
M: MEDIUM risk
L: LOW risk

The risk assessment done in table 5.2 shows that with the appropriate counter measures, all risks are reduced to a MEDIUM level. This is an acceptable level.

5.6 Primary activities in further work

Table 5.3: Main activities.

Nr	Main activity	Time/scope
A1	**Writing**	**18 weeks**
A11	Preliminary report	3 weeks
A12	Bachelor's thesis	15 weeks
A2	**voxelizer**	**7 weeks**
A21	Core improvements	1 week
A22	Algorithm improvements	2 weeks
A23	Texture support	1 week
A24	Extending 3D model file loading	1 week
A25	Extending data exporting	1 week
A26	Write tests	3 days
A27	Optimization	2 days
A3	**three-voxel-loader**	**2 weeks**
A4	**voxelizer-desktop**	**3 weeks**
A41	Core	1 week
A42	GUI	2 weeks
A5	**voxelizer-cli**	**1 week**
A6	**jsdoc-action**	**1 week**
A7	**Automation**	**1 week**

5.7 Progress plan

5.7.1 Master plan

Following is a gantt diagram 5.1 for the planned time sceduling. This includes all activities listed in section 5.6. These activities primarly include writing and software development. Activity A1 is concerned about writing the preliminary report and the thesis. Activity A2, A3, A4, A5 and A6 is concerned with the development of various software systems, where each activity is a confined project. A7 is concerned with automation of various tasks in many of the software projects.

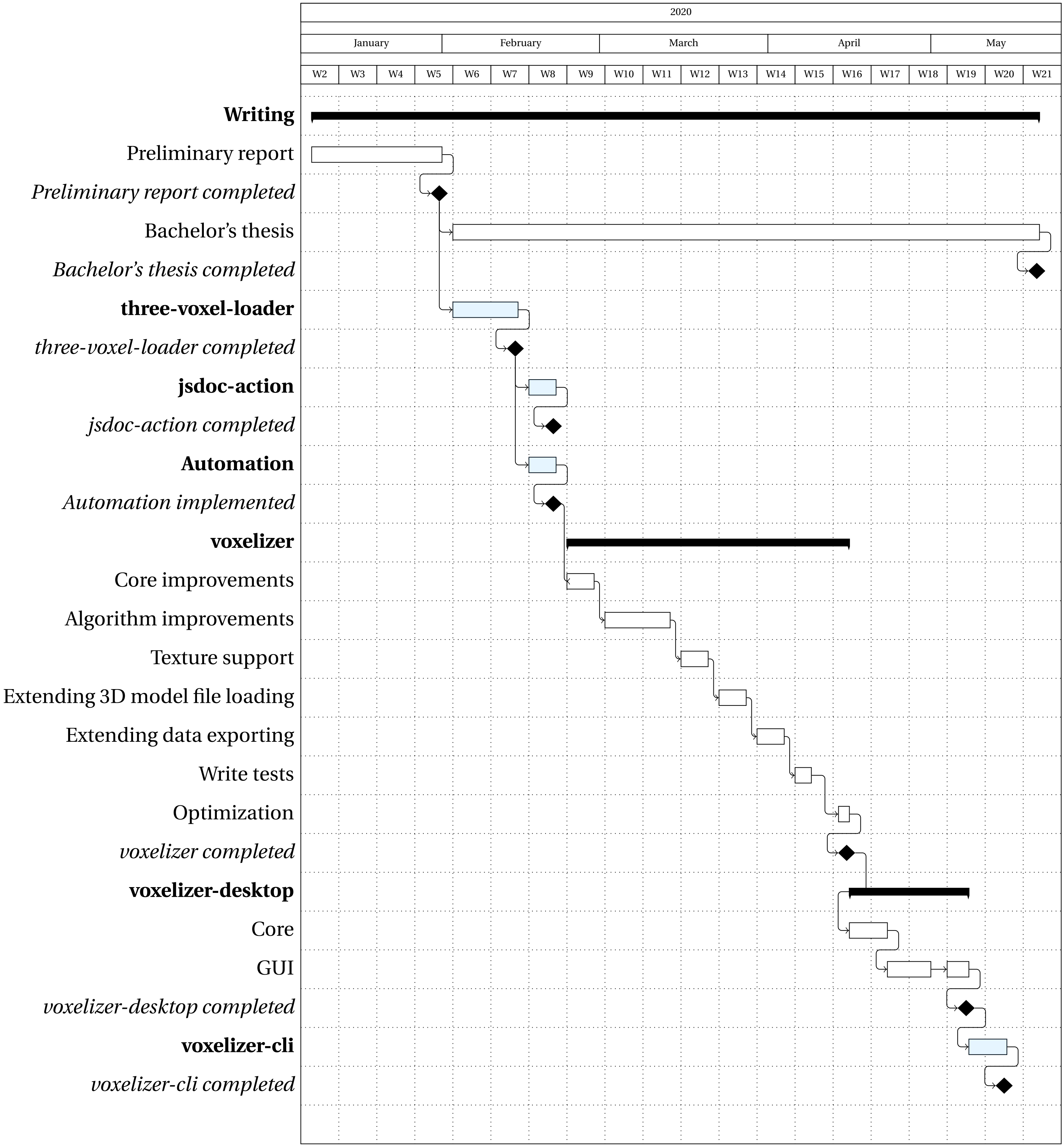


Figure 5.1: Gantt diagram of progress plan.

5.7.2 Project control assets

In order to keep the project on track, Jira [4] will be used. Jira is a project management tool developed by Atlassian, supporting a vast number of features such as issue tracking and project management. The main reason for choosing Jira over for example GitHub's solutions, is that Jira supports the agile methodology Scrum. For managing documents, minutes of meetings, UML diagrams, etc., Confluence [3] will be used.

Since this project revolves around open source projects, Jira and Confluence will only be used for internal related work. For public usage, the GitHub issue tracker and wiki will be used. Any issues, bugs or documentation of public interest shall be be placed on GitHub, instead of Jira and Confluence.

5.7.3 Development assets

For developing the various systems, the development tools listed in table 5.4 will be used.

Table 5.4: Development tools.

Development tool	**Description**
Visual Studio Code [13]	Editor for writing and debugging code.
Blender [7]	3D modeling software.
Git [8]	Version control.
GitHub [1]	Hosting of git repositories.
SourceTree [5]	Git desktop client.

5.7.4 Internal control and evaluation

At the end of each sprint, a review of the completed sprint will be conducted. A burndown chart will be generated for each sprint. This will help identifying if adjustments to the plan is nesseseary.

The requirements specification will serve as the primary criteria in order to decide wether a goal is completed or not. If a system is implemented but contains minor bugs, it will still be considered complete.

Chapter 6

Documentation

6.1 Reports and technical documents

Firstly, a progress report shall be created for every two weeks. Secondly, documentation for the various systems will be produced. In order to make a successful software library, good documentation is imperative. A lot of this documentation will mainly be automatically generated by JSDoc. The automation will be integrated in the workflow of a new version release of a system (GitHub repository). This ensures the validity of the documentation, as well as ensures future maintenance. This integration will be provided by a GitHub action, also to be a part of this project. The generated documentation will be publicly available, hosted at GitHub Pages. Lastly, various UML diagrams will be created. These will serve as illustration for the relationship between the various systems.

As mentioned in section 5.7.2, internal documentation will be kept private in Jira and Confluence. Documents of public interest will be placed publicaly available on GitHub.

Chapter 7

Planned meetings and reports

7.1 Meetings

A meeting with the advisor will be held every two weeks. These meetings will be used for reporting on the current progress. The meetings are an opportunity of gathering constructive feedback from the supervisor. Further, they will serve as documentation for working both professionally and responsible. A minutes of a meeting report will be written for every meeting.

7.2 Progress reports

Progress reports will be developed up-front of each meeting. These will describe what activities were planned, and what activities were actually seen through. If any deviations from the plan occurred during the period, these should also be included in the progress report. Further, the activities planned for the next period should also be outlined. The report will be sent to the supervisor at least a day before the meetings. This will form the basis for the matters to be discussed at the meetings.

Chapter 8

Planned deviation management

In the event of deviations from the current plans, both in terms of progress or content, several measures needs to be taken. If the deviations from the plan are of greater significance, the supervisor should be alerted. If the deviation is of lesser importance, it should be discussed with the supervisor at the regular meeting. One should then consider to change the planned approach.

Many of the planned systems builds upon one another. Therefore, if a task shows to be harder and more time consuming than first anticipated, it should consume time from tasks of lower priority. However, if a task exceeds its planned time scedule because of minor bugs, then these bugs should be properly documented and the task considered finished. These bugs should then be revisited at a later stage if there is time to spare. Since the systems are open source projects, these bugs might also be resolved by volunteers after this project is finished.

Bibliography

[1] About github, . URL https://github.com/about.

[2] Extend github, . URL https://github.com/marketplace.

[3] Atlassian. Confluence, . URL https://www.atlassian.com/software/confluence.

[4] Atlassian. Jira, . URL https://www.atlassian.com/software/jira.

[5] Atlassian. Sourcetree, . URL https://www.sourcetreeapp.com.

[6] ephtracy. Magicavoxel. URL https://ephtracy.github.io/index.html?page=mv_main.

[7] Blender Foundation. About blender. URL https://www.blender.org/about.

[8] Git. About git. URL https://git-scm.com/about.

[9] GitHub. Github actions, . URL https://github.com/features/actions.

[10] GitHub. Managing vulnerabilities in your project's dependencies, . URL https://help.github.com/en/github/managing-security-vulnerabilities/about-security-alerts-for-vulnerable-dependencies#alerts-and-automated-security-updates-for-vulnerable-dependencies.

[11] Khronos Group. Webgl overview. URL https://www.khronos.org/webgl/.

[12] JSDoc. Jsdoc. URL https://github.com/jsdoc/jsdoc.

[13] Microsoft. Visual studio code. URL https://code.visualstudio.com.

[14] Patrick Min. Binvox voxel file format specification. URL https://www.patrickmin.com/binvox/binvox.html.

[15] Mozilla. Mdn web docs. URL https://developer.mozilla.org/en-US/.

[16] Node.js. Node.js documentation. URL https://nodejs.org/api/.

[17] Stack Overflow. Stack overflow. URL https://stackoverflow.com.

[18] Scrum.org. What is a product backlog?, . URL https://www.scrum.org/resources/what-is-a-product-backlog.

[19] Scrum.org. Scrum, . URL http://www.scrum.org/.

[20] André Storhaug. Voxelizer. URL https://github.com/andstor/voxelizer.

[21] three.js. three.js, . URL https://threejs.org.

[22] three.js. three.js docs, . URL http://threejs.org/docs/.

[23] three.js. Mesh, . URL https://threejs.org/docs/#api/en/objects/Mesh.

[24] Wikipedia. Octree, . URL https://en.wikipedia.org/wiki/Octree.

[25] Wikipedia. Polygon mesh, . URL https://en.wikipedia.org/wiki/Polygon_mesh.

[26] Wikipedia. Cross-platform software, 1 2020. URL https://en.wikipedia.org/wiki/Cross-platform_software.

[27] Wikipedia. Open source, 1 2020. URL https://en.wikipedia.org/wiki/Open_source.

Appendix B

Progress reports

asd

Appendix C

Requirements specification

Requirements Specification (Jira)

Author: André Storhaug

Revision: 07/May/20

Table of Contents

3D Testing Model

DONE

A 3D model should be created to be a able to test the system.

It needs a relatively high level of complexity.

It should be textured in a way to showcase the coloring functionality of the Voxelizer system.

Resolution

DONE

As a user, I want to be able to set the wanted resolution of the voxelized output.

Relates	
relates to	Shell voxelization
relates to	Solid voxelization

Restructure code base

DONE

As a developer, I need the core codebase to have a good structure and be easy to maintain, so that other functionality that builds upon the core is easy to develop with high quality.

Implement a spatial index

DONE

three.js raycasting optimization

DONE

Surface voxel coloring

DONE

Blocks	
is blocked by	Shell voxelization

Voxel coloring

TO DO

MTL file format import support

TO DO

Blocks	
is blocked by	OBJ file format import support

Solid voxelization

DONE

As a user, I want to be able to produce a filled volume voxelization of a 3D model.

Relates	
relates to	Resolution
Blocks	
is blocked by	Shell voxelization

Shell voxelization

DONE

As a user, I want to be able to produce a shell voxelization of a 3D model.

Relates	
relates to	Resolution
Blocks	
blocks	Solid voxelization
blocks	Surface voxel coloring

Voxelization algorithm

TO DO

As a user, I want to be able to voxelize a 3D model.

SCHEMATIC file format export support

TO DO

BINVOX file format export support

DONE

XML file format export support

DONE

VOX file format export support

TO DO

Exporting support for various file formats and data structures

TO DO

Octree export support

TO DO

3D array export support

DONE

STL file format import support

TO DO

glTF file format import support

TO DO

OBJ file format import support

TO DO

Blocks	
blocks	MTL file format import support

Automation

TO DO

As a maintainer, I want the project to be heavily automated, so that is easy to maintain.

Automatic testing

DONE

As a repository maintainer, I want all new code changes to be automatically tested.

Automatic publishing

DONE

As a repository maintainer, I want the publishing of new modules to be automated.

3D model importing support

TO DO

As a user, I want to be able to load 3D model files of various types, so that I can voxelize the model.

Requirements Specification (Jira)

Author: André Storhaug

Revision: 07/May/20

Table of Contents

Octree import support

DONE

As a user, I want to be able to load voxel data stored in an octree data structure, so that I can easily view voxel data.

3D array import support

DONE

As a user, I want to be able to load voxel data stored in a 3D array, so that I can easily view voxel data.

Generate three.js mesh

DONE

As a user, I want to be able to generate a three.js mesh from voxel data, so that I can visualize the voxel data.

Relates	
relates to	Customize styling of voxels
Blocks	
is blocked by	Octree backed structure

Voxel data importing support

TO DO

Octree backed structure

DONE

As a user, I want the voxels to be stored in an octree, so that I can manipulate the voxels easily.

Blocks	
blocks	Generate three.js mesh

Customize styling of voxels

DONE

As a user, I want to be able to customize the styling/appearance of the voxels.

Relates	
relates to	Generate three.js mesh

SCHEMATIC file format import support

TO DO

As a user, I want to be able to load voxel data stored in SCHEMATIC file format, so that I can easily visualize voxel data.

BINVOX file format import support

DONE

As a user, I want to be able to load voxel data stored in BINVOX file format, so that I can easily view voxel data.

XML file format import support

DONE

As a user, I want to be able to load voxel data stored in XML file format, so that I can easily view voxel data.

VOX file format import support

DONE

As a user, I want to be able to load voxel data stored in VOX file format, so that I can easily view voxel data.

Requirements Specification (Jira)

Author: André Storhaug

Revision: 07/May/20

Table of Contents

glTF file format import support

DONE

STL file format import support

TO DO

OBJ file format import support

TO DO

File dropper

DONE

Files should be possible to drop on the application in order to load them.

Desktop application

DONE

As a user, I want to be able to be able to run the program as a desktop application, so that it I easy to conduct voxelization of 3D models.

Exporting support

TO DO

As a user, I want to be able to export the vowelized result to a file. The file type should be of one of the types supported by the voxelizer library.

Importing support

TO DO

As a user, I want to be able to import 3D models of various file formats.

The 3D model file formats that should be supported are the ones which three.js supports.

Shell or solid option

TO DO

As a user, I want to be able to toggle between doing a shell or filled voxelization.

Coloring option

TO DO

As a user, I want to be able to toggle between whether or not to do color voxelization.

Clipping

TO DO

Resolution

TO DO

Logo

TO DO

As a user, I want a logo for the application, so that I can easily find it among my other applications.

Automatic updates

DONE

As a user, I want that the application updates automatically when new releases are published. This way, time could be spent actually using the application, not maintaining it.

Voxelization

IN PROGRESS

As a user, I want to be able to voxelize a 3D model easily with the help of the voxelizer library.

Language

DONE

As a user, I want the application to be displayed in my language, so that I can easily understand the content.

Requirements Specification (Jira)

Author: André Storhaug

Revision: 07/May/20

Table of Contents

Upload to GitHub Pages

DONE

As a user, I want to be able to publish the generated documentation to GitHub Pages.

Templates

DONE

As a user, I want to be able to use a custom template for JSDoc, so that I can customize the style of my documentation.

Publishing

DONE

As a user, I want to be able to install the GitHub action from the official GitHub marketplace, so that it is easy to find and install the GitHub Action.

GitHub Action

DONE

As a user, I want to be able to generate JSDoc through a GitHub Action, so that it is easy to generate JavaScript documentation

JSDoc

DONE

As a user, I want to be able to generate JavaScript documentation with the use of JSDoc, so that I can generate high quality documentation.