CT SCANNING AS A TOOL FOR QUALITY ASSURANCE IN ADDITIVE MANUFACTURING

Jens Fossan Tingstad, Eirik Gjertsen Norbye, Sander Island

20 May 2022

Sammendrag

Additiv tilvirkning er i stadig vekst og kan erstatte konvensjonell produksjon på mange områder, og behovet for kvalitetssikring er stort. Jotne EPM Technology er en ledende aktør innen utveksling av produktdata. De vil se på hvordan CT-scanning kan bli brukt for å løse denne oppgaven og hvordan CT-scandataen kan bli distribuert best mulig.Prosjektet har som målsetning å konvertere rådata til det standariserte formatet ISO 10303 (også kjent som STEP, "STandard for the Exchange of Product model data"), noe som vil resultere i en mer sømløs og forutsigbar distribusjon av modelldata.

I denne avhandlingen har det blitt utviklet en funksjonell model viewer og GUI. Et EDMsdk-eksempel ble levert av Jotne, hvor videre utvikling er nødvendig for integrasjon mellom model viewer og GUI. Prosjektet har endret kurs flere ganger i løpet av utviklingen, noe som har påvirket slutt resultatet.

Model viewer er i stand til å prosessere wavefront objekter, tekstur og har et kamera- og bevegelsessystem. Vi klarte å få EDMsdk til å kjøre som forventet ved bruk av AP209 skjemaet. En enkel GUI ble bygget for å håndtere funksjoner som vi hadde planlagt å implementere, men dette ble ikke fullført.

Abstract

Additive manufacturing is increasing in usage and can replace conventional manufacturing in many areas, and the need for quality assurance is high. Jotne EPM Technology is a leader in product data exchange. They want to look at how CT scanning can be used as a tool for this task and how the CT scan data can be exchanged most seamlessly. This project aims to convert raw data to the standardized format ISO 10303 (also known as STEP, "STandard for the exchange of Product model data"), which will enable less friction and greater predictability in the exchange of model data.

In this thesis, components of a model viewer and GUI has been developed. An EDMsdk-example was delivered by Jotne, which requires further development for integration between the model viewer and GUI. The project has changed course multiple times during development, which has affected the results to some extent.

The model viewer we created is able to render wavefront objects, layer texture and has a camera and movement system. We were able to get the EDMsdk example to run as expected using the AP209 schema. A simple GUI was built to handle the functions calls we planned to implement, but this work was never completed.

Preface

This bachelor's thesis is written by Jens Fossan Tingstad, Eirik Gjertsen Norbye, and Sander Island, students at NTNU in Gjøvik, department of Computer Science.

We want to thank Jotne EPM Technology, particularly Henrik Galtung, Tord Kaasa, Remi Lanza, and Arne Tøn, for giving us an exciting assignment. They have provided exceptional feedback and support while also being patient and resourceful when needed. The project has been challenging but interesting while also allowing us to gain knowledge within multiple fields, such as CT scanning, CAD, system development and project management.

We want to thank our supervisor Ivar Farup for giving us valuable guidance and feedback throughout the project.

Contents

Figures

Tables

Code Listings

Acronyms

AP Application Protocol. 8

API Application Programming Interface. 2, 8, 9, 78

CAD Computer-Aided Design. vii, 8, 9, 11

CAE Computer-Aided Engineering. 8

CAM Computer-Aided Manufacturing. 8

CFD Computational Fluid Dynamics. 8

CPU Central Processing Unit. 9

CT Computed Tomography. iii, v, vii, 1, 2, 5, 6, 75, 81

EBO Element Buffer Object. 35, 36

EDM EXPRESS Data Manager. 8, 75

ESA European Space Agency. 1

FEA Finite Element Analysis. 8

GLM OpenGL Mathematics. 25, 27, 47

GLSL OpenGl Shading Language. 36

GPU Graphics Processing Unit. 9, 35, 73

GUI Graphical User Interface. iii, v, 11, 21, 22, 25, 50, 59, 69, 73–75

IDE Integrated Development Environment. 11, 25, 59, 74, 78

IP Internet Protocol. 15

MAIT Manufacturing, Assembly, Integration, Testing. 1

OAIS Open Archival Information System. 1

OpenGL Open Graphics Library. 9–11, 13, 25, 28, 30, 31, 33, 35, 36, 38, 39, 46, 50, 61, 78, 81

PDM Produuct Data Management. 8

PLM Product Lifecycle Management. 8

PMI Product Manufacturing Protocol. 8

RAM Random Access Memory. 9

STEP STandard for the Exchange of Product model data. iii, v, 1, 7, 8, 13, 15, 22, 70, 73–75, 78–81

UX User Experience. 21

VAO Vertex Array Object. 35, 62

VBO Virtual Buffer Object. 35, 36

VR Virtual Reality. 9, 27

Glossary

AP209 Part 209 of the ISO 10303 STEP standard.. iii, v, 70, 74, 81

Assimp Open Asset Import Library (Assimp) is a cross-platform 3D model import library which aims to provide a common application programming interface (API) for different 3D asset file formats [1]. 11, 61, 65, 69, 79

C A general-purpose programming language[2]. 25

C++ A popular high-level programming language created as an extension of the C programming language. 2, 8, 25, 28, 36, 59, 78, 81

CLion A cross-platform IDE for Linux, Windows and macOS made by JetBrains. 25, 59, 61, 70, 74, 75, 78, 81

Clockify An online service for tracking work hours[3]. 19

CMake In software development, CMake is cross-platform free and open-source software for build automation, testing, packaging and installation of software by using a compiler-independent method [4]. 25, 28, 29, 33, 69, 78, 80, 81

Discord A free communication platform that makes it possible to chat with voice, video and messages, as well as sharing files and other media. 19

EDMsdk Jotne's EXPRESS Data Manager software development kit. iii, v, 70, 74, 75, 78, 80, 81

GitLab A web-based platform used for version control in coding using git, while also making cooperation easier. 20, 78

Glad Is an OpenGL Loading Library that helps OpenGL manage and access function pointers more effectively. 16, 25, 29

GLFW Graphics Library Framework (GLFW) is an open-source library for OpenGL. GLFW is a framework that handles windows, contexts and events in an OpenGL-program. It also handles user input. 16, 25, 27, 29–33, 48

Go A programming language developed by Google. 2

ISO The International Organization for Standardization is an international organization that develops and publishes both technical and non-technical standards. "ISO" is not an abbreviation/acronym, but comes from the Greek word for "equal" [5]. iii, v, 1, 2, 7, 8, 79, 80

Java An object-oriented programming language. 2

JetBrains A software development company that makes tools for software developers. 25, 59, 78

Microsoft Teams Is a business communication platform. 19

Python An object-oriented programming language. 2

Qt A software framework for developing Graphical User Interfaces (GUIs). 11, 25, 59, 74

Qt Creator Qt's own IDE for developing Graphical User Interfaces (GUIs). 59, 74

Visual Studio Microsoft's cross-platform IDE for Linux, Windows and macOS. 25, 61, 69, 70, 74, 75, 78

Chapter 1

Introduction

1.1 Background

Jotne EPM Technology is a leader in the development of standards-based software products. They specialize in product data exchange, product life-cycle management, long-term data and product, Open Archival Information System (OAIS)-archiving, data validation and verification, code checking, rule-based data modeling, and cross-platform data sharing within the structure of objects Jotne has produced.

As technology advances, additive manufacturing (3D printing) is increasingly relevant in the aerospace industry. In collaboration with the ESA (European Space Agency)-project METRIC, Jotne is looking at the opportunity of producing a satellite component of aluminum using additive manufacturing technology. The main goal of this joint effort is to use state-of-the-art technology to reduce cost and increase quality and quality assurance to the MAIT (Manufacturing, Assembly, Integration, Testing)-processes for telecom satellites. We use CT scans of the component to assure the required quality, both internal and external structure, and geometric dimensions. This is to validate and document that the final product satisfies all parameters.

Jotne wants to see how the scan data is managed and how it can be used for a digital twin of the component. A digital twin is a virtual representation that serves as a real-time digital counterpart of a physical object; in simpler terms, the digital twin is a copy of an object from the real world to the virtual world. Digital twins are a focus area for Jotne and a vital part of the METRIC project. Important addition is that the digital twin is based on the ISO-standard, ISO 10303, informally known as "STandard for the Exchange of Product model data (STEP)," which is essential in most of Jotne's work.

1.2 Target Group

A target group is a group of people that our product targets explicitly. The use-case for this software is within a small field since Jotne and potential partners will mainly use the software.

1.3 Project Group & Roles

The project group consists of three bachelor's students in Computer Science. One of the students has a bachelor's degree in Machine Engineering from earlier studies and has some relevant knowledge about 3D modeling. From the study program, the students have gained basic knowledge of coding in different languages such as Java, C++, Go and to some extent Python. They also have relevant knowledge within subjects such as computer vision, algorithmic methods, and system development.

The responsibilities and roles within the group have been vague, as all members have done a bit of everything. Still, the prominent roles consist of Jens, portraying as project leader and chairman regarding the follow-up meetings. He was assigned to this role because of his prior knowledge about the subjects we are currently working within. We agreed that he would have the authority to make final decisions regarding project decisions. Sander has been assigned as Scrum master while also responsible for meeting documentation. Eirik has been assigned to the project's documentation, which entails documenting significant and relevant decisions within the project.

1.4 Delimitation

Since an external party provided the project, we expected that a set of limitations would follow regarding the project's implementation. First of all, they demanded that we should write the code in C++, this is because the Application Programming Interfaces (APIs) they provide uses this language.

They also specified that the platform should be compatible with Windows while also being deployable within the ISO standard that Jotne currently works with.

Permissions regarding the use of the report were provided in the contract between the employer and us, which specifies that we have all the rights to the final product, they will cover travel expenses, and potential CT scans we have to perform in the future. If there are any extra expenses beyond this, they will have to be discussed with the employer.

1.5 Employer

Jotne EPM Technology AS
Location: Oslo, Norway
Represented by: Henrik Galtung and Tord Kaasa

1.6 Supervisor

Ivar Farup, Professor
Department of Computer Science
Faculty of Information Technology and Electrical Engineering

1.7 Report Structure

The report structure is categorized into multiple chapters, where we begin with the theory that provides a general description of key concepts that has been used throughout the development phase. After theory, we move on to "requirements," which explains the specifications we had set before the development stage, including use-cases. Moving on to the next chapter, we go through finding the most suitable development method and explaining why we chose this particular method. We also include other sections such as status meetings, time management, and version control. Next, we have included all of the main chapters, where we start with "Process & Implementation," where we discuss the implementation of the software which is followed by sprints. Results go through the product's final results, explaining what we were able to complete and what remained unfinished. In the discussion chapter, we discussed the thought process throughout the whole project in general, while in the conclusion chapter, we talked about thoughts we had regarding the outcome of the project.

Chapter 2

Theory & Technology

2.1 CT Scanning

Computed Tomography (CT) scanning is a technology that uses X-rays to make an external and internal image of an object. It is widely used in medical aspects, but here the focus area is on the industrial use of CT scanning. Industrial CT scanning is being used to inspect both the internal and external structure of a component or object. An image of the internal structure makes it possible to detect flaws, cracks, and other attributes such as porosity. In 3D software, it is also possible to accurately see the dimensions of the scanned part, which is useful to check if the component is manufactured correctly. [6]

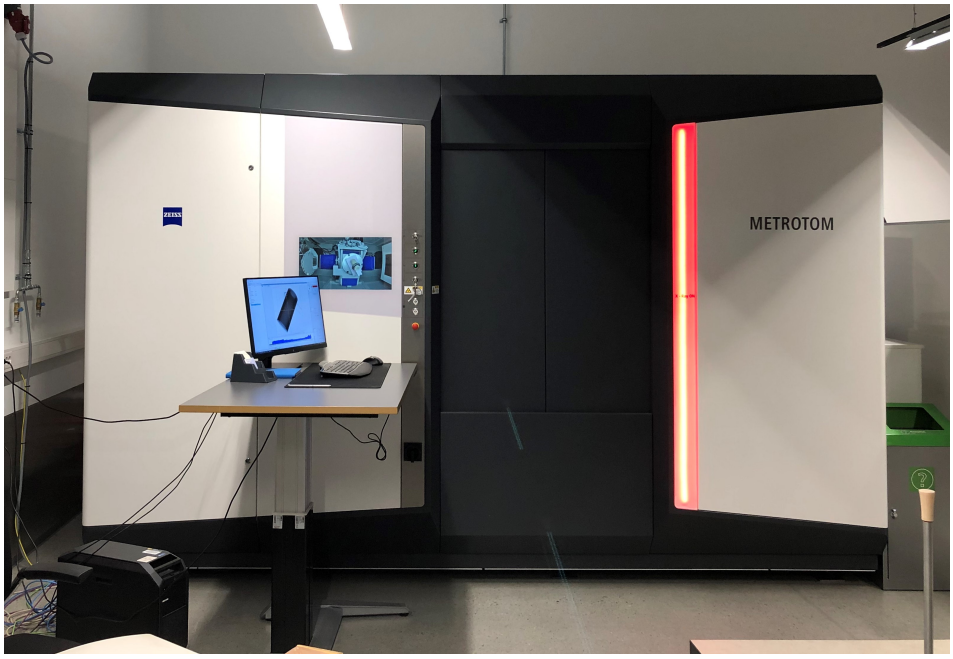


Figure 2.1: Zeiss Metrotom 1500, the CT scanner used in this project

2.1.1 How It Works

A CT scanner is built up by an X-ray generator, a rotating stage where the component is placed, and an X-ray detector field. The X-ray generator produces a cone-shaped beam that shoots through the element, and the detector field detects the X-rays. As the rotating stage is rotating, the generator and detector make a considerable amount, usually thousands of 2D images, that are later processed to a 3D image.

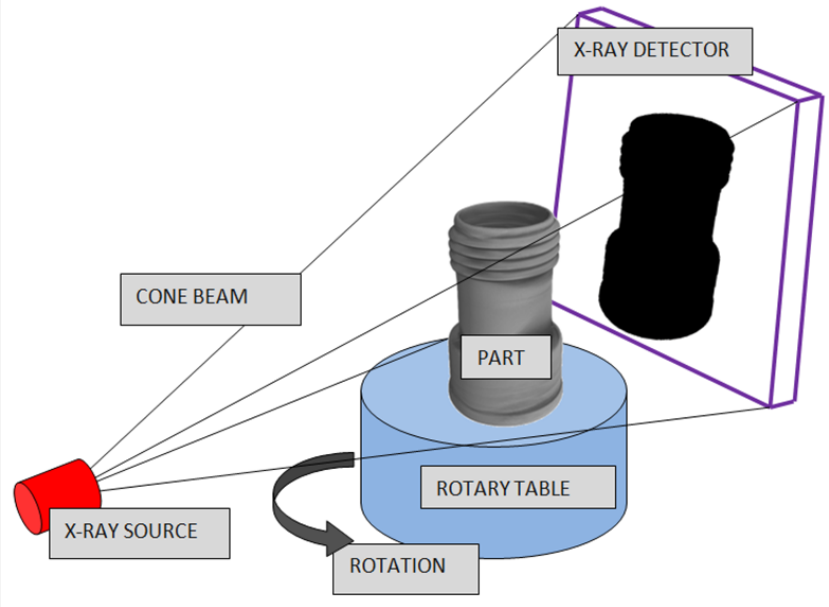


Figure 2.2: An image showing how an industrial CT scan is done

(Downloaded from Wikipedia, public domain)
(https://en.wikipedia.org/wiki/File:Ct_scan_cone_beam.png)

2.2 Voxel Models

Voxels are explained as three-dimensional pixels, and the word itself comes from "volume" and "pixel." A pixel, which comes from "picture" and "element," is a small part of a two-dimensional picture. Voxels are the same, but for three-dimensional images. Voxels are commonly shaped as cubes, and together, all these volume elements make up a 3D voxel model.

A voxel model can be of different resolution, just as regular 2D pictures, depending on the size of each voxel. A higher resolution model will give a much more detailed representation than a lower resolution image. The torus in Figure 2.3 is represented in three different resolutions, and the model with higher resolution gives a much more accurate and rounder shape.[7]

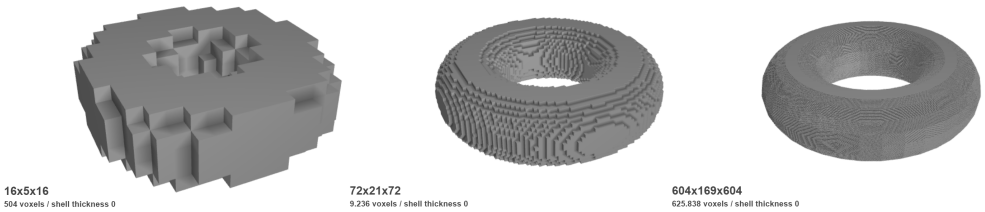


Figure 2.3: Higher resolution gives more detailed images

2.3 Mesh Models

A mesh model is a 3D model built up of polygons. The polygons are usually triangles but can be other polygons as well. With the use of triangles, a program can attach triangles to make up the shape of a specific object. A 3D mesh is generally a hollow shell of an object, which means there is an empty internal structure.[8]

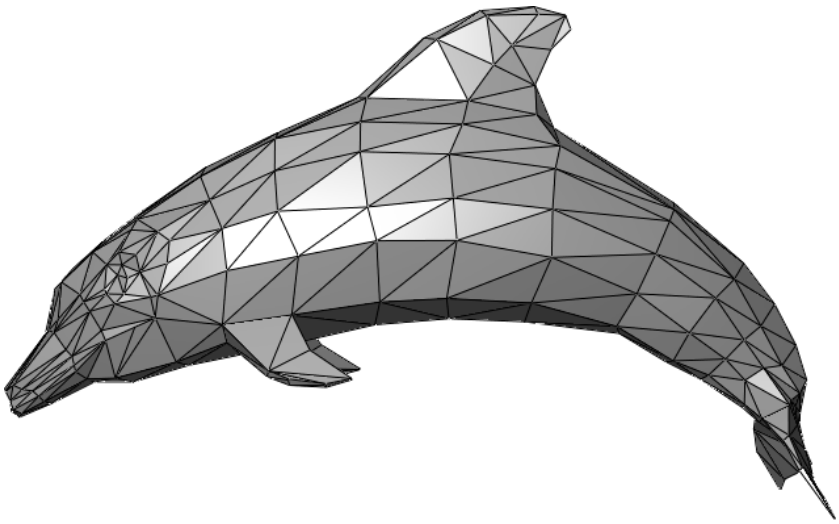


Figure 2.4: An example of a mesh model

(Downloaded from Wikipedia, public domain)
(https://en.wikipedia.org/wiki/Polygon_mesh/media/File:Dolphin_triangle_mesh.png)

2.4 ISO 10303 (STEP)

ISO 10303 is an ISO standard for representation and exchange of product manufacturing information. The standard's official title is "Automation systems and integration — Product data representation and exchange." It is informally known as STEP, which stands for "STandard for the Exchange of Product model data." Devel-

opment of STEP began in the mid-1980s to make sharing and exchanging files containing 3D data across different software much easier by providing a standardized file format to work on 3D models. The STEP file format is used in Computer-Aided Design (CAD), Computer-Aided Manufacturing (CAM), Computer-Aided Engineering (CAE), product data management, enterprise data management and other CAx systems.[9][10]

The ISO 10303 standard is available as a series of documents. Most of these documents are data models written in the EXPRESS data modeling language, which is documented by the same standard. Each data model part covers a certain concept, that is, mathematical constructs, product description, product versioning, geometric shapes, product properties, meshes and materials. These data models are bundled in Application Protocols (APs), to cover a complete engineering domain. Most known are the APs AP203, AP214 and AP242, which are implemented in CAD applications. AP209 is a superset of AP242, which in addition to CAD, Produuct Data Management (PDM), PDM and Product Manufacturing Protocol (PMI), also supports Finite Element Analysis (FEA), Computational Fluid Dynamics (CFD) and composite materials. AP209 is the STEP Application Protocol we wish to support in the STEP translation process.[11]

2.5 AP209

AP209 (Multidisciplinary Analysis and Design) is an Application Protocol in the ISO 10303 standard. It is a data model written in the data modelling language EXPRESS which covers the representation of simulation, CAD, PDM, Product Lifecycle Management (PLM) and PMI data. Similarly, as AP242 is used by CAD tools to share, store and archive design models independent of proprietary formats, AP209 is intended as a format for exchanging, storing, and archiving models from CAE applications.[11]

2.6 EXPRESS Data Manager

EXPRESS Data Manager (EDM) is a database system that implements the methodology of ISO 10303. It can be used to create and manage databases that uses EXPRESS schemas. This allows the creation of databases based on ISO 10303 Application Protocols such as AP242 or AP209. The databases can be local or hosted by an EDM server for remote connection. EDM provides a client application and an API (in multiple languages) for managing, among other things, databases, and servers. It can also generate specific APIs (C++) for EXPRESS schemas. In this project Jotne provided us, in addition to EDM, a C++ API generated from the AP209 schema. This allows to us to create an AP209 model in an EDM database directly from our application. Such AP209 model can then be exported to a STEP (.stp) file.[11]

2.7 OpenGL

Open Graphics Library (OpenGL) is a cross-language, cross-platform API for rendering 2D and 3D graphics. OpenGL interacts with the computer's Graphics Processing Unit (GPU) to achieve hardware-accelerated rendering. OpenGL comes with a large number of functions that you can use to make complex 2D and 3D graphics from simple geometric entities such as points, lines, and polygons. The interface also contains functions for rendering scenes with lighting control, object surface properties, transparency, anti-aliasing, and texture mapping. OpenGL enables software developers to create high-performance graphics applications for areas such as CAD, medical, manufacturing, Virtual Reality (VR) and entertainment such as gaming and content creation. Although OpenGL was released in 1992 by Silicon Graphics inc., it is still managed and maintained by Khronos Group and able to use all the features of the newest graphics hardware.[12]

In the sections below, we will cover functions within OpenGL that are relevant to the project. We will use shaders, transformations, coordinate systems, and camera functionality in the development phase. These are essential functionalities to display a functional 3D model. Other features could be used as well, such as textures and lightning. However, these features are used for design purposes, which is not the main priority.

2.7.1 Shaders

Shaders are mainly divided into vertex- and fragment shaders, where the main task of the vertex shader is to specify the calculations that transform 3D coordinates of vertices to 2D screen coordinates. The coordinates of each triangular element are passed to the fragment shader, which specifies the calculations for the coordinates and colors of the pixels within each rendered triangle. The calculations are performed by the GPU.[11]

2.7.2 Transformations

Transformations can be described as vector manipulation because we are working with different vector structures. Transformations are helpful in making adjustments to positional or size-based functionality. Developing these transformations can be accomplished using different approaches; an example of that is by changing the vertex positions manually and updating the buffers for each frame, but that is an unprofessional approach that does not make use of the functionality that OpenGL offers. The most common practice would be to use identity matrices, translation vectors, and transformation matrices to perform the transformations. This approach also helps reduce the strain on the Random Access Memory (RAM) and Central Processing Unit (CPU) because manually updating each buffer each frame requires more computational power.

2.7.3 Coordinate Systems

In terms of 3D geometry, a coordinate speaks to a position in space. It is defined by a position along the x-, y-, and z-axis.

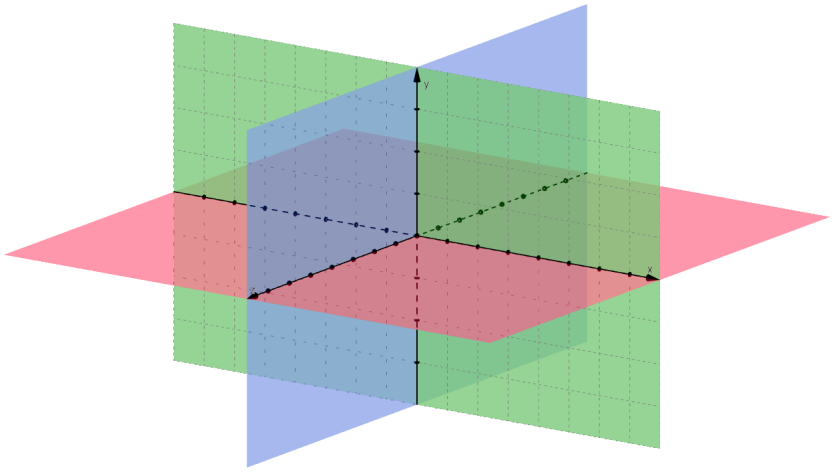


Figure 2.5: An example of a three dimensional coordinate system

2.7.4 Local Space

Local space can be described as the local space according to our object, as in the origin of our object. When we create objects in OpenGL, the origin of our object is generally positioned at (0,0,0), but the object may be positioned differently."Probably all the models you've created all have (0,0,0) as their initial position. All the vertices of your model are therefore in local space: they are all local to your object." [13]

2.7.5 World Space

World space means transferring our local object out into world space, scattering our objects into a "larger" world, thus giving your vertices different positions. We can accomplish this by using a "Model matrix." "The model matrix is a transformation matrix that translates, scales, and/or rotates your object to place it in the world at a location/orientation they belong to." [13]

2.7.6 View space

View space can be described as a camera that stimulates the front of the user's view. "This is usually accomplished with a combination of translations and rotations to translate/rotate the scene so that certain items are transformed to the front of the camera." [13]

2.7.7 Clip Space

Clip space is a conversion tool used after each vertex run. Since the vertex expects NDC- coordinates, "NDC –coordinates are coordinates that are normalized, which means they are within –1 to 1 on the xyz-axis. Clip space circumvents this, making it possible to have coordinates outside this normalized range.

2.7.8 Camera View

The camera viewer enables the user to move around the object, making it more manageable to get a closer look at different angles of the object. This will help determine and reveal flaws with the rendered object. OpenGL do not have any specific camera functionality, which means we have to use a different strategy. We can use a trick to simulate a camera effect, which is done by moving the objects in the scene instead of having a camera that is moving. By moving the objects backward or forward in the background, we create the simulated effect of the camera.

2.7.9 Assimp

Assimp is an extensive library that contains various model struct data, which enables users to import different types of models into a pre-made struct. This simplifies the process of importing models since, normally, without the library, we would have to build these struct variables manually. Assimp is mainly used for game-related scenes, which contain texture and mesh-based scenery-modeling, but it also supports various 3D-printing and CAD-formats.

2.8 Qt

Qt is a software framework for developing modernGraphical User Interfaces (GUIs) interfaces. Qt comes with an Integrated Development Environment (IDE) which is called Qt Creator. In the IDE, the GUI creation is fairly simple as you can drag and drop buttons and other objects to the window.

2.9 Wireframe

A wireframe is a type of sketch where we draw outlines of the desired design. This will give a rough idea of how much space is needed, what type of content we should include, and what functions seem necessary. Wireframes can often be portrayed as a simplistic designs; this is to avoid the risk of misunderstanding between the designers, developers, and the employer. Elements that are often prominent within wireframes consist of "squares and rectangles that represent images and icons, text that is visualized with ’lorem ipsum’ or lines inside a box, buttons that are either rounded or darker grey, icons that are either small symbols or squares

with an x." [14] Wireframes have three different variants that are most commonly used. These are low fidelity, medium-fidelity, and high fidelity wireframes.

Low fidelity is more of a user-centered process, where the design process is not at the center of focus. Here we will often see a simplistic design, often hand-drawn, with only the essential features of the prototype.

Medium fidelity is where the design process becomes more refined and detailed. Here the focus lies more on the functionality within the prototype while also giving some thought to the placements of the different elements.

High fidelity is a prototype that should be as close to the final product as possible; here, you have taken a great deal of consideration towards coloring and design issues while also having functionality that is relevant and useful inside the prototype.[15]

Chapter 3

Requirements

In this chapter, we will cover a set of requirements within our system. The requirements has been set based on the project description. (Appendix A), meetings and decisions with our employer.

3.1 Use Cases

This section will go through use case, design, and system requirements. We have included tables that will help establish an overview.

Figure 3.1 tries to convey how the user would operate within this system. The diagram has a user, which can import, export, view, show info, and calculate density. These are the main features that will be available when opening the application. While on the other side of the application, there is a STEP-converter, which Jotne is responsible for, that converts the model into STEP format, and the OpenGL application that is responsible for the "model viewer."

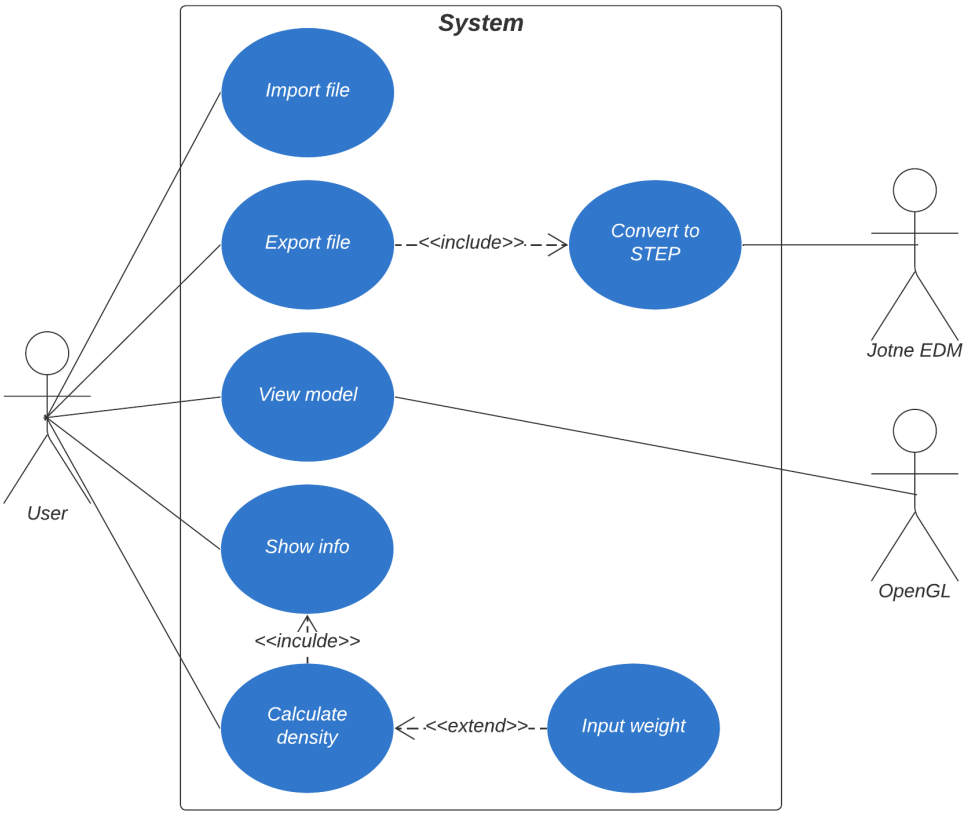


Figure 3.1: Use case diagram

Tables 3.1-3.5 describes how each use case works.

Table 3.1: Import file use case table

Action:	Import file
Actor:	User
Goal:	Import file to view or convert
Description:	The user gets directed to the file explorer and selects a file to import to the program.

Table 3.2: Export file use case table

Action:	Export file
Actor:	User
Goal:	Export file or convert to STEP
Description:	The user exports the already imported file as a STEP file.

Table 3.3: Inspect model use case table

Action:	Inspect 3D model
Actor:	User
Goal:	Inspect 3D model for flaws
Description:	The user inspects the 3D model in the graphics view

Table 3.4: Get model information

Action:	Get model information
Actor:	User
Goal:	Gain knowledge about the model data
Description:	The user gets information about the 3D model.

Table 3.5: Calculate density

Action:	Calculate density
Actor:	User
Goal:	Gain knowledge about the object's average density
Description:	The user inputs a known weight of the object so the program can calculate the average density.

3.2 System Requirements

In this section, we will cover all of the requirements we thought were appropriate to list, according to the development plan we made prior to the development phase. The list will be based on low to high priority, where the functions given high priority are essential for the product to be complete. In contrast, the lower priorities are not defined as "Must have," but they would complement the application. These requirements were made through discussions with the employer, ensuring that we had the right priorities from the start of the development.

3.3 Security Requirements

This section will cover all of the requirements we have regarding security within the application. The security aspect within the product is important considering we have received a custom STEP library from Jotne, which they have explicitly told us to keep private. Jotne has already implemented some preventive actions within the application by providing a license registration for each Internet Protocol (IP) address that tries to use the application. Jotne directly provides this licence key. We, however, have the responsibility of making sure that all of the code is written within the terms of confidentiality and integrity. Availability is not that relevant to our project since the software will not be running on a server that demands to be active at all times; it is software that will be run locally on the computer.

Table 3.6: System Requirements

System Requirements		
Function	**Description**	**Priority**
Convert to STEP-format	Converts a voxel model into STEP-format	High
Import voxel file	Takes voxel models as input	High
3D environment	Displays the voxel model in 3D space	Medium
Prompts showing additional information	Showing information like density, porosity, texture etc.	Low
Camera functionality	Enables the user to move around the model	Medium

Confidentiality is the practice of keeping private and sensitive information separate from the public. Integrity within the application entails giving the appropriate access rights to eligible people. Integrity is an essential aspect of security; if the access rights are not managed properly, we increase the risk of people who are unauthorized to modify or change the code intentionally with malicious intentions.

3.4 Testing

This section will cover the requirements we set regarding tests within our application. The tests would include functional testing and validation testing. Functional testing includes checking that all functions work correctly within the parameters that have been set. Validation testing is used to make sure that the application works correctly. It is responsible for checking that specific inputs and outputs are correct.

The validation tests would mainly include if-loops, ensuring that the input inserted into each function is valid; it would be logged into the terminal if not. The same procedure would be done for output. We currently have not implemented a test environment within the application, but we have included tests for compilation and linking status within our fragment and vertex shader which were important components. We also included some tests regarding Glad and GLFW, which checks if the libraries were initialized correctly, if not they would print out an error statement.

Chapter 4

Software Development Method

When researching different development strategies, we had a few demands. The first being it must be an iterative process, meaning it is possible to make changes and updates to the planned schedule as we go. This is especially important for code-heavy projects since making changes in one section of the code might mean reviewing other parts of the project. Secondly, we required a method that broke down the project into smaller bits for more manageable review sessions. Since we plan to rely on input from the employer and supervisor, we want smaller chunks of work so it is quickly approved or rejected during discussions.

Based on this, we quickly decided to research agile development methodologies. After reviewing multiple methods, we decided on something that matched all our criteria and formalized the workflow we had already intended.

Since creating an efficient workflow is essential, we decided to use Scrum, an agile development strategy suited for small project groups. We chose this strategy as it focuses on short sprint-based development cycles where we set ourselves small goals with short and specific deadlines. After each sprint, we will review our work in collaboration with our supervisor and employer. Here we will discuss problems and produce viable solutions. This is to keep the next goal as straightforward as possible and ensure it is within the realm of reason that the work is finished within the given time frame. Another reason we chose this strategy is because of how available and flexible our employer is; if we have any questions outside of the arranged meetings, they are always there to answer them.

4.1 Scrum

Scrum is described as a set of frameworks to help and encourage the team to work and reflect on each win and loss. Scrum enables the team to learn through experience and failure while adjusting accordingly, depending on our results. Scrum is defined as agile since it circles around flexibility, adaptability, and quality work that should be executed within a short timeframe.

4.1.1 Sprints

Sprints can be described as the amount of work needed within a set time. Sprints are effective for getting work done in a structured and time-efficient manner, which keeps the agile team free from headaches. "With Scrum, a product is built in a series of iterations called sprints that break down big, complex projects into bite-sized pieces" .[16]

4.2 Kanban

We used a kanban board to keep track of what tasks to do. A kanban board can be described as sticky notes sorted into categories that describe a task's stage during the project. This is a straightforward method for organizing tasks during a project.

We have used kanbantool.com[17], which provides a simple kanban board. The categories we have used are "to do," "doing," and "done." This is a very simplistic and transparent way of sorting the tasks. kanbantool.com allows for delegating tasks to a person in the group, however the free license of the software only accepts two users per kanban board. As we were only three people in the group, we decided that setting a name on each task was unnecessary as we easily would remember who was set to do what.

Each Scrum sprint will have its kanban board. At the beginning of every sprint, a new kanban board will be set up with new tasks. When a group member completes a task, the other two members will review the work before placing the card into the categorized sections. If tasks are done before the sprint is over, new tasks will be discussed and put onto the kanban board.

An additional feature the software provides is the opportunity to categorize each card in colors. Table 4.1 shows how we have categorized with colors.

Table 4.1: Kanban category colors

Category	**Color**
Research related	Yellow
Organizational	Orange
Implementation related	Blue
Report related	Green

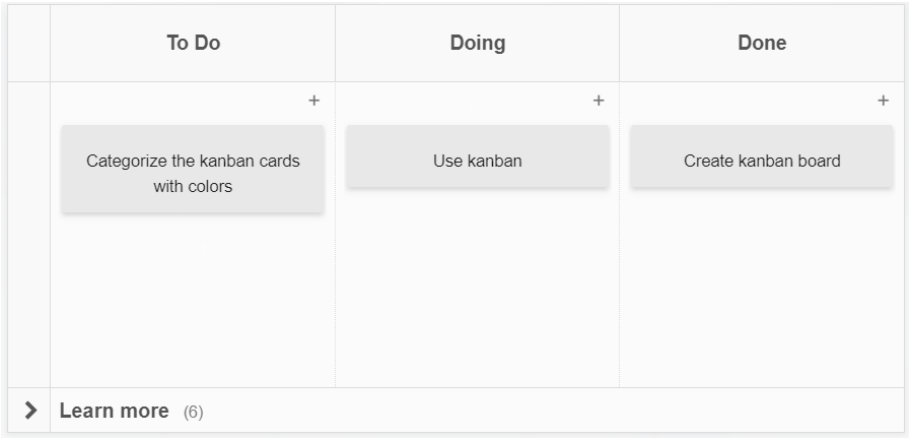


Figure 4.1: An example of a kanban board

4.3 Status Meetings

Status meetings will be performed every other week to keep our employer up to date. We will also have weekly status reports with our supervisor to get feedback concerning changes we make within our project. These meetings will be of high value, ensuring that the choices we make under development correlate with the employer's expectations.

4.4 Communication

To ensure seamless internal communication, we have set up a Discord server where we can share everything related to the project in a structured manner. We also have most of our digital meetings in a voice channel, and as a quick way to send messages, we have used Facebook Messenger.

Meetings with supervisor and employer have been held on Microsoft Teams. Several physical meetings have also been done with the supervisor and two with the employer.

4.5 Clockify

To track time usage, we have used Clockify, which is an online service that makes it possible for all group members to track their work time. In Clockify, we set predefined categories we had worked on to track how much time we spent on each phase of the project. Clockify also gave us an opportunity to give precise information concerning what was worked upon at different time stamps.

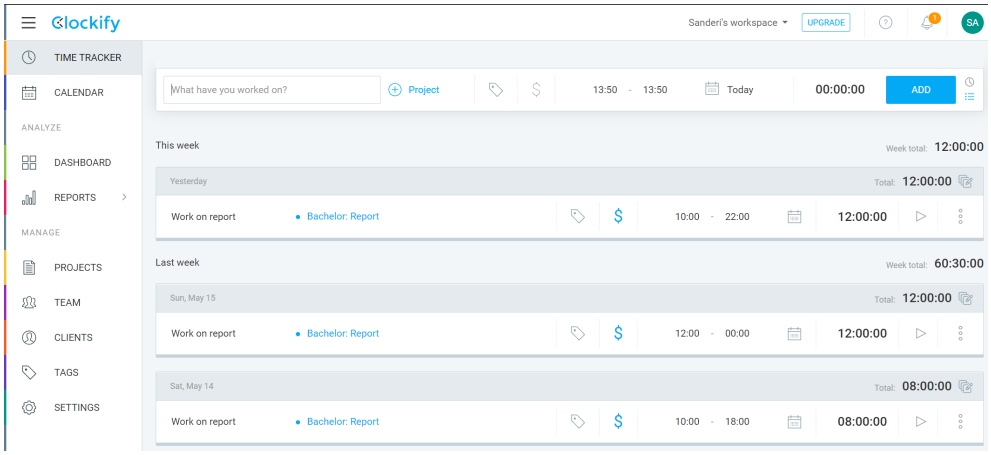


Figure 4.2: Clockify interface

4.6 Version Control

Version control is often used in both smaller and larger projects where the development phase is extended over weeks or months. This method is useful since it is possible to browse through previous versions of the project. If there is a problem within the project, you can go back to previous versions and try a different approach. In this project we will take use of Jotne's GitLab repository. In this repository, we operate with different branches for each person in the group while also having one master branch where the code that is completed will initially be pushed to. This is to ensure that any changes made within the code never reach the master branch without proper reviewing.

Chapter 5

Design

In this chapter, we will cover the backend and GUI design. To start with, we cover the backend of the system and how this will be implemented while also going through some features with the help of use-case tables in the requirements chapter (3).

5.1 Backend

The backend of the GUI would consist of a controller inside the viewer application. This controller would mainly consist of "action" based code that operates through "if" loops. These commands will be based on user operations within the application. An example of this is if the user clicks on the "import" function, this operation would activate a function within the controller that is linked up with the import feature inside the viewer.

5.2 GUI

GUI is an extra addition to the project that we wanted to implement. The focus area within the GUI was to ensure simplicity for the user, making the software user-friendly. We would approach this by performing usability testing and retrieving feedback from users to ensure that the final product would be within high standards. Apart from a simplistic and clear design, User Experience (UX) and design principles will not be a priority as the primary focus lies within the functionality. Table 5.1 shows how we have prioritized functions in our GUI.

5.2.1 Buttons

The buttons are the most crucial feature within the GUI since, without them, the user would not be able to operate within the interface. The import button is responsible for importing files that are within the specified format, while the export

Table 5.1: GUI Features

GUI Requirements		
Feature	**Description**	**Priority**
Import button	Opens a file explorer where the user can import a voxel file	High
Export button	Exports an AP209 STEP file	High
Graphics view	Displays the voxel model	Medium
Mesh import	Functionality for loading mesh models	Low
Prompts showing additional information	Showing information like density, porosity, texture etc.	Low
Jotne colors	Using Jotne's turquoise color in the GUI. Not very important, but nice to have	Low

button is responsible for converting the imported file into STEP-format. These buttons are essential for the software to work as intended.

5.2.2 Other Features

Other features are functionalities within the GUI that are not crucial for the interface to work, such as "calculate density," where the user inputs the object's weight to calculate the average density. Another eye-catching functionality would be the graphics viewer, where we can view and rotate the model that has been imported into the GUI. These features are not essential, but they would complement the interface, making it more informative and exciting.

5.2.3 Wireframe

Figure 5.1 is a high fidelity wireframe that aims to be as close as possible to the final product. The wireframe contains all the desired features while still maintaining a clean and simplistic design.

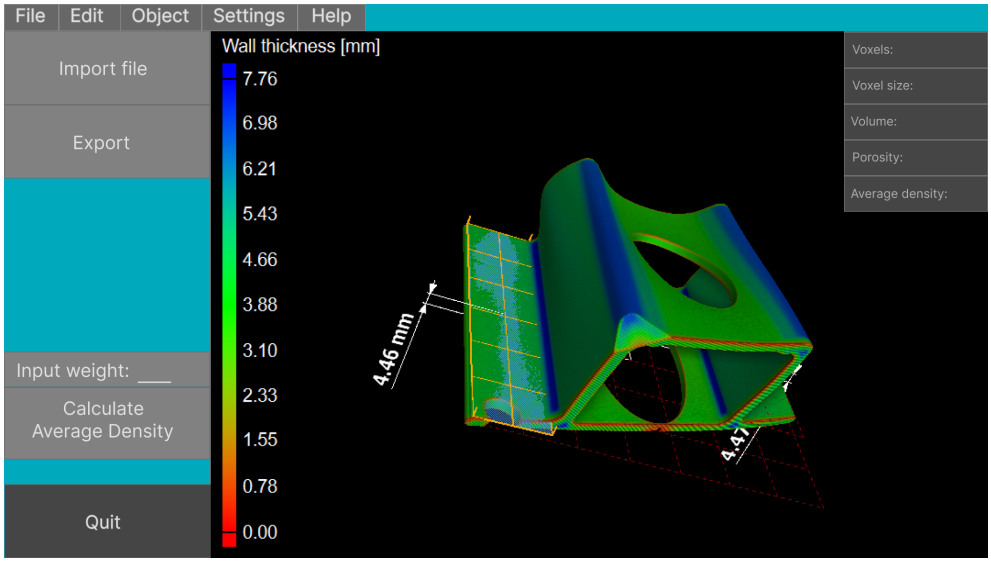


Figure 5.1: A wireframe of an optimal GUI

Chapter 6

Process & Implementation

In this chapter, we will cover the process behind implementing the software. The software development is executed according to the Scrum framework with short sprints, followed by kanban boards included in each sprint. The process behind the implementation will be followed by code listings that will be explained thoroughly in each sprint intermission.

6.1 Setting Up Environment

The project was built using CLion, which is a cross-platform IDE for C and C++ developed by JetBrains. This platform does not have any included plugins for OpenGL integration, which meant we had to integrate it manually. This was accomplished by importing external libraries and using pre-compiled binaries that matched our build system. We used several OpenGL based libraries for ease of certain operations, as doing it manually would be time-consuming and inefficient. We made use of GLFW for window and context handling, OpenGL Mathematics (GLM) for mathematical operations and Glad as a loader. We had to use CMake to include and link the different libraries with the folder structure we currently had. We also used a Visual Studio compiler to compile our project in CLion.

When we built the GUI, we chose to use Qt, which is commonly used within the development of modern GUIs. It is also the preferred choice for integrating with separate IDEs.

6.2 Scrum Sprints

This section will cover all of the sprints throughout the planning and implementation phase. Each sprint will include a summary explaining the goal of the sprint, followed by an intermission.

6.2.1 Sprint 1

This sprint consists of important sections we need to cover in the project plan (Appendix C). We used sprints in the project plan, hoping it would prove advantageous later in the development stage.

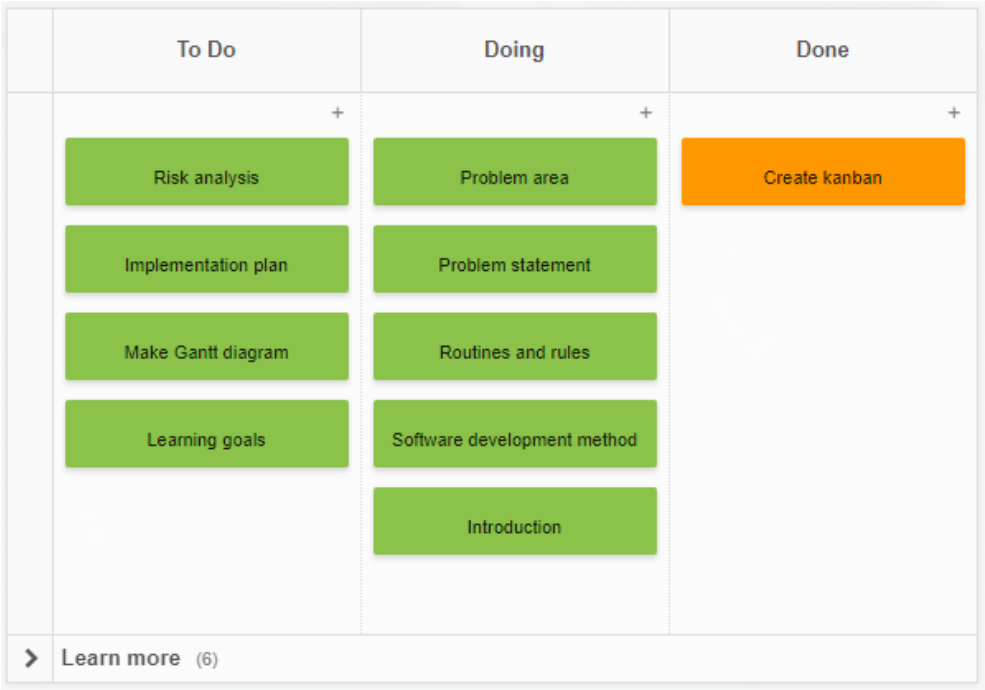


Figure 6.1: Kanban board from the start of sprint 1

Intermission 1: The result of intermission 1 is according to the project plan (Appendix C).

6.2.2 Sprint 2

This sprint consists of all the participants researching, getting an overview of what we have to do, and deciding upon which approach benefits us the most. The first sprint had a timeframe of two weeks, where all participants discussed what they should work on in different stages of development.

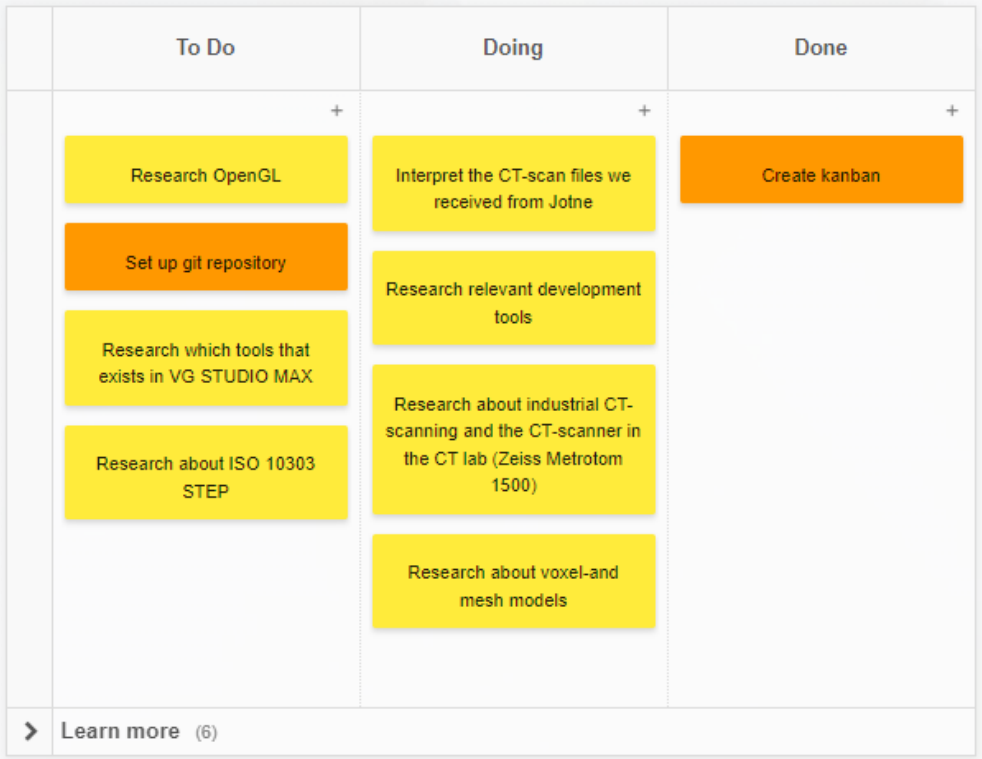


Figure 6.2: Kanban board from the start of sprint 2

Intermission 2: Taking from the results and discussions of this sprint, we set up a schedule for who should be working on what using our kanban board. Initial delegation states that two group members should be assigned to code related work, while the other should have the written report as their primary responsibility.

During research, we found several approaches to implement libraries and discussed which libraries to choose from and which to disregard. Ultimately we settled for GLFW and GLM. Kan vel nevne noe om OpenGL her?

Logistically we were situated in the VR lab, where work could be done in peace. However, due to new restrictions in the new booking system, we were not granted access to admin users. The group's momentum was affected by the lack of these access rights, as certain installations and implementations required supervision of third parties that were often unavailable.

6.2.3 Sprint 3

A three-week period was allotted for re/-learning C++ syntax and a light introduction to working with OpenGL. We did some C++ exercises and started on the introduction chapter on "learnOpenGL"[18], which can be seen as an online textbook for learning to work with computer graphics. We set up an initial model viewer for viewing simple 2D models based on "learnOpenGL." Later we played around with different functionality, color allocation, placements, orientations, and scaling until we were familiar with our current functionality. This meant implementing a build system for including, linking, and implementing different libraries to get access to the required functionality. CMake seemed like the perfect tool for the job.

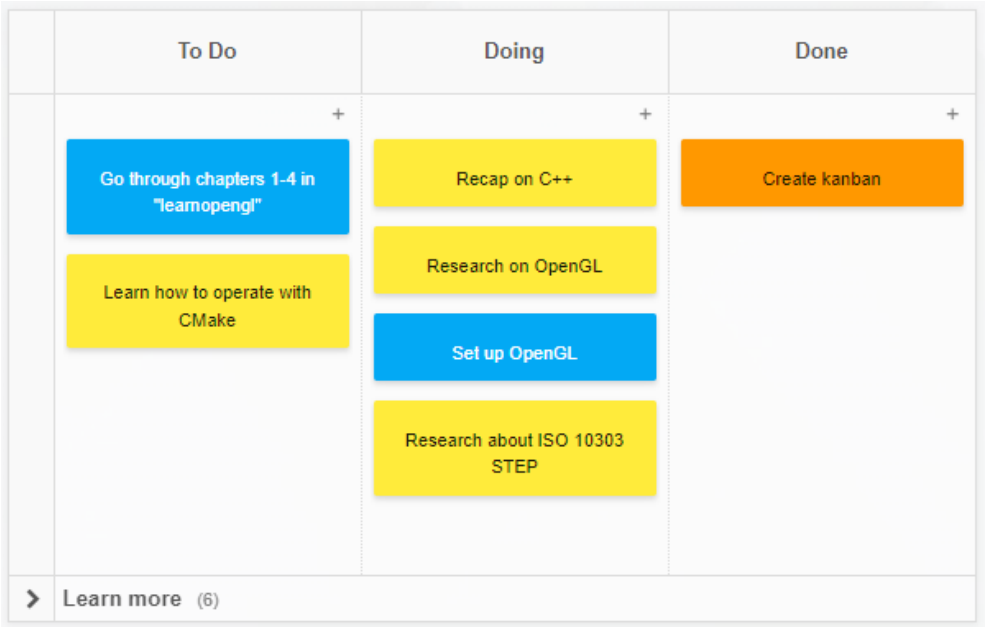


Figure 6.3: Kanban board from the start of sprint 3

Intermission 3: We started the implementation of OpenGL by linking the required libraries with our project using CMake. This took a lot of time and effort, as the build system was difficult to grasp at first glance. This sprint had the steepest learning curve as we implemented multiple concepts. Using CMake to define what to include and how it should be linked proved quite a challenge, but we managed to get a stable build in the end.

```
    cmake_minimum_required(VERSION 3.12)
project(Gltest)

set(CMAKE_CXX_STANDARD 14)


#Header directory
include_directories(C:/glfw-3.3.6/build/include/GLFW)

#External library link
link_directories(C:/glfw-3.3.6/build/lib)



add_executable(Gltest main.cpp)

target_link_libraries(Gltest glfw3)
```

Code listing 6.1: Initial CMake file

In the above snippet, you can see how we initially linked the GLFW library to our project. First, we included the directories where the header files were located. In this iteration, the paths were hardcoded to the specific machine. We used this technique originally to make sure the project worked as intended before generalizing the paths, mainly since we did not have enough experience with CMake as a build system to implement this yet. Secondly, we link the libraries to our project using the link_directories function and use the *add_executable* function to set which executable(s) should be built from the source files. Lastly, we link the GLFW library to our project using the *target_link_libraries* function; this is to ensure that the header files get access to all the functions and class definitions it needs. We also added the Glad library; this is useful as there are many different drivers that support a variety of graphics cards. This means the location of different functions is not known before they are queried at runtime. Luckily, the Glad library helps us define and store this information in function pointers for later use.

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

#include <iostream>
```

Code listing 6.2: Includes

Now we can include the headers for both Glad and GLFW, and make use of their functionality.

```
int main()
{
  // Set up GLFW, set glfw version to 3 or above
  glfwInit();
  glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
  glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
  glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

  //Create window, if window is not created print statement.
  GLFWwindow* window = glfwCreateWindow(screen_width, screen_height, "JESviewer",
     NULL, NULL);
  if (window == NULL)
  {
    std::cout << "Failed to create window" << std::endl;
    glfwTerminate();
    return -1;
  }
  //set current glfw context to created window
  glfwMakeContextCurrent(window);
  //Callback to scale window if scale is changed
  glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

  // Makes glad load function pointers to current driver, print error statement if
     glad is not working properly
  if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
  {
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
  }
  //Render loop
      while (!glfwWindowShouldClose(window))
  {
    // input
    //Input is linked to current context (window)
    processInput(window);

    // render
    //Sets background color taking RGBA input
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    //swap the color buffer, check for events
    glfwSwapBuffers(window);
    glfwPollEvents();
  }
  // clear all allocated glfw resources
  glfwTerminate();
  return 0;
}
```

Code listing 6.3: First window

We start our main function by initiating GLFW with *glfwInit*, this lets us configure GLFW with which version of OpenGL we are using. In this case, we are using 3.3,

so we set both context versions to 3. Finally, we set the OpenGL profile to the core profile, so we do not have to worry about backward compatibility.

Now we create our first window by calling the *glfwCreateWindow* function, we set it's size parameters to predefined variables *screen_width* and *screen_height*. Then we name the window, meaning the text that appears above the window when it is rendered. The two last variables are for fullscreen compatibility; here you can reference different monitor types to make it scale to fullscreen mode automatically. We will not be using this, so we set them both to NULL. Finally, we run an if-loop to check if the window was created correctly. If it was not, we print an error statement and clear the allocated GLFW resources. However, if it works properly the function returns a window object which we then set as the main context on our current thread.

To access more OpenGL functionality we have to initiate glad. We do this with an *if-statement* where we use *GLADloadproc* to load our OpenGL function pointers, and *glfwGetProcAddress* to define the correct loading function for our operating system. An error statement is printed if the function pointers do not load. In order to keep the window scaling properly, we introduce a callback function (Code listing 6.4) to handle scaling. The function *glfwSetFramebufferSizeCallback* takes our window and current scale as an argument and changes the viewport accordingly.

```
1
2 void framebuffer_size_callback(GLFWwindow* window, int width, int height)
3 {
4   glViewport(0, 0, width, height);
5 }
```

Code listing 6.4: Scaling function

This scaling function takes a window, width, and height as parameters. Then it adjusts *glViewport* to the new scale. The viewport function is used to set the size of our rectangle(window); the first two parameters are the x and y coordinates of the bottom left corner of the rectangle. The last two parameters are in the top right corner.

Our render loop starts by checking whether or not the window should close; it does this at every iteration of the while loop. Then it runs a function for processing the input since the previous frame(one iteration of the while loop = one frame).

```
1 void processInput(GLFWwindow *window)
2 {
3   if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
4     glfwSetWindowShouldClose(window, true);
5 }
```

Code listing 6.5: ProcessInput function

Right now, our function for processing user input only handles closing the window by pressing the escape key while it is active. The GLFW library has functions for all of this; here we check to see if the escape key is pressed and if it is, we set *glfwSetWindowShouldClose* to true. Thereby the window closes.

We can now start to play around with the window; emphglClearColor sets the background color in our window by *RGBA* input, where the first three values decide our color and the fourth our windows background opacity. An input of (0,0,0,1) as shown in Figure 6.4 gives a completely black background. Now we can call the *glClear* function by passing it *GL_COLOR_BUFFER_BIT* to clear the current color values and assign it the *glClearColor* values we just set.

The *glfwSwapBuffers* functions swaps out the color buffer for the previous render loop iteration for the new one. The color buffer is a large 2D buffer that contains color values for all pixels on the screen, and for changing color states, this needs to be done for each iteration. Lastly, we poll for events, any user input like keyboard presses and mouse movement will be recognized here, and the corresponding functions called. At the end of our main function, we call *glfwTerminate* to clear all resources allocated to GLFW.

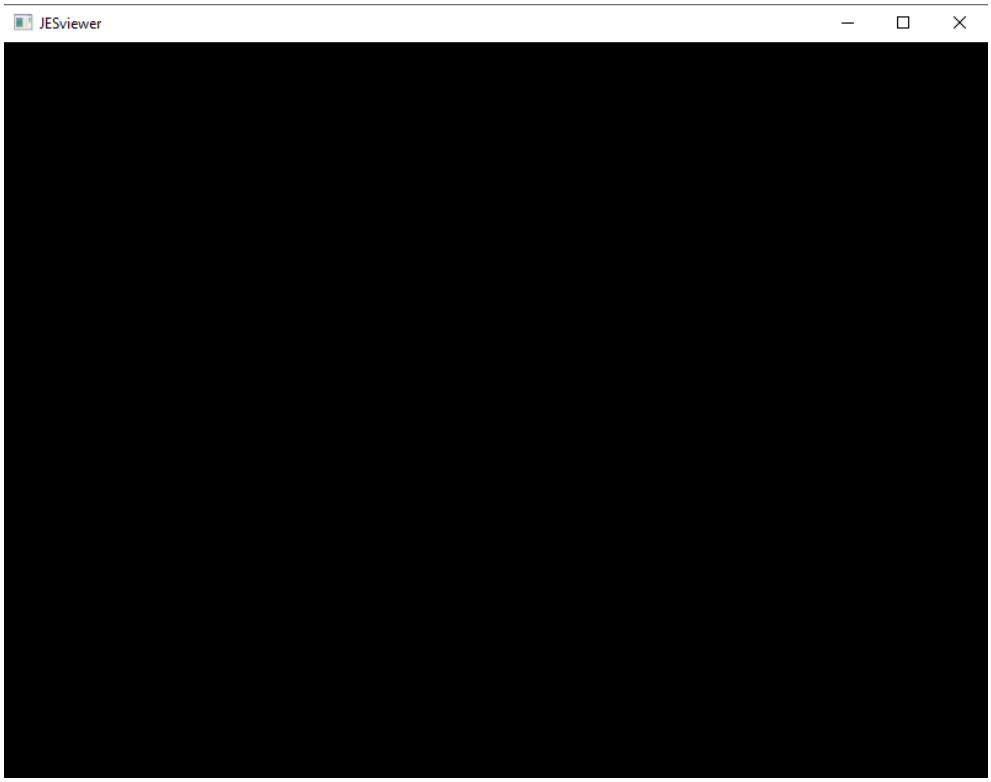


Figure 6.4: Current window

6.2.4 Sprint 4

This sprint is more code-focused as we go deeper in exploring OpenGL and start to introduce complex concepts. We struggled with all of the imports we had to manage in order to get the libraries working correctly. This resulted in using more time to understand CMake. We knew this would be useful for us later, as we would need to import many more libraries further in the development process. This caused the sprint to be extended an additional five days.

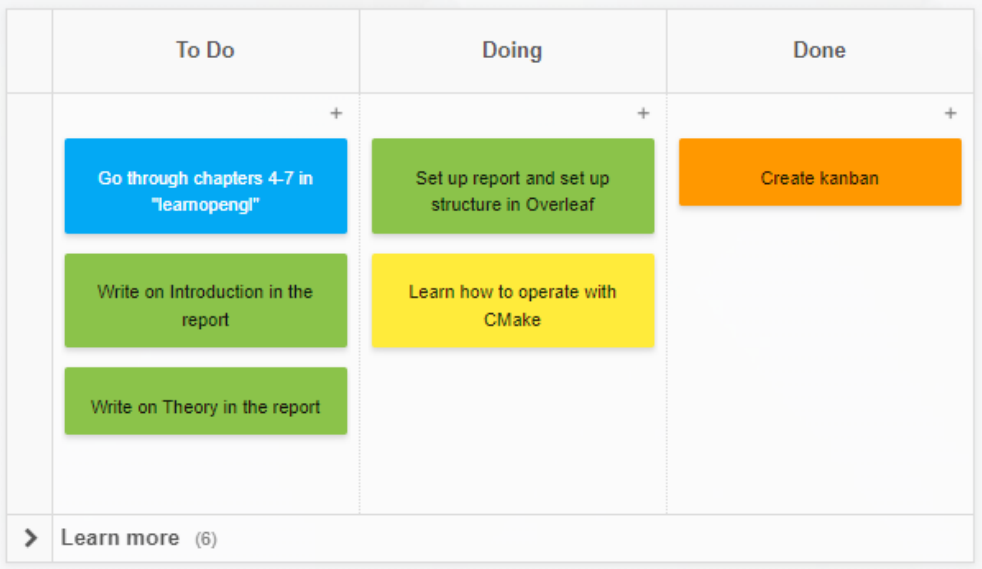


Figure 6.5: Kanban board from the start of sprint 4

Intermission 4: After sprint 4, we were using GLFW windows to create and display simple 2D models, manipulating their properties by using shaders. Concepts like graphics programming take time to comprehend and even longer to understand. However, this sprint gave us sufficient insight to be able to create the first iteration of our class diagram, to be further improved and refined as the project went on.

To start displaying a model in our window, we first had to define it. We did this by defining normalized coordinate values for the different points. Since our windows view is defined between -1.0 and 1.0, anything outside these values would be clipped (not shown). To define our first model, we added an array of vertices, points in normalized space, that would be used to build our model.

```
float vertices[] = {
     -1.0f, 0.0f, 0.0f,     //left edge
     -0.25f, 0.25f, 0.0f,   //top right edge
     -0.25f, -0.25f, 0.0f,  //bottom left edge
      0.0f, 1.0f, 0.0f,     //top edge
      0.25f, 0.25f, 0.0f,   // top right edge
      1.0f, 0.0f, 0.0f,     //right edge
      0.25, -0.25, 0.0f,    // bottom right edge
      0.0f, -1.0f, 0.0f,    // bottom edge
  };
```

Code listing 6.6: Vertex positions

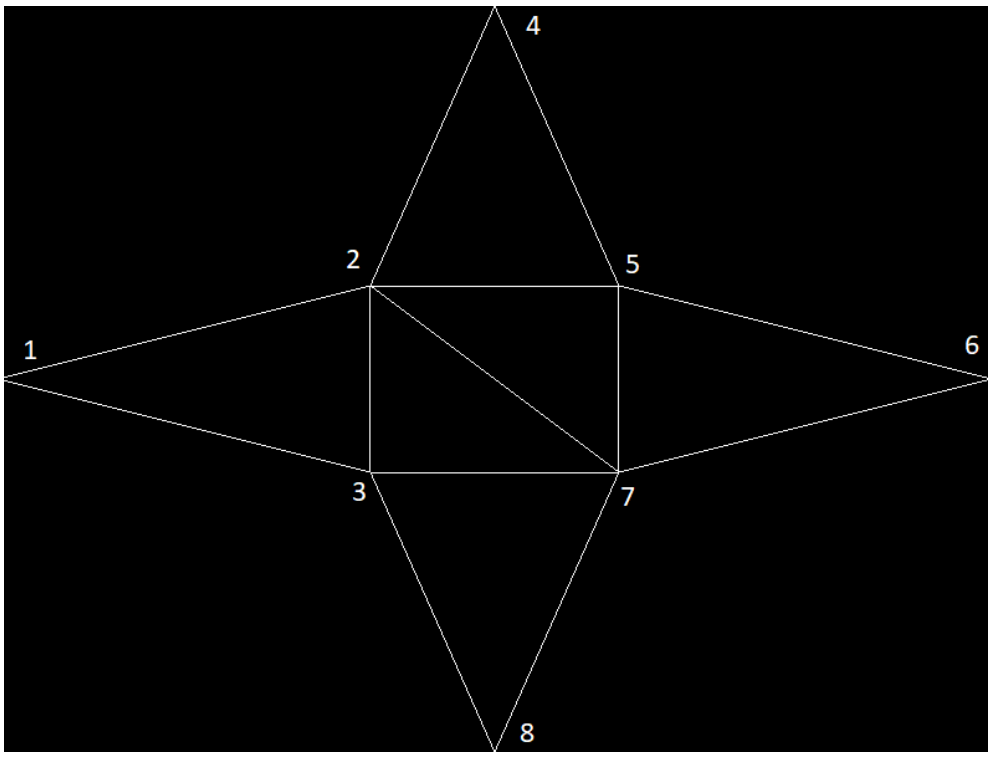


Figure 6.6: Star visualization

In this case we are making a star made up of triangles, and all the points defined here are the corners of the different triangles.

```
unsigned int indices[] = {
    0,1,2,   //left
    1,3,4,   //top
    4,5,6,   //right
    6,7,2,   //bottom
    1,2,6,   //mid bottom
    1,4,6    //mid top
  };
```

Code listing 6.7: Setting indicies

Now we can set the indicies, based on the vertices. The indicies are the points that make up the edges of our triangles, so all we do here is define which points make up which triangles.

```
unsigned int VBO, VAO, EBO;
  glGenVertexArrays(1, &VAO);
  glGenBuffers(1, &VBO);
  glGenBuffers(1, &EBO);
```

Now we define a Virtual Buffer Object (VBO), a Vertex Array Object (VAO) and an Element Buffer Object (EBO). The virtual buffer object is used to store information in the GPU‘s memory. Having the data stored in the GPU makes it fast to access later. We generate a new buffer by passing *glGenBuffer* the buffer's index and a reference to the empty buffer object we created. Vertex array objects are required by OpenGL to interpret vertex inputs, and we generate one by calling the *glGenVertexArrays* function.

When drawing triangles, we will not always have to specify three new vertices. Sometimes the same vertex is used for multiple triangles. Avoiding duplications like this is essential for larger models built by thousands of triangles. The EBO helps us avoid duplication by storing indices defined by unique vertices. We generate an EBO the same way as an VBO.

```
glBindVertexArray(VAO);

  glBindBuffer(GL_ARRAY_BUFFER, VBO);
  glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
```

Code listing 6.8: Binding and transfering data to buffers

We start by binding the VAO by calling *glBindVertexArray*, now all calls to other buffer objects will be stored in the bound VAO. Now we call the *glBindBuffer* function and bind the VBO to the GL_ARRAY_BUFFER type. After binding the VBO we call *glBufferData* to copy the vertex data in to it‘s memory. This function takes the buffer type, how many bytes to be stored, the predefined data, and which

draw function to use. GL_STATIC_DRAW is used here, but GL_DYNAMIC_DRAW should be used if the vertex positions are not static. After the VBO is bound and configured, we do the same to the EBO. The difference here is that the EBO is bound to the GL_ELEMENT_ARRAY_BUFFER, which stores index data.

Before we start rendering models with OpenGL we need to introduce shaders. First we will introduce the vertex shader, which is a small program that takes a vertex position as input. Shaders have their own language, OpenGl Shading Language (GLSL), which is very similar to C++.

```
1 const char *vertexShaderSource = "#version 330 core\n"
2 "layout (location = 0) in vec3 position;\n"
3 "void main()\n"
4 "{\n"
5 "   gl_Position = vec4(position.x, position.y, position.z, 1.0);\n"
6 "}\0";
```

Code listing 6.9: Vertex shader code

As of now, the vertex shader is stored as a C string in our main file, and we start by defining which OpenGL version we are using. Here we are using OpenGL version 3.3(330), and as set previously (Code listing 6.3, line 7), we are using the OpenGL core profile. Secondly we define the shaders input as a three-dimensional vector "position" and specify that it is found at location = 0. Inside the main body, we use *gl_Position* to set the output of our vertex shader. This function is by default a four-dimensional vector; therefore we get the x, y, and z values from our input vector and set them accordingly. The last attribute is used for transformations and normalizing coordinates, but we will not be using that yet, so we set it to 1.0.

```
1 const char *fragmentShaderSource = "#version 330 core\n"
2 "out vec4 FragColor;\n"
3 "void main()\n"
4 "{\n"
5 "   FragColor = vec4(1.0f, 1.0f, 1.0f, 1.0f);\n"
6 "}\n\0";
```

Code listing 6.10: Fragment shader code

We now introduce a fragment shader, this shader is responsible for calculating the color values of each pixel. We set its output as a four-dimensional vector called FragColor, defined in the main body. The three first variables in this vector define our models color; in this case it is white.
Now that we have defined what our shaders should do, we can implement their functionality.

```
unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
glCompileShader(vertexShader);
// check for and print compilation errors
int success;
char infoLog[512];
glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
if (!success)
{
  glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
  std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog << std::
   endl;
}
```

Using the *glCreateShader* function, we create a new shader and specify that it should be a vertex shader. The first argument of the *glShaderSource* function defines where the source code should be compiled to. After telling it to compile into our newly created shader object, we have to specify how many strings are in our shader's source code; we only have one in this state. In the third argument, we reference the source code itself, and the fourth specifies the lengths of the different input strings. The last argument we can set to null since we only pass the function one string.

Finally, we can compile the shader by calling *glCompileShader* on our vertex shader object. After doing this, we have to check for compilation errors; this is done by calling *glGetShaderiv*. This function allows us to query the shader for different information. In our case, we query for GL_COMPILE_STATUS, define an integer to indicate successful compilation and pass it as the third argument. If the compilation has errors, we want to print them, so we have to define a space to print the error messages. We do this by defining an array of characters which we pass to *glGetShaderInfoLog*, and print the errors.

```
  unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
```

Code listing 6.11: Creating a fragment shader

The fragment shader is created and compiled the same way as the vertex shader, but this time we specify to *glCreateShader* that we want to create a fragment shader.

```
  unsigned int shaderProgram = glCreateProgram();
glAttachShader(shaderProgram, vertexShader);
glAttachShader(shaderProgram, fragmentShader);
glLinkProgram(shaderProgram);
```

Code listing 6.12: Shader program

Since we already created and compiled our shaders, we now have to create a shader program that we link them to. The shader program is created by calling

glCreateProgram, we then attach both our shaders with *glAttachShader*. Finally, we link our shaders to our program; this is done by the *glLinkProgram* function. After our shaders are linked, the output of the first shader in the graphics pipeline is used as input for the next one. In our case, the output of our vertex shader is now our fragment shader input. As a last check, we implement an error check similar to the ones used on our shaders, but this time to check for linking errors. Once our shaders are correctly linked to our shader program, we can delete the shader objects.

```
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
  glEnableVertexAttribArray(0);
```

Even though of shader program is complete, we still have to tell our shaders how our data is structured and how it is supposed to be read. The *glVertexAttribPointer* function does exactly this; it takes a number of arguments we have to define. The first is which attribute we want to configure, and we previously defined our attribute location as 0 in the vertex shader source code. The second argument specifies the number of components per vertex attribute; since we previously defined this as a three-dimensional vector, it takes three values. The third argument specifies the data type, and the fourth attribute is for normalizing data. Since our data is already normalized, we leave the fourth argument as false. The next argument is the stride, meaning the space between the attributes in bytes. Since each of our vertices consists of three float values, the stride is set to 3**sizeof(float)* or 3*4=12 bytes. The last component is for offset; this is if the data you are reading does not start at index = 0 in the array. In our case, it does, but the argument needs to be of type void, so we have to cast it to 0. Now we can finally call *glEnableVertexAttribArray* by passing it our attribute location.

```
    glUseProgram(shaderProgram);
    glBindVertexArray(VAO);
    glDrawElements(GL_TRIANGLES, 18, GL_UNSIGNED_INT, 0);
```

Code listing 6.13: Drawing our shape

Everything is set up and ready to draw our first shapes. Inside the render loop, we first call *glUseProgram*; this makes sure the rest of the function calls use our shaders. We once again bind our vertex array object, so the drawing function knows where to look and call *glDrawElements*. To draw our model using elements, we first need to specify that we want OpenGL to draw triangles. Secondly, we need to specify the number of elements to draw; in our case, we have a total of six triangles, all built from three points. The last two arguments specify which type of values are in our indices and the offset of these.

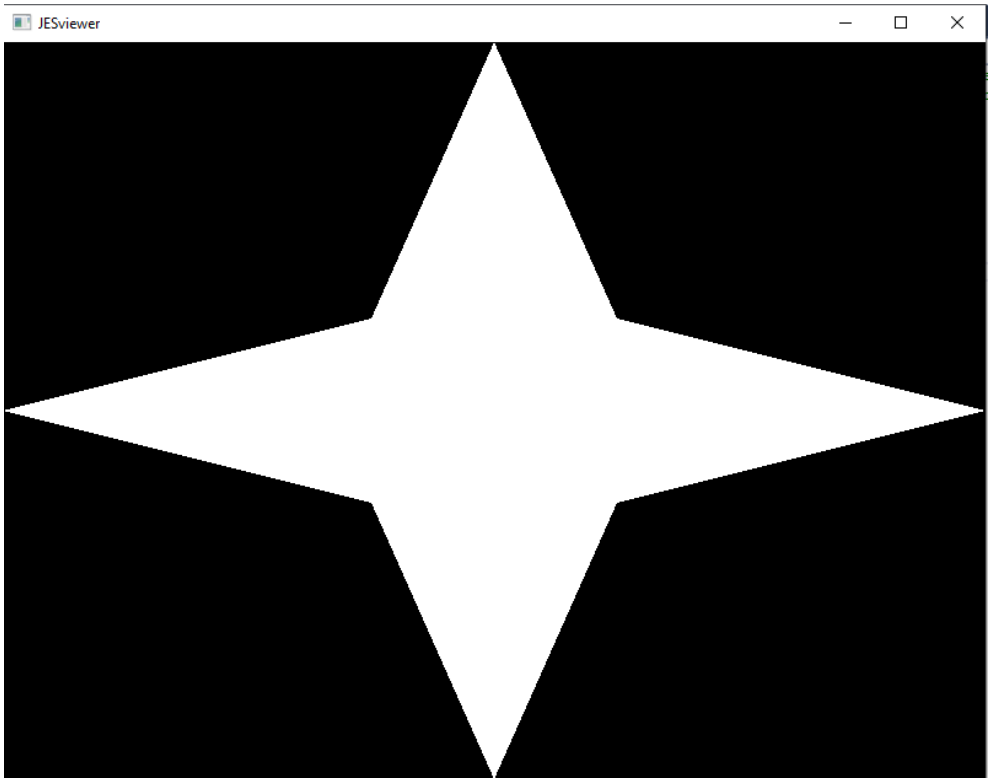


Figure 6.7: Rendered star

6.2.5 Sprint 5

These weeks went to improving our class layout, loosening coupling and improving cohesion. Furthermore, we continued work on the OpenGL curriculum we set for ourselves. Throughout these chapters we created a shader class that handles both the vertex and fragment shaders, which allowed us to manipulate different visual aspects in our 3D model rendering more efficiently. We introduced different matrix operations, such as transformations for rotating models either by static variables or dynamic functions.

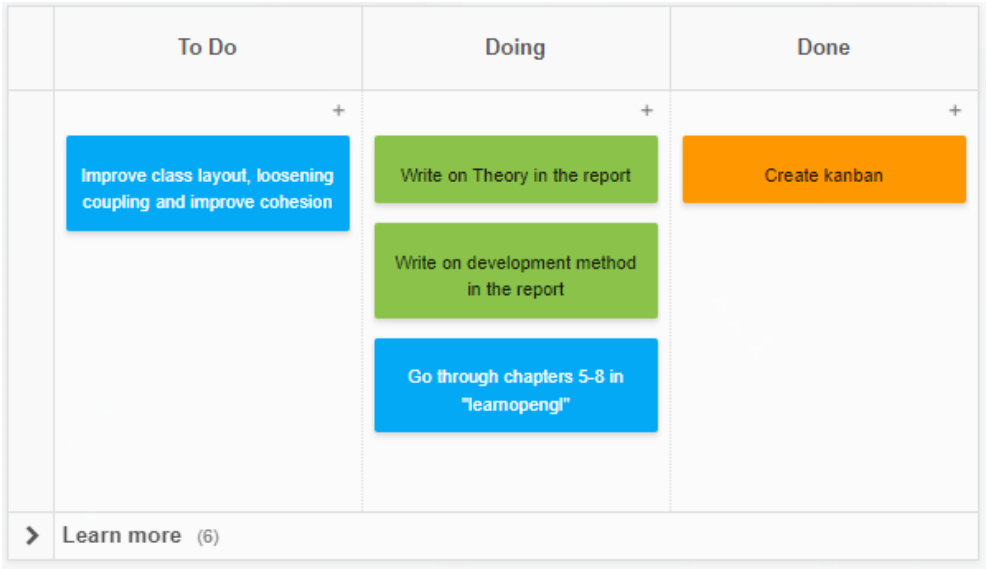


Figure 6.8: Kanban board from the start of sprint 5

Intermission 5: As we previously defined our two shaders on top of our main file, it leaves room for improvement. In the current state we have to compile them individually and manually check for compilation errors for every shader. If we introduce more shader variations this process will quickly become tedious. Therefore we want to create our own shader that reads our source files, compiles and links them to a shader program.

```
    #ifndef SHADER_H
#define SHADER_H
#endif
```

Code listing 6.14: Definition prefix

We start by creating a new header file, starting with the preprocessor directives *ifndef* and *define*. This makes sure our shader is compiled and included only once, avoiding linking problems. The *endif* directive is inserted at the bottom of our class definition to decide where the *ifndef* directive is completed.

```
    public:
  unsigned int shaderID;
  // constructor generates shader by specified paths
  Shader(const char* vertexPath, const char* fragmentPath)
```

Code listing 6.15: Identifier

Next we assign our shader a unique identifier, introduced here as shaderID. Our constructor takes two arguments, the paths to our vertex and fragment shader code. Our shader class needs to take these paths, open the files, read, store, com-

pile and link them to a shader program.

```
    std::string vertexCode;
   std::string fragmentCode;
   std::ifstream vShaderFile;
   std::ifstream fShaderFile;

   vShaderFile.exceptions(std::ifstream::failbit | std::ifstream::badbit);
 fShaderFile.exceptions(std::ifstream::failbit | std::ifstream::badbit);
```

Code listing 6.16: Storage objects

To read the files we need somewhere to store them and a way to read them. We preemptively create a string and an input file stream (ifstream) object for each shader. The string object is meant to store their source code, while the stream objects will read them from the files. When using ifstreams it important to make sure exceptions can be thrown when needed, here we allow *failbit* and *badbit* exceptions. These exceptions are thrown if a file is unsuccessfully read, due to loss of stream integrity or a logic issue.

```
    try
   {
     // open files
     vShaderFile.open(vertexPath);
     fShaderFile.open(fragmentPath);
     std::stringstream vShaderStream, fShaderStream;
     // read file from buffer
     vShaderStream << vShaderFile.rdbuf();
     fShaderStream << fShaderFile.rdbuf();
     // close file
     vShaderFile.close();
     fShaderFile.close();
     // convert stream into string
     vertexCode = vShaderStream.str();
     fragmentCode = fShaderStream.str();

   }
   catch (std::ifstream::failure& readError)
   {
     std::cout << "ERROR::SHADER::FILE_NOT_SUCCESFULLY_READ: " << readError.what()
     << std::endl;
   }
```

Code listing 6.17: Reading files and storing data as strings

We start by trying to call the *open* function on our stream objects, opening the files and associating them with the corresponding stream objects. After they are associated we define the streams as stringstreams, and call the *rdbuf* function to read the data from the buffer to the streams. Now that the data is read we can close the files. Finally we call the *str* function to copy the stream data into our initial string objects. We end with a catch statement, printing which exception was

thrown if the files were not read properly.

```
        const char* vShaderCode = vertexCode.c_str();
    const char * fShaderCode = fragmentCode.c_str();
        //compile shaders
    unsigned int vs, fs;
        // vertex shader
    vs = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vs, 1, &vShaderCode, NULL);
    glCompileShader(vs);
        // fragment Shader
    fs = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fs, 1, &fShaderCode, NULL);
    glCompileShader(fs);
```

Since *glShaderSource* takes needs its source code as a string of characters, we use the *c_str* function to set a pointer to an immutable character array containing it. Compiling the shaders is done the same way as before, and with the same arguments.

```
        // shader Program
    shaderID = glCreateProgram();
    glAttachShader(shaderID, vs);
    glAttachShader(shaderID, fs);
    glLinkProgram(shaderID);
    // delete shaders after link
    glDeleteShader(vs);
    glDeleteShader(fs);
```

Code listing 6.18: Creating and binding our shader program

The compiled shaders are linked to a new program defined by shaderID and the shaders are deleted after they are linked.

```
      private:
  //function for checking shader compilation/linking errors.
  void checkCompileErrors(GLuint shader, std::string type)
  {
    GLint success;
    GLchar infoLog[1024];
    if (type != "PROGRAM")
    {
      glGetShaderiv(shader, GL_COMPILE_STATUS, &success);
      if (!success)
      {
        glGetShaderInfoLog(shader, 1024, NULL, infoLog);
        std::cout << "ERROR::SHADER_COMPILATION_ERROR of type: " << type << "\n" <<
     infoLog << std::endl;
      }
    }
    else
    {
      glGetProgramiv(shader, GL_LINK_STATUS, &success);
      if (!success)
      {
        glGetProgramInfoLog(shader, 1024, NULL, infoLog);
        std::cout << "ERROR::PROGRAM_LINKING_ERROR of type: " << type << "\n" <<
    infoLog << std::endl;
      }
    }
  }
```

Code listing 6.19: Function for printing error info

We used to have two seperate functions for printing error messages in the initial setup, here we group them all into one so we can call the same function to check both compilation and linking errors. This function takes a shader object and a type as input, then contains an if statement to separate the different queries. This is mostly the same process as the previous iteration, only this version of more generalized and can be called with a simple oneliner. The infolog is also twice as large, allowing more information to be returned.

```
      checkCompileErrors(vs, "VERTEX");
      checkCompileErrors(fs, "FRAGMENT");
      checkCompileErrors(shaderID, "PROGRAM");
```

Code listing 6.20: CheckCompileError calls

The new function is inserted after each compilation, so we can easily spot where any errors occur.
Now that we have a working shader program, created by our shader class we need to specify the input and outputs of our vertex and fragment shader.

```
        #version 330 core
layout (location = 0) in vec3 position;

out vec4 vertexColor;

void main()
{
  gl_Position = vec4(position.x, position.y, position.z, 1.0);
  vertexColor = vec4(0.0, 1.0, 0.0, 1.0);
}
```

Code listing 6.21: New vertex shader

```
        #version 330 core
out vec4 FragColor;

in vec4 vertexColor;
void main()
{
  FragColor = vertexColor;
}
```

Code listing 6.22: New fragment shader

This version of the vertex shader differs from our last iteration, in that it specifies its output as vec4 (a floating point vector with four components). This is now set as the input for our fragments shader. The vertex shader passes a color value, in this case green, to the fragment shader. Which in turn gives the rendered model its color.

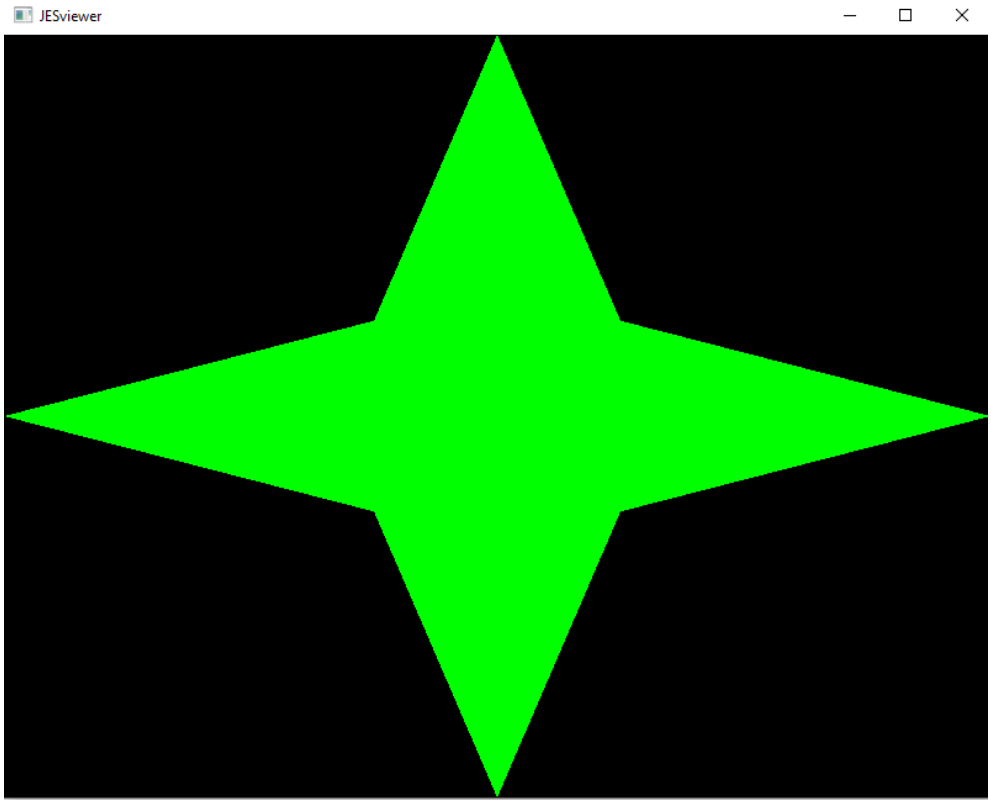


Figure 6.9: Star rendered with color values from shader communication

Defining the color data in the shaders themselves is not always a good idea. Different models can have different data attached, a mesh model might have a texture or a voxel model might have material data attached. In this case we will add some color data to the vertex array, and extract it from our shaders.

```
1       float vertices[] = {
2     -1.0f, 0.0f, 0.0f,    1.0f, 0.0f, 0.0f,     //left edge
3     -0.25f, 0.25f, 0.0f,  0.0f, 1.0f, 0.0f,   //top right edge
4     -0.25f, -0.25f, 0.0f, 0.0f, 0.0f, 1.0f,  //bottom left edge
5      0.0f, 1.0f, 0.0f,    0.0f, 1.0f, 0.0f,  //top edge
6      0.25f, 0.25f, 0.0f,  1.0f, 0.0f, 0.0f,    // top right edge
7      1.0f, 0.0f, 0.0f,    0.0f, 1.0f, 0.0f,    //right edge
8      0.25, -0.25, 0.0f,   0.0f, 0.0f, 1.0f,     // bottom right edge
9      0.0f, -1.0f, 0.0f,   0.0f, 1.0f, 0.0f,    // bottom edge
10  };
```

The data is now structured so that the first three floats define vertex positions, and the next three define the vertex color attribute. This sequence repeats for all vertices.

```
//Vertex shader

       #version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 colorData;

out vec3 vertexColorFromData;

void main()
{
  gl_Position = vec4(position.x, position.y, position.z, 1.0);
  vertexColorFromData = colordata;
}

//Fragment shader

#version 330 core
out vec4 FragmentColor;

in vec3 vertexColorFromData;
void main()
{
  FragmentColor = vec4(vertexColorFromData, 1.0);
}
```

Code listing 6.23: Shader adaptation to find color data

The process is much like we did when specifying positional data, we create a new layout. This time we say the layout has location = 1, and give it a name to signify it defines color. We pass the fragment shader the color data with a vec3 output, and adapt it to a vec4 so the drawing function will accept it.

```
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
    sizeof(float)));
  glEnableVertexAttribArray(1);
```

Code listing 6.24: Specifying how the data should be read

We have to specify to OpenGL how the data should be read. Earlier we set a vertex attribute pointer for layout = 0, in this case we want to specify for layout = 1. Some arguments have different parameters than previous iterations. The stride needs to be changed to 6*4 bytes, since the next value of the same type is stored 6 positions (32 bytes)later in the array. For location =1 (colorData), we also specify an offset, saying that it should skip the first three instances and start reading from vertices at index 3.

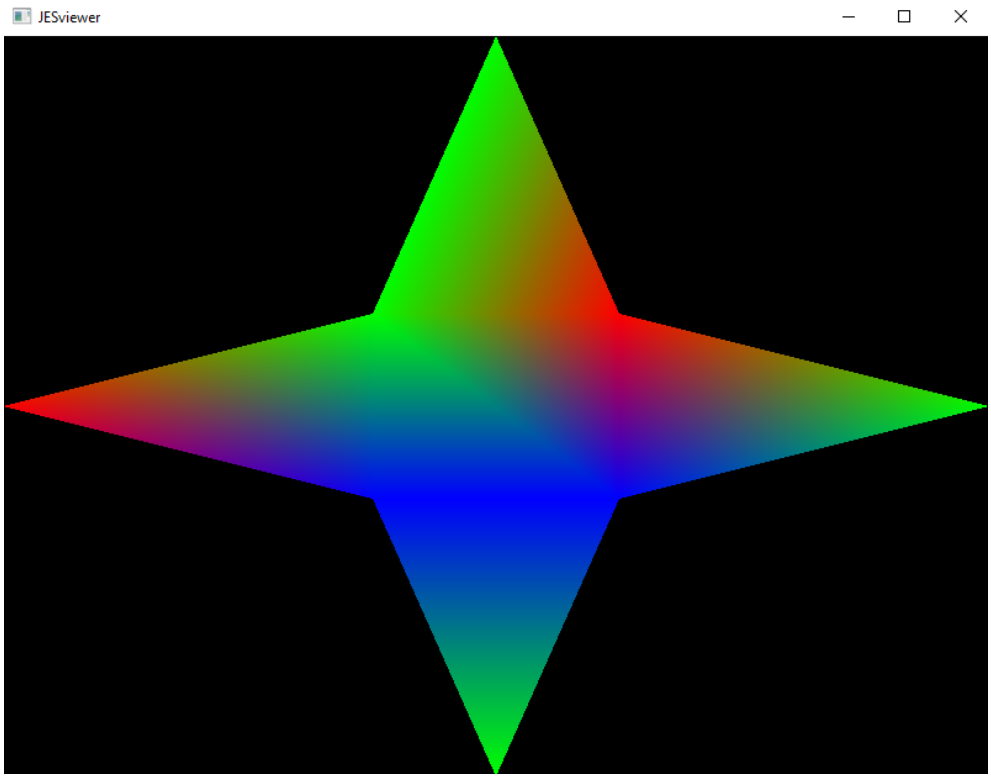


Figure 6.10: Star with fragment interpolation

The result is not what we first expected. The fragment shader creates way more fragments then the ones we specified in our indices array. It then extrapolates the color values based on its proximity to the specified color fragments. So if a fragment is directly between a green and a red instance, the pixels color output is 50% green and 50% red, resulting in an orange pixel.

A model viewer is not complete without some kind of interaction with either perspective, position or both. As our current iteration only renders static 2D objects, we needed a way to change the positions of the rendered triangles. To do this we used the GLM library to access matrix operations that makes it possible to change the position of the vertices without manual input.

```
1 #include <glm/glm.hpp>
2 #include <glm/gtc/matrix_transform.hpp>
3 #include <glm/gtc/type_ptr.hpp>
```

Code listing 6.25: Glm includes

To start using GLM functions, we need to include the proper header files. There are many different header files in the GLM library, we only include the ones we need.

```
#version 330 core
layout (location = 0) in vec3 position;
layout (location = 1) in vec3 colorData;

out vec3 vertexColorFromData;

uniform mat4 transformation;

void main()
{
  gl_Position = transformation * vec4(position, 1.0);
  vertexColorFromData = colorData;
}
```

Code listing 6.26: Transformation data passed to vertex shader

We want to start implementing rotation by using matrix transformations. The first thing we need to do is make sure our vertex shader is connected to the rotation matrix. To do this we create a uniform of type mat4. We multiply this matrix with our vertex position before passing it to *gl_Position*. The type mat4 represents a 4x4 matrix, which when multiplied with the position will allow us to rotate our model.

```
glm::mat4 transform = glm::mat4(1.0f);
    transform = glm::rotate(transform, (float)glfwGetTime(), glm::vec3(0.0f, 1.0f,
    0.0f));
```

Code listing 6.27: Defining our rotation

Inside our render loop we initialize a new 4x4 matrix, and set it as an identity matrix. The argument we pass to *glm::mat4* specifies the diagonal values, and an identity matrix is a 4x4 matrix where all the diagonals have a value of 1. Next we call the *glm::rotate* function on our new matrix. The rotate function takes a few argument, we need to specify which matrix to preform our rotation on, the angle of rotation and its direction. Here we set the angle by calling the *glfwGetTime* function, which gives the time since GLFW was initiated. Lastly we want to rotate around the y-axis, to give us our first impression of 3D-space.

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta)x + \sin(\theta)z \\ y \\ -\sin(\theta)x + \cos(\theta)z \\ 1 \end{bmatrix}$$

Figure 6.11: Calculating rotation around y-axis

The above figure shows the math behind the rotation sequence. A 4x4 matrix multiplied with a vertex position giving a rotated output. In our case the angle() is given by time since program initiation, while the time value rises the angle a vertex rotates increases linearly for each frame. It can clearly be seen in the result of the multiplication that the y-value stays the same while the other positions change.

```
1        unsigned int transformLoc = glGetUniformLocation(newShader.shaderID, "
    transformation");
2    glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(transform));
```

This final part is done after activating our shader program, first we ask our shader for the location of our uniform by calling *glGetUniformLocation* and passing it our shaders id and the name of our uniform. Secondly we set the value of our uniform with *glUniformMatrix4fv*, passing it the uniform location, its count, transposition and a pointer to the transformation matrix we set earlier. Since we did this inside the render loop, the rotation will update the vertex positions continuously for as long as the program is running. This means the object will be spinning around the y-axis.

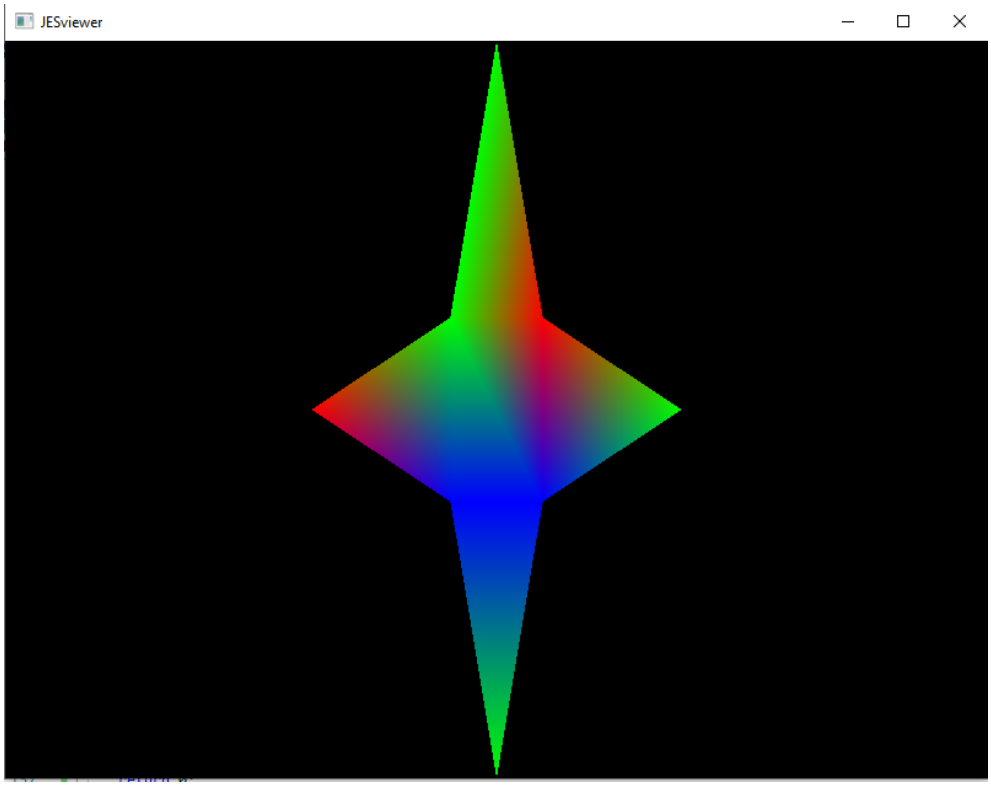


Figure 6.12: Star rotating

Figure 6.12 shows the star halfway through its rotation. This gives the im-

pression of 3D-space. However, since our model has no thickness, it is more like a section of a plane rotating.

6.2.6 Sprint 6

This late in the development stage we redistributed responsibility over different project areas. Development was started on a GUI, focus shifted to more documentation and report work while development continued on the OpenGL codebase. The goal of this sprint was to implement our first 3D model and a working camera system. These seemed to work perfectly fine, but are still applied exclusively to locally created models.

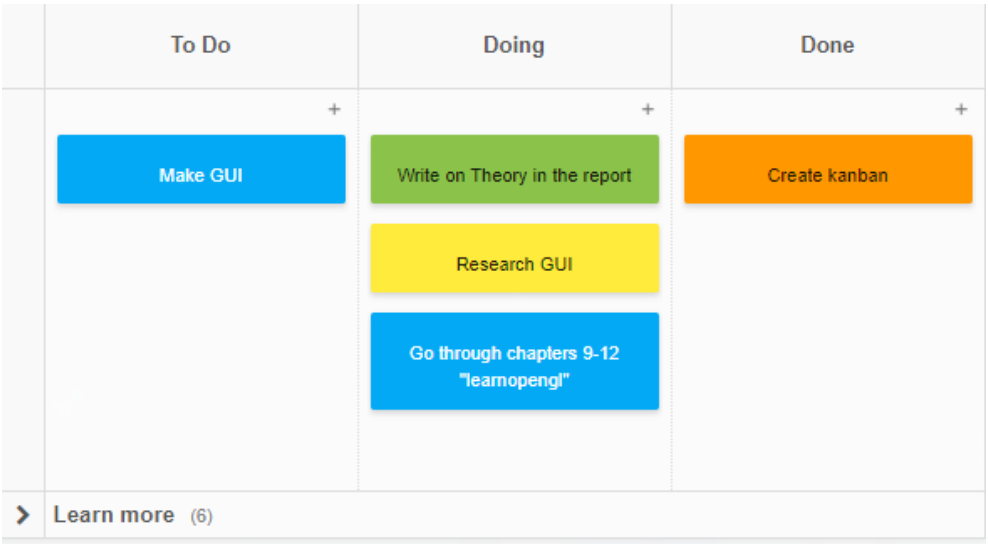


Figure 6.13: Kanban board from the start of sprint 6

Intermission 6: This sprint was dedicated to get camera functionality up and running, meaning we could play around with perspectives, rotations and translations in a more efficient manner. Naturally the first thing we will need is an actual 3D-model. Since we still lack import functions for external models, we created our own.

```
1
2     float vertices[] = {
3      //3D star
4     -1.0f, 0.0f, 0.0f,  0.0f, 0.0f, 1.0f,//left edge
5     -0.25f, 0.25f, 0.0f, 1.0f, 0.0f, 0.0f,//top left edge
6     -0.25f, -0.25f, 0.0f, 1.0f, 0.0f, 0.0f,//bottom left edge
7      0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f,//top edge
8      0.25f, 0.25f, 0.0f, 1.0f, 0.0f, 0.0f,// top right edge
9      1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f,//right edge
10     0.25, -0.25, 0.0f, 1.0f, 0.0f, 0.0f,// bottom right edge
11     0.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f,// bottom edge
12       0.0f, 0.0f, 0.2f, 0.0f, 0.0f, 1.0f, //front :
13     0.0f, 0.0f, -0.2f, 1.0f, 0.0f, 0.0f//back :
14   };
15   unsigned int indices[] = {
16     //triangles front
17     0,8,1,
18     0,8,2,
19     3,8,1,
20     3,8,4,
21     4,8,5,
22     6,8,5,
23     6,8,7,
24     2,8,7,
25
26     //triangles back :
27     0,9,1,
28     0,9,2,
29     3,9,1,
30     3,9,4,
31     4,9,5,
32     6,9,5,
33     6,9,7,
34     2,9,7,
35   };
```

Code listing 6.28: Simple 3D expansion of our star

Extending our star into three dimensions is relatively straight forward, add two points at the origin (center) and offset them along the z-axis on both sides. Then we decide which triangles to draw in the indices array. An approach like this is very input heavy, but for now it works just fine.

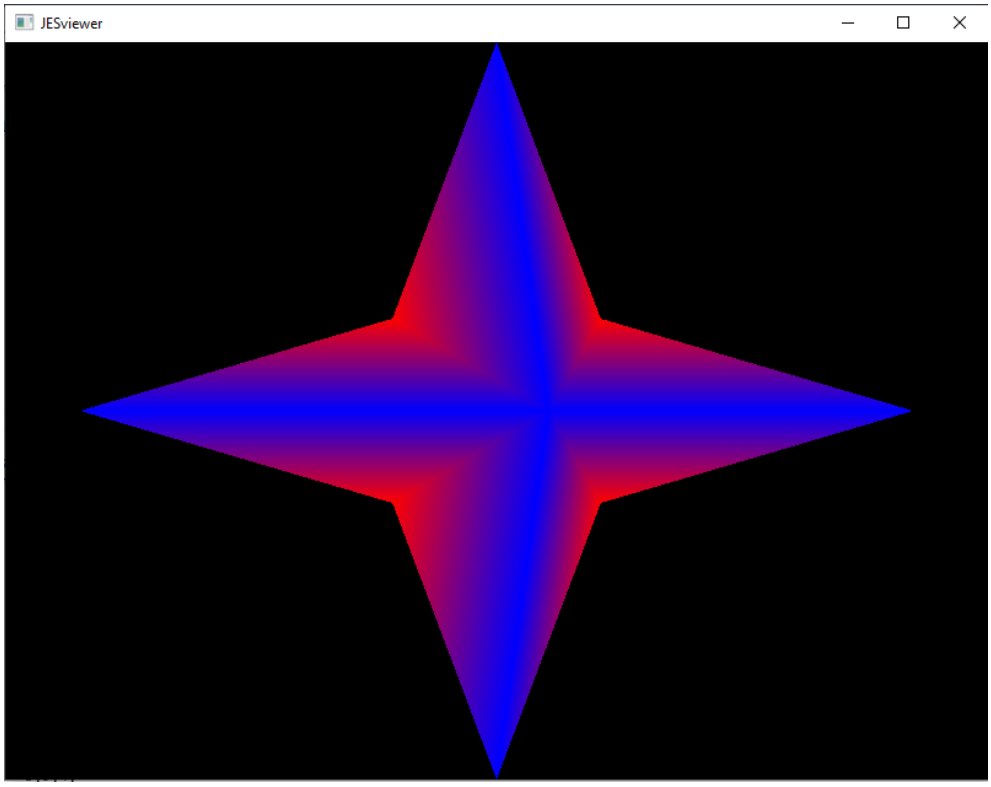


Figure 6.14: 3D star

Now that our star finally is a real 3D model, we need to implement some features for inspecting it. To do this we want to implement a camera system, that allows us to view a model from any angle. We start by creating a new camera class.

```
        #ifndef CAMERA_H
#define CAMERA_H

#include <glad/glad.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

#include <vector>

// Which directions the camera can move
enum Camera_Movement {
  FORWARD,
  BACKWARD,
  LEFT,
  RIGHT
};

const float SENSITIVITY = 0.2f;
const float SPEED = 4.0f;
const float ZOOM = 45.0f;
const float YAW = -90.0f;
const float PITCH = 0.0f;
```

Code listing 6.29: Camera preprocessor directives and includes

We start by defining some essential enumerations, consisting of the base movement features of our camera. Next we set up some starting variables for camera control. We decide how aggressively the mouse reacts to input, the movement speed of our camera and to which degree the field of view can be affected by zooming. The default yaw value is set to -90 degrees, this is because if it was set to 0 the camera would start by looking at the positive x-axis. Rotating it by -90degrees starts the camera view towards the negative z-axis, which is exactly what we want.

```
        class Camera
{
public:

  // camera Attributes
  glm::vec3 Position;
  glm::vec3 Front;
  glm::vec3 Up;
  glm::vec3 Right;
  glm::vec3 WorldUp;

  // euler Angles
  float Yaw;
  float Pitch;

  // camera options
  float MovementSpeed;
  float MouseSensitivity;
  float Zoom;

  // constructor
  Camera(glm::vec3 position = glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3 up = glm::vec3
     (0.0f, 1.0f, 0.0f), float yaw = YAW, float pitch = PITCH)
  {
    Front = glm::vec3(0.0f, 0.0f, -1.0f);
    MovementSpeed = SPEED,
    MouseSensitivity = SENSITIVITY,
    Zoom = ZOOM;
    Position = position;
    WorldUp = up;
    Yaw = yaw;
    Pitch = pitch;
    updateCameraVectors();
  }
```

Code listing 6.30: Attributes and constructor

The first thing we need to do is define which attributes the camera needs. It needs some vectors to define both movement and direction. The position vector gives the camera its current position in world space and the front vector tells the camera which direction it is looking. There are two up-vectors, the first is the cameras local "up" which changes depending on where the camera is looking, the second is the global "up". It also needs a right or a left vector, but not both as a left vector is the same as a -right vector.

We need some float values for the yaw and the pitch. These represent rotation around the y and the x-axis, more easily visualized as if the camera is looking diagonally up/down or left/right. Our constructor instantiates the camera, and sets it to its starting values. We set our camera to start at the global origin and looking down the negative z-axis.

```
1       glm::mat4 GetViewMatrix()
2   {
3     return glm::lookAt(Position, Position + Front, Up);
4   }
```

Code listing 6.31: View matrix function

The *GetViewMatrix* function returns the cameras view matrix. The view matrix is the rendered space visible by our camera, we find it by calling *glm::lookAt*. This function takes a position as its first argument. Secondly it takes a direction, in our case this is the position plus our front vector, this way we make sure we are always looking forwards. The up vector is used to calculate our right vector.

```
1   void ProcessKeyboard(Camera_Movement direction, float deltaTime)
2   {
3     float velocity = MovementSpeed * deltaTime;
4     if (direction == FORWARD)
5       Position += Front * velocity;
6     if (direction == BACKWARD)
7       Position -= Front * velocity;
8     if (direction == LEFT)
9       Position -= Right * velocity;
10    if (direction == RIGHT)
11      Position += Right * velocity;
12  }
```

Code listing 6.32: Keyboard input processing

We want our camera to be able to move around our scene, here we define the output of different button presses. This function takes one of our predefined enums as input and the time since the last frame. We specify which changes in position we want for the different movement directions with a series of if-statements.

```
void ProcessMouseMovement(float xoffset, float yoffset, GLboolean constrainPitch
  = true)
{
  xoffset *= MouseSensitivity;
  yoffset *= MouseSensitivity;

  Yaw += xoffset;
  Pitch += yoffset;

  // Statement to stop pitching > 90 degrees
  if (constrainPitch)
  {
    if (Pitch > 89.0f)
      Pitch = 89.0f;
    if (Pitch < -89.0f)
      Pitch = -89.0f;
  }

  // update vectors
  updateCameraVectors();
}
```

Code listing 6.33: Mouse movement processing

We want the input from our mouse to decide where the camera is looking and we specify this by changing the pitch and yaw values. The changes in our pitch and yaw values scale with our preset sensitivity, if not the response would be way to quick. If the pitch value goes to or above +/-90 degrees the screen will flip, causing unwanted behaviour. This can be avoided by constraining the pitch to never go above 89 degrees. Finally we call the *updateCameraVectors* function to adjust the vectors according to the new pitch and yaw values.

```
  void updateCameraVectors()
{
  //new front vector
  glm::vec3 front;
  front.x = cos(glm::radians(Yaw)) * cos(glm::radians(Pitch));
  front.y = sin(glm::radians(Pitch));
  front.z = sin(glm::radians(Yaw)) * cos(glm::radians(Pitch));
  Front = glm::normalize(front);

  // Handle up and right vector
  Right = glm::normalize(glm::cross(Front, WorldUp));

  Up = glm::normalize(glm::cross(Right, Front));
}
```

Seeing as the camera is able to look in all directions, we need a function for updating its vectors as their values change. The *updateCameraVectors* function does just this. It instantiates an empty vector, and assigns it the adjusted values based

on the new yaw and pitch values. The values are calculated by using formulas derived from *Euler angles*, for instance the x component of our front vector is the product of cos(pitch) and cos(yaw). Our right vector is the cross product of our new front and the global up vector, and the new local up vector a cross of our new right and front vectors.

```
  void ProcessMouseScroll(float yoffset)
  {
    Zoom -= (float)yoffset;
    if (Zoom < 1.0f)
      Zoom = 1.0f;
    if (Zoom > 45.0f)
      Zoom = 45.0f;
  }
```

We also want to be able to zoom in our view, to get a better look at model details. The *ProcessMouseScroll* function handles scroll input, based on our field of view. Our global field of view value is set at 45, so we implement a constraint so we are not able to zoom out more than where we started. The camera class is now completed and we can start to utilize it.

```
    void mouse_callback(GLFWwindow* window, double xposIn, double yposIn)
{
  float xpos = static_cast<float>(xposIn);
  float ypos = static_cast<float>(yposIn);

  if (firstMouse)
  {
    lastX = xpos;
    lastY = ypos;
    firstMouse = false;
  }

  float xoffset = xpos - lastX;
  float yoffset = lastY - ypos; // reversed since y-coordinates go from bottom to
     top

  lastX = xpos;
  lastY = ypos;

  camera.ProcessMouseMovement(xoffset, yoffset);
}

// glfw: whenever the mouse scroll wheel scrolls, this callback is called
// -------------------------------------------------------------------
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset)
{
  camera.ProcessMouseScroll(static_cast<float>(yoffset));
}
```

Code listing 6.34: Callback functions for mouse inputs

The *mouse_callback* function takes the last position of our mouse on screen, subtracts the new position and stores the change in position. It then sets the new po-

sition as the last position so the function avoids stalling. The function concludes by calling the *ProcessMouseMovement* function from our camera class, passing it the offsets as arguments. There is also a check to see if this is the first time the mouse enters the screen, this is to avoid jagged movement when the mouse is first triggered. We also have a function for reacting to scrolling, this directly calls the *ProcessMouseScroll* function from the camera by passing it its y-offset.

```
    void processInput(GLFWwindow *window)
{
  if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
    glfwSetWindowShouldClose(window, true);

  if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
    camera.ProcessKeyboard(FORWARD, deltaTime);
  if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
    camera.ProcessKeyboard(BACKWARD, deltaTime);
  if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
    camera.ProcessKeyboard(LEFT, deltaTime);
  if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
    camera.ProcessKeyboard(RIGHT, deltaTime);
}
```

Code listing 6.35: Key triggers

For movement we add additional triggers to our *processInput* function. They work for the WASD keys by passing *ProcessKeyboard* function the associated enumerations and the time since last frame.

```
    float currentFrame = static_cast<float>(glfwGetTime());
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;
```

Code listing 6.36: Delta frame

The time since last frame is calculated at the beginning of each render loop by subtracting the current frame from the last frame. The last frame is then reset.

```
    //inside render loop
    glm::mat4 projection = glm::perspective(glm::radians(camera.Zoom), (float)
     screen_width / (float)screen_height, 0.1f, 100.0f);
    glm::mat4 view = camera.GetViewMatrix();
    newShader.setMat4("projection", projection);
    newShader.setMat4("view", view);

    // render the loaded model
    glm::mat4 model = glm::mat4(1.0f);
    newShader.setMat4("model", model);

    //Position calculation in vertex shader
    gl_Position = projection * view * model* vec4(position, 1.0);
```

Code listing 6.37: Rendering our model

To render our model we now have to send the perspective to our shader for each frame. We call the *glm::perspective* function by passing it our current camera zoom in the field of view argument this time. We also send the current view matrix and the identity matrix to the vertex shader uniforms, as the vertex positions now are calculated by the product of its projection, view, model and position.

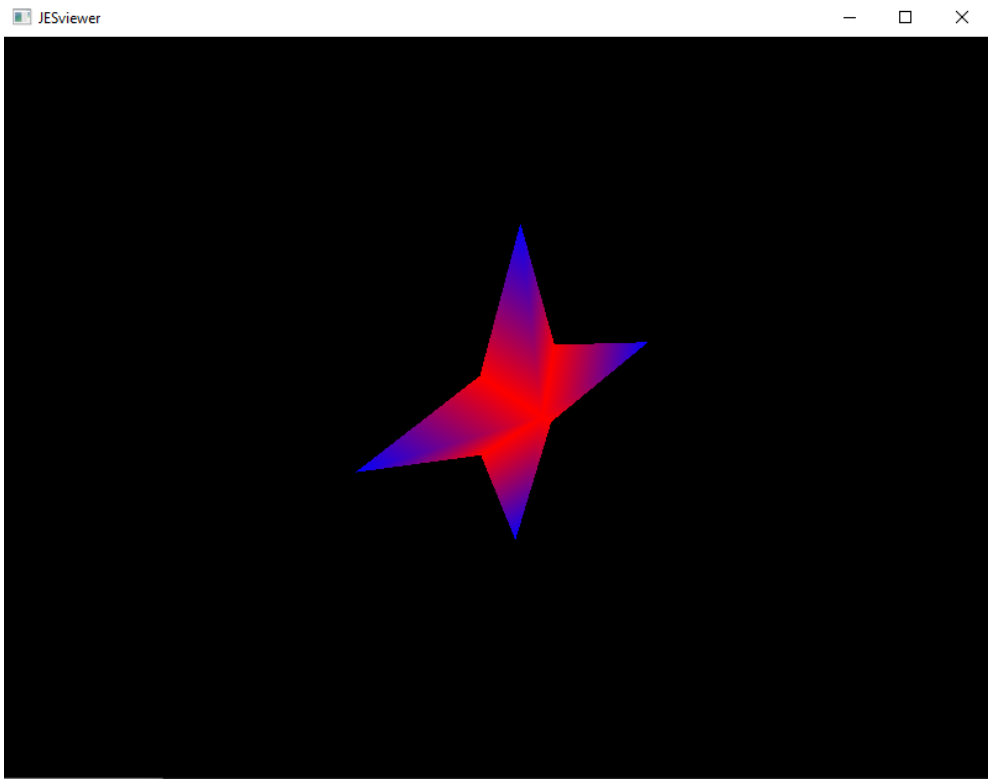


Figure 6.15: Star viewed with camera

Another part of this sprint involved building a GUI that would have the most essential features. Research found that the GUI-builder tool Qt, was a good choice for building GUI in C++.[19] Building a GUI in Qt's own IDE "Qt Creator" was a simple task, as its drag-and-drop functionality is very easy to use. However, we encountered some problems mid sprint, as we struggled with getting the Qt framework to work within CLion. This was due to a configuration error that happened, which is shown in Figure 6.17. The latest version of Qt, 6.2.4 was used at first, which proved to be difficult for integration because of lack of documentation within this version. Therefore, we proceeded with trying Qt5 instead, but the same error message kept popping up, even though we tried to follow JetBrains' own Qt setup tutorial. After researching the error message for several days with no progress, we decided to go away from the GUI implementation and focus on developing the other applications, which was of higher significance. Figure 6.16 shows the Qt Creator interface, which has drag-and-drop features.

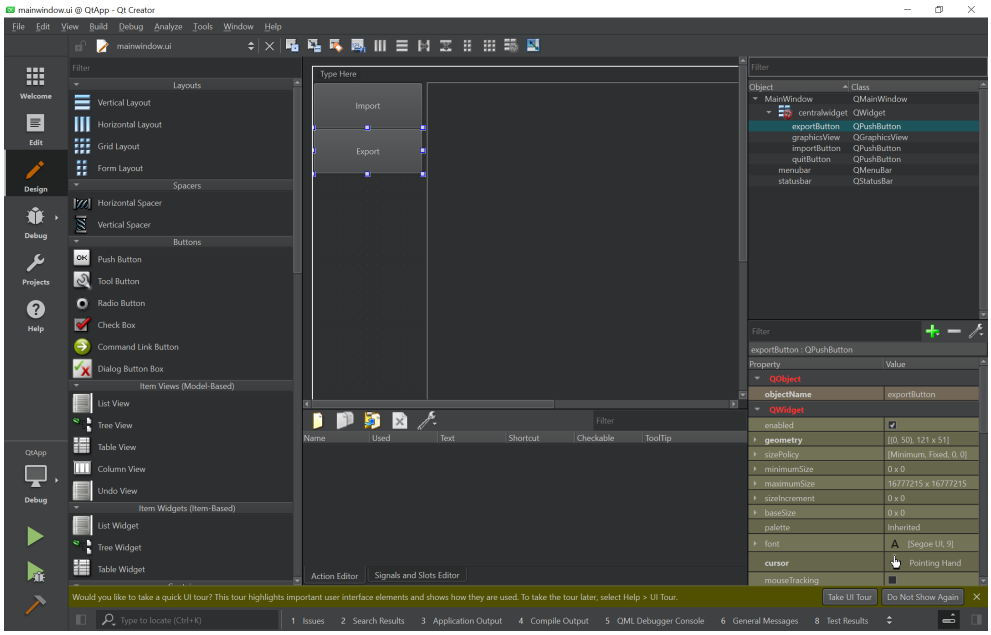


Figure 6.16: Qt Creator IDE

```
CMake Error at CMakeLists.txt:11 (find_package):
  Could not find a configuration file for package "Qt6" that is compatible
  with requested version "".

  The following configuration files were considered but not accepted:

    C:/Qt/6.2.4/mingw_64/lib/cmake/Qt6/Qt6Config.cmake, version: 6.2.4 (64bit)
    C:/Qt/6.2.4/msvc2019_64/lib/cmake/Qt6/Qt6Config.cmake, version: 6.2.4 (64bit)
```

Figure 6.17: Error message from Qt in CLion

6.2.7 Sprint 7

In this stage of the development we wanted to integrate Assimp with OpenGL, which would then later be used for retrieving structs that could take mesh as input . We also had to do learn how to port a Visual Studio project to CLion, which proved to be more difficult than anticipated.

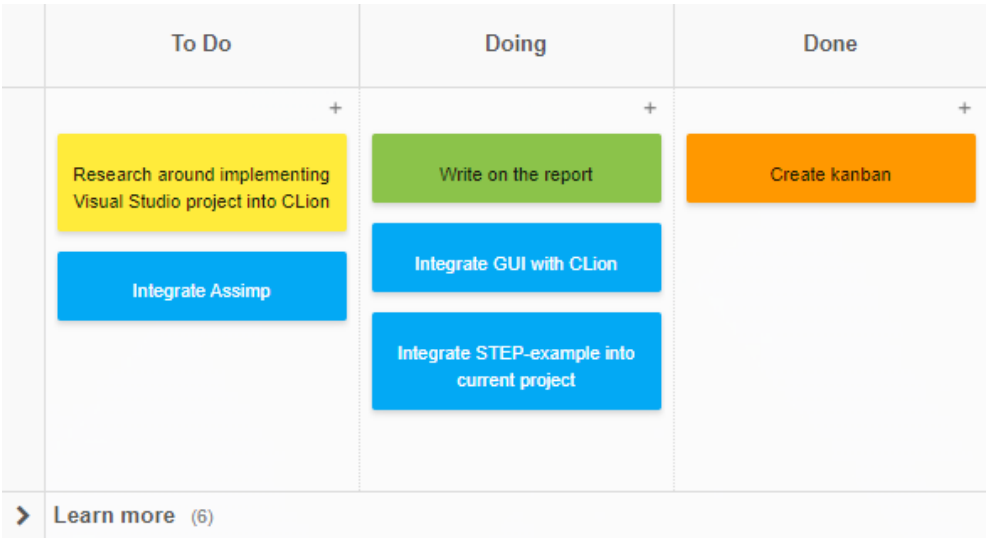


Figure 6.18: Kanban board from the start of sprint 7

Intermission 7: Our goal during this sprint was to render an external model using Assimp functionality. To do this we will need a mesh and an abstracted model class that can be expanded to import other data structures in the future. We start by implementing the mesh class.

```
struct Vertex {
  // attributes
  glm::vec3 Position;
  glm::vec3 Normal;
  glm::vec2 TexCoords;
};

struct Texture {
  unsigned int id;
  string type;
};
```

The struct Vertex holds all our vertex attributed. Since we are targeting external models now, we include some texture data since the models we found has one or more texture-maps accompanying them. We then store a texture-id and a type for each texture we load.

```
class Mesh {
public:
  // mesh Data
  vector<Vertex>       vertices;
  vector<unsigned int> indices;
  vector<Texture>      textures;
  unsigned int VAO;

  // constructor
  Mesh(vector<Vertex> vertices, vector<unsigned int> indices, vector<Texture>
     textures)
  {
    this->vertices = vertices;
    this->indices = indices;
    this->textures = textures;

    // set buffers and attribpointers
    setupMesh();
  }
```

Code listing 6.38: Data assignment and constructor

All necessary data is initiated and a new VAO is set up. The constructor passes the mesh object the required data, a vector of vertex structs, indicies and a vector containing texture data. It then calls a function for mesh data definition.

```
unsigned int VBO, EBO;

void setupMesh()
{
  glGenVertexArrays(1, &VAO);
  glGenBuffers(1, &VBO);
  glGenBuffers(1, &EBO);

  glBindVertexArray(VAO);
  // vertex buffer
  glBindBuffer(GL_ARRAY_BUFFER, VBO);
  glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0],
   GL_STATIC_DRAW);

  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int), &
   indices[0], GL_STATIC_DRAW);

  // vertex attribute pointers
  glEnableVertexAttribArray(0);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
  // vertex normals
  glEnableVertexAttribArray(1);
  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof
   (Vertex, Normal));
  // vertex texture coords
  glEnableVertexAttribArray(2);
  glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof
   (Vertex, TexCoords));

  glBindVertexArray(0);
}
```

This section is generalized in the mesh class now, so we can avoid doing this in our main file for every object. It works in much the same way, except this time we are also parsing texture coordinates.

```
    void Draw(Shader &shader)
  {
      //texture handling
    unsigned int diffuseNr = 1;
    unsigned int specularNr = 1;
    for (unsigned int i = 0; i < textures.size(); i++)
    {
      glActiveTexture(GL_TEXTURE0 + i);
      string number;
      string name = textures[i].type;
      if (name == "texture_diffuse")
        number = std::to_string(diffuseNr++);
      else if (name == "texture_specular")
        number = std::to_string(specularNr++);


      glUniform1i(glGetUniformLocation(shader.shaderID, (name + number).c_str()), i
     );
      glBindTexture(GL_TEXTURE_2D, textures[i].id);
    }

    // draw mesh
    glBindVertexArray(VAO);
    glDrawElements(GL_TRIANGLES, static_cast<unsigned int>(indices.size()),
     GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);

    glActiveTexture(GL_TEXTURE0);
```

Our drawing function now has to handle texture data. We start by checking how many textures we have of each type by iterating through our texture vector, calling *glActiveTexture* for each instance to prepare it for binding. Next we get the texture type as a string to match the vertex uniforms. Then we give our uniform the location of the active texture and bind it. The *glBindTexture* function binds a texture to a texture type, in this case to 2D textures. Finally we draw the mesh, but after we are done call *glActiveTexture* to set the the active texture to 0.

Large and complex models are often built by multiple meshes, which are later connected and rendered together. In order to render a model consisting of more then one mesh, we need to create a model class to load and render multiple meshes at the same time.

```
    unsigned int TextureFromFile(const char *path, const string &directory);

class Model
{
public:
  // model data
  vector<Texture> textures_loaded;
  vector<Mesh>    meshes;
  string directory;

  // constructor
  Model(string const &path)
  {
    loadModel(path);
  }
```

Code listing 6.39: Model constructor

We define the model data, all textures currently bound, all meshes and the directory in which they are found. The constructor takes a path to the .obj wavefront file as its argument and calls the *loadModel* function.

```
    {void loadModel(string const &path)
  {
    Assimp::Importer importer;
    const aiScene* scene = importer.ReadFile(path, aiProcess_Triangulate |
     aiProcess_FlipUVs);
    // check for errors
    if (!scene || scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE || !scene->mRootNode)
    {
      cout << "ERROR::ASSIMP:: " << importer.GetErrorString() << endl;
      return;
    }

    directory = path.substr(0, path.find_last_of('/'));

    processNode(scene->mRootNode, scene);
  }}
```

Code listing 6.40: Load model function

Our *loadModel* function creates an importer object from the Assimp library. We call the importers *ReadFile* function by passing it the path and calling several methods to process the data. First we call *Triangulate* which turns all non triangle shapes in the mesh into triangles. Secondly we call *FlipUVs* to flip the texture maps, since most texture maps are flipped by default.
Next we have a function to check for importer errors, that checks if the scene or rootnode are null and query one of its flags to check for incomplete data. Then we store the directory of the model and call the *processNode* function on our root node.

```
    void processNode(aiNode *node, const aiScene *scene)
{
  for (unsigned int i = 0; i < node->mNumMeshes; i++)
  {
    aiMesh* mesh = scene->mMeshes[node->mMeshes[i]];
    meshes.push_back(processMesh(mesh, scene));
  }

  for (unsigned int i = 0; i < node->mNumChildren; i++)
  {
    processNode(node->mChildren[i], scene);
  }
}
```

Code listing 6.41: Recursive function for processing nodes

To process all the nodes in our meshes we first have to iterate through them. ProcessNode takes a root node and a assimpScene as arguments. We check each of the nodes indicies and store their meshes in the mMeshes array. We then call the processMesh function on the mesh. Finally we do the same for all the nodes children until the function stops.

```
      Mesh processMesh(aiMesh *mesh, const aiScene *scene)
{
  // data
  vector<Vertex> vertices;
  vector<unsigned int> indices;
  vector<Texture> textures;

  //iterate vertices
  for (unsigned int i = 0; i < mesh->mNumVertices; i++)
  {
    Vertex vertex;
    glm::vec3 vector;
    // positions
    vector.x = mesh->mVertices[i].x;
    vector.y = mesh->mVertices[i].y;
    vector.z = mesh->mVertices[i].z;
    vertex.Position = vector;
    // normals
    if (mesh->HasNormals())
    {
      vector.x = mesh->mNormals[i].x;
      vector.y = mesh->mNormals[i].y;
      vector.z = mesh->mNormals[i].z;
      vertex.Normal = vector;
    }
    // texture coordinates
    if (mesh->mTextureCoords[0])
    {
      glm::vec2 vec;
              //store textureData
      vec.x = mesh->mTextureCoords[0][i].x;
      vec.y = mesh->mTextureCoords[0][i].y;
```

```
33          vertex.TexCoords = vec;
34        }
35        else
36          vertex.TexCoords = glm::vec2(0.0f, 0.0f);
37
38        vertices.push_back(vertex);
39      }
40      // check faces
41      for (unsigned int i = 0; i < mesh->mNumFaces; i++)
42      {
43        aiFace face = mesh->mFaces[i];
44        for (unsigned int j = 0; j < face.mNumIndices; j++)
45          indices.push_back(face.mIndices[j]);
46      }
47
48      aiMaterial* material = scene->mMaterials[mesh->mMaterialIndex];
49
50      //  diffuse
51      vector<Texture> diffuseMaps = loadMaterialTextures(material,
        aiTextureType_DIFFUSE, "texture_diffuse");
52      textures.insert(textures.end(), diffuseMaps.begin(), diffuseMaps.end());
53      //  specular
54      vector<Texture> specularMaps = loadMaterialTextures(material,
        aiTextureType_SPECULAR, "texture_specular");
55      textures.insert(textures.end(), specularMaps.begin(), specularMaps.end());
56
57
58      // return a mesh object
59      return Mesh(vertices, indices, textures);
60    }
```

Since we have extracted the meshes it is time to process them. The function starts with iterating through the vertex and storing the position data. It then checks the normal data for each vertex, a normal decides which way a vertex is facing. Knowing which side of a face is up or down is essential when layering textures. Next it checks for texture data and stores all the data in the vertices vector. After the vertex data is stored, we retrieve all the faces and store their index values in our indices vector. The next step is querying for material data, as each mesh only uses a single material. We can get this material data with *mMaterialIndex*. Finally we look for textures, here we look for a specific naming convention, which is not the most reliable. We store our textures as different types, in this case diffuse and specular textures. The only thing left is to return a Mesh object, passing it the vertices, indices and textures we recovered.

```
        vector<Texture> loadMaterialTextures(aiMaterial *mat, aiTextureType type,
    string typeName)
{
  vector<Texture> textures;
  for (unsigned int i = 0; i < mat->GetTextureCount(type); i++)
  {
    aiString str;
    mat->GetTexture(type, i, &str);
    // check if texture is loaded
    bool skip = false;
    for (unsigned int j = 0; j < textures_loaded.size(); j++)
    {
      if (std::strcmp(textures_loaded[j].path.data(), str.C_Str()) == 0)
      {
        textures.push_back(textures_loaded[j]);
        skip = true;
        break;
      }
    }
    if (!skip)
    {
      Texture texture;
      texture.id = TextureFromFile(str.C_Str(), this->directory);
      texture.type = typeName;
      texture.path = str.C_Str();
      textures.push_back(texture);
      textures_loaded.push_back(texture);
    }
  }
  return textures;
}
```

Code listing 6.42: Loading textures from materials

The *loadMaterialTextures* function loads material textures that are not already loaded. Queries the material data for the texture type, checks if it already is loaded in our texture vector, if not it calls the *TextureFromFile* function. This function loads the texture and returns us its id, we set its type and path before we add it to our texture vector. Finally we add it to textures_loaded so it doesnt try to load the same texture twice. The function returns the new textures vector.

```
        unsigned int TextureFromFile(const char *path, const string &directory,
    bool gamma)
{
  string filename = string(path);
  filename = directory + '/' + filename;

  unsigned int textureID;
  glGenTextures(1, &textureID);

  int width, height, nrComponents;
  unsigned char *data = stbi_load(filename.c_str(), &width, &height, &nrComponents,
      0);
  if (data)
  {
    GLenum format;
    if (nrComponents == 1)
      format = GL_RED;
    else if (nrComponents == 3)
      format = GL_RGB;
    else if (nrComponents == 4)
      format = GL_RGBA;

    glBindTexture(GL_TEXTURE_2D, textureID);;

    stbi_image_free(data);
  }
  else
  {
    std::cout << "Texture failed to load at path: " << path << std::endl;
    stbi_image_free(data);
  }

  return textureID;
}
```

Finally we have the TextureFromFile function, this function reads a texture file and returns a texture. It takes the textures path and directory as its arguments, and stores these as a string. The function uses stbi$_{l}oad to load the file data, it needs the path to our texture, awi$
$Now our Model class is ready for duty, all we need to do is adapt our drawing to work with our new class.$

```
        //outside render loop
        Model newModel("Models/backpack.obj");
        //Render loop
        newModel.Draw(newShader);
```

We create a new model object, and give it a path to a file we know is compatible. In theory this should work, but there are issues with the Assimp dynamic link library. We have tried to link it both with manual CMake input, and more recently rebuilt our codebase in Visual Studio 16 by using the CMake GUI and integrated linking systems. This seems to work for all but assimp.dll, and we have not solved this problem yet.
This is, however, not the same error message as we got when we tried building

```
1>------ Build started: Project: JESviewer, Configuration: Debug x64 ------
1>main.cpp
1>C:\Users\Student\source\repos\JESviewer\JESviewer\dlls\assimp-vc142-mtd.dll : fatal error LNK1107: invalid or corrupt file: cannot read at 0x378
1>Done building project "JESviewer.vcxproj" -- FAILED.
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

Figure 6.19: Visual Studio error

with precompiled binaries in CLion. So this might just be a local compilation issue that is easily fixed.

The EDMsdk example we retrieved currently generates a database with the AP209 schema, with additional validation checks, such as; if a database does not exist, create a database. The example also generates a model and a data repository, which is given the same validation check as the database.

The example we received was developed in Visual Studio, while our application was made in CLion, which made the configuration between the two applications more difficult than needed. We started of by opening the STEP example we received in CLion, which resulted in multiple errors.

After being unsuccessful with CLion, we decided to use the software that was originally used for developing the example, Visual Studio. Here we managed to run the program after doing some minor adjustments regarding the libraries, since libraries within Visual Studio is imported differently.

```
> Schema already defined
> Repository opened
> Model exists
> Model deleted
> Model created
> Model opened
> Nodes created
> Elements created
> Completed

C:\Users\sande\OneDrive\Dokumenter\Bachelor\Jotne\edmcore-v602.210.93_with_example_20220424\edmcore-v602.210.93_with_exampl
e_rela\Edmi_cpp_interface_sample\x64\Debug\Edmi_cpp_interface_sample.exe (process 11456) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console
when debugging stops.
Press any key to close this window . . .
```

Figure 6.20: Output from the EDMsdk program

The last thing we did was look into the file structure of the CT-scanned parts we were supposed to convert. We found that the original scanned files were 20-30% larger then the .raw files we were working with. They are stored as voxel elements with accompanied density values ranging from 0.0->22000. These density values represent relative density, with 0 being the lowest density in the sensors field of view and 22000 being the maximum.

The difference in size is most likely due to clipping of the model area, to remove voxels with close to zero in density values. The scale of the matrixes was also much smaller as the original was 1807x1858x2051 voxels and the .raw file 1104x1031x1414 with a voxel size of 0.00278mm*3. The data was structured by their density values, as an array of 2bit unsigned integers. We had planned to create a voxel class to read this format, but do to our problems with the original

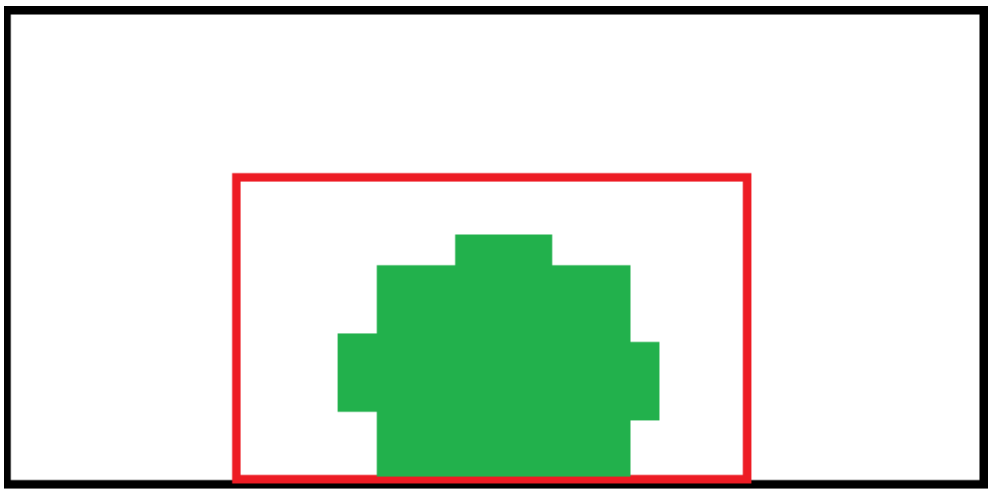


Figure 6.21: Voxel scan data

viewer this was not done in time.

6.2.8 Sprint 8

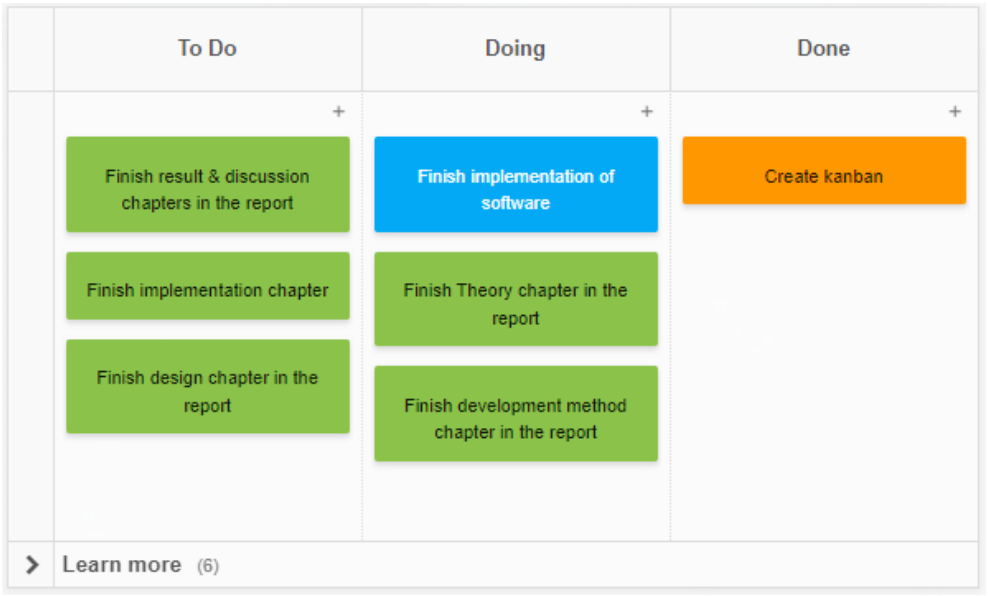


Figure 6.22: Kanban board from the start of sprint 8

In the final sprint, all implementation and developing was temporarily stopped, we had to focus on writing the final report. This was due to the short time we had left before the deadline. We discussed with Jotne and explained that we had to prioritize the report for now, and agreed that we would continue development

after the project deadline.

Intermission 8: The result of sprint 8 is the finished report.

Chapter 7

Results

In this section we will cover the final results of the project. Throughout this period we have been working on solutions regarding the model viewer, STEP-converter and the GUI.

7.1 Class Structure

The code structure contains six different classes where each one of them serve a unique purpose. These classes work as intended, however issues caused by model path definitions keeps the project from compiling properly. The project consists of struct variables that only takes textured mesh models as input, with camera functionality that allows the model to be inspected visually from all angles. An abstract model class takes different model types as input and feeds it to the renderer. Initially, the plan was to take mesh and voxel models as input, but due to time constraints caused by compilation issues, we had to focus on only implementing the mesh clash, followed by a definition of what the voxel class should look like.

The mesh class handles mesh structure(s), which enables import and viewing of mesh based models. Visual texture and color data is layered directly on the model inside the render loop. We accomplished this by implementing the shader class, supported by a fragment and vertex shader. The vertex shader takes predefined color layouts and texture maps, and outputs a 2D vector to the fragment shader. Then, the fragment shader interprets this information and converts it to a 4D vector based on several preset variables. The output from the fragment shader tells the shader class what to overlay on triangular faces by utilizing the GPU. Since the model class divides all non-triangular faces into triangles, this works perfectly.

7.2 EDMsdk

We were provided a software development kit by Jotne, called EDMsdk. The EDMsdk is capable of writing STEP files based on different predefined inputs utilizing the AP209 schema. Although it works on predefined, locally created and simplified models, it requires a deepened and generalized voxel class to function properly for more complex models. Since we have not implemented such a class, it currently does not work as intended.

7.3 GUI

The GUI is functioning in Qt's own IDE, Qt Creator, but was never implemented in the main project due to difficulties getting it to work with CLion. The current GUI has the most essential functionalities, such as import and export button. It also has a graphics viewer to display the imported 3D model.

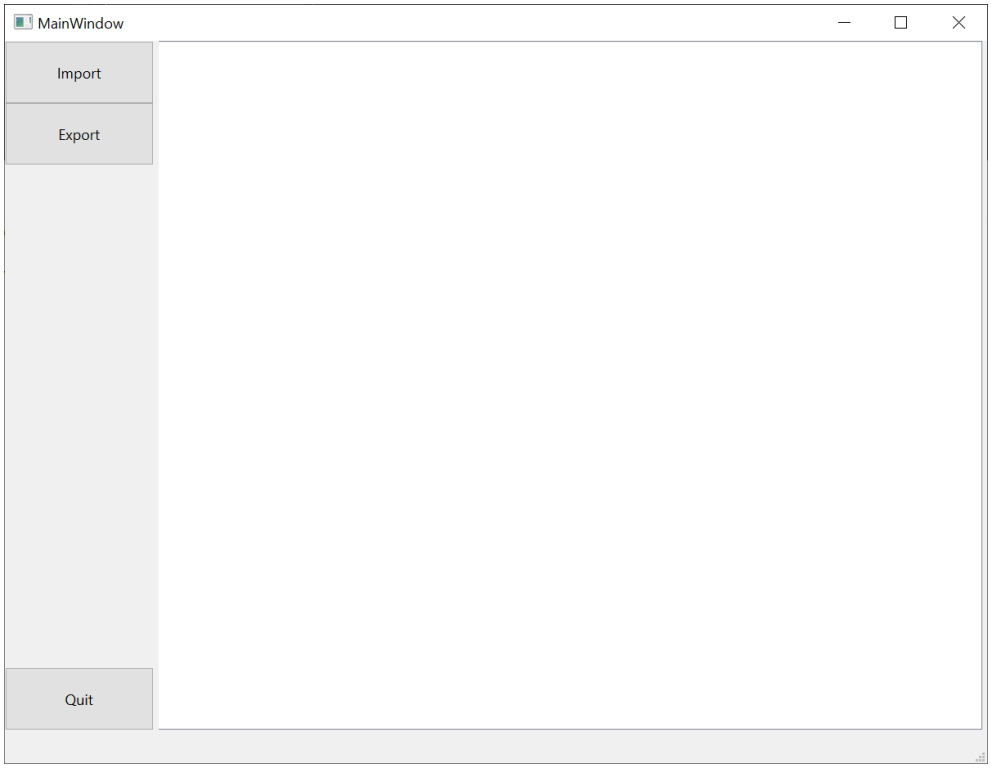


Figure 7.1: Screenshot of the current GUI

The last part consists of integrating the CLion project with both the EDMsdk and the Qt GUI. We did not manage to integrate the EDMsdk with the CLion project do to library linking errors and configuration issues. Therefore we ported the project to Visual Studio 2017 were the EDMsdk worked properly. Since time was

taken from GUI development to prioritize getting the CLion project to compile, GUI is neither completed nor integrated.

The EDMsdk-example that we process as of now is running and working, but as mentioned above we are still missing a deepened and generalized voxel class for further implementation, such as model properties, values and materials that naturally contains within a CT scanned object. We chose Visual Studio to develop in, since this is the software where EDMsdk was originally developed; this is to prevent any further integration errors than necessary.

As of now, the EDMsdk example can create an EDM database, generate an EDM model, establish nodes, create model elements and write out a model in STEP format.

Chapter 8

Discussion

8.1 Development Method & Process

This section will go through the different choices taken throughout the development process and reflect upon the results.

At the beginning of the project, we had to choose a development strategy that would be used throughout the development process. We went for a combination of two agile development methodologies, primarily due to our initially vague defined problem statement, which resulted in multiple alterations within the original plan. A combination of both kanban and Scrum seemed like a good starting point. These options complement each other as kanban boards are an excellent solution for getting an overview of the tasks we are currently working in; while the sprints were used as a more detailed description of what had to be done, followed by an intermission with in-depth explanation, where figures and code listings are included.

Using these development strategies did not work as planned throughout the development phase. We set specific deadlines for milestones that we expected to complete without always being able to, as in figure 6.5, where the task involved completing the chapters 4-7 in "learnopenGL"; but as you can see in the kanban board from sprint 5, figure 6.8, we only managed to complete 1 chapter throughout that sprint. This was due to difficulties we encountered within the configuration , causing our external libraries to malfunction. We mostly used pre-compiled binaries, while in the future, we would compile the libraries ourselves, ensuring that the libraries are compatible with our system.

During the times we encountered technical issues, we should have asked for assistance more often instead of wasting valuable time on minor issues. We tried reaching out to the supervisor and the employees whom we had close contact with, but they had limited knowledge within the field. If we had we been more proactive, we would be able to avoid time delays that were caused by linking and configuration errors.

Figuring out the most suitable development tool was difficult; since we had no prior experience working with 3D models, it was challenging to find the right software with the necessary tools. We decided on using CLion simply because we had experience with other JetBrains products. Our supervisor thought it made sense to write our program around the OpenGL API. We took his advice after not coming up with any better options. We had no prior experience using OpenGL, but since it was a C++ based API we had to first spend time learning C++ syntax. Another point to mention is that OpenGL had extensive documentation regarding the different functions available in the application, which made the learning curve manageable at the beginning.

Later in the project development, we started using more advanced functionality within OpenGL, which again required more libraries to make use of the modern functions that were related to 3D modeling. This is where the efficiency started to decline since we had to manually import all the libraries through CMake. This is a toolkit within CLion, where none of us had any experience.

When the OpenGL application was functional, we could finally point our attention towards implementing the application with the EDMsdk example that Remi Lanza provided. The only concern we had at this point was the deadline of the project since we received the EDMsdk example 22nd of April, which was already late in the project phase.

After looking into the EDMsdkexample, we felt overwhelmed, since we had no idea how to integrate this with the application made in OpenGL. First of all, the application was developed in Visual Studio, which is a different IDE than the one we are currently using. Still, we wanted to integrate the EDMsdk example with CLion. After struggling with the integration for several days, we tried migrating the code within Visual Studio instead. This was initially a safer approach since the EDMsdk-example was more complex than the application we had made, so running it within the IDE it was originally made in was a better option.

8.2 Future Development

In this section, we will discuss the possibilities for future development.

At the moment, we have not developed a complete product ready for use. We have three separate applications that have not been integrated yet, and that are still missing some adjustments within the code. As of now, we only have a OpenGL viewer application that can take mesh-based models as input, but initially, we would like the application to take voxel models as input. The same goes for the STEP-converter that we have received; this application only handles mesh as well. We have discussed and come to an agreement with Jotne that we will keep on developing after the deadline to complete the software. Jotne has access to all of the code in the GitLab repository, which gives them the opportunity to review the

code and give some feedback regarding further development. Jotne understood that we had to prioritize the report and were supportive; they were also glad to hear that we were going to continue developing after the deadline.

8.3 Workflow

This section entails how the group has functioned together as a unit and how we combined work outside of studies and other subjects that we had this semester.

In general, the process has gone in waves. There are some weeks when the development comes to a halt. This is partly because two group members had to work outside of their studies. They had a plan to work more at the start of the semester to work less later. The same members also had a subject where the exam in one of the subjects dated early, while the other dated later in the semester. This resulted in us working in a pair of two throughout a couple of weeks. Also, The development got affected by the Covid-19 breakout within the group, which set another dampener on the development for another three weeks in total. After the covid breakout, two of the members struggled to recover entirely and kept getting ill throughout the project. The project suffered too much from these events since we did not achieve the number of hours we sought to reach with all three present.

Risk analysis defined in section 5.3 in the project plan (appendix C) shows that we had foreseen this event.

8.4 Project Goals

In this section, we will discuss the goals we set at the start of the project, how it turned out and how it could have been done differently.

Initially, the plan was to develop a model viewer to render mesh and voxel data. Additionally, we needed an algorithm to manipulate this data, so it complied with the ISO 10303 standards. We were supposed to develop the STEP-converter from scratch, implementing struct variables that would suit the properties of the models. After researching this approach, we figured that creating a model viewer for mesh and voxel models was too comprehensive. We approached Jotne and talked with them, explaining the situation, and came up with a new approach.

This approach entailed developing a STEP-converter for the mesh model, excluding the voxel model from the equation. We rapidly began researching and figured that there are already available solutions online, such as the "freeCAD converter." We received this information from the lead developer on the Assimp library, who had substantial experience within this field; he also mentioned that "step is a massive format beast." We relayed this piece of information to our employer, explaining the situation, and agreed to change the project's direction again.

In this new approach, we were expected to first develop a viewer for voxel models, which would display relevant information, such as material properties and metadata. This would be accompanied by a STEP-converter that would be implemented later in the development phase. The voxel viewer proved to be more difficult than anticipated. A considerable amount of time went into creating just a basic model viewer with the necessary struct variables that supports the voxel format. This was due to configuration errors within CMake, where we could not efficiently import libraries. After struggling with this side of the task, we also figured that creating a STEP-converter from scratch was unrealistic since we would have to go through a huge amount of information. The standard itself had a guidebook that contained complex information that we were unable to integrate into our project. This was a stressful moment since we realized we would not complete the product in time for the deadline. This resulted in a discussion with our supervisor where we explained the current situation.

After discussing the situation with our supervisor, he came up with two potential solutions. In the first option, he suggested that we write an assessment report, which basically means explaining multiple approaches we took within a problem area, describing what worked and what did not. We were skeptical of this approach since it would be a total do-over when it was initially supposed to be a development report, but we still considered it. The other option included implementing only the crucial parts of ISO 10303; this would narrow down the amount of information we would have to go through, so we were more optimistic about this approach. After some discussion within the group, we contacted Jotne and shared the options which our supervisor had suggested. They were supportive of both options but said that they would prefer the option where we implemented a small portion of the ISO 10303.

Further on in the meeting, they told us that we could use a portion of the EDMsdk system they are using. We were told that one of the employees at Jotne would develop an example that would work as a STEP-converter. This made the development of the STEP-converter a lot more manageable since now we would not have to create the STEP-converter from scratch, which felt like an impossible task. After this turn of events, we had a clear problem area that we had discussed thoroughly and were satisfied with.

8.5 Learning Goals

This section will cover everything according to the project plan about what we expected to learn and what we actually learned.

Prior to the start of the development phase, we made a list of several skill sets that we wanted to develop. The first goal stated the ability to execute a long-term project, which we feel that we have accomplished. Even though we were not able to develop a complete product, we still followed the process throughout

the whole project period. This includes everything from weekly status meetings, using development methodologies, making a project plan, project management and logging time.

The second goal in the project plan states that we should expect to acquire knowledge about CT scan data and how to use it correctly. This is something we accomplished partially. We managed to interpret the data correctly and figure out how it was structured, but we did not have time to implement the voxel class to utilize it.

The third section in the project plan points out the ability to gain further knowledge of different approaches and techniques we can use to implement into our project. At the start of the development phase, we made sure to do a lot of research ahead of the actual development stage. This was to ensure that we approached the task with the right set of tools, avoiding any unnecessary headaches.

The fourth section suggests getting insight into how the client operates inside their own work environment and how they work together. This section was not correlated with the work in the project, but this was a way to gain more knowledge regarding how they operate within their workstation and how they execute long-term projects. This proved to be very educational. Learning goals are located in the project plan (Appendix C), section 1.4.

Throughout this project, we have acquired a good amount of exposure within multiple topics and fields. Some of these topics include 3D modeling, whereas we researched voxel models, mesh models, and how they are structured and built. In conjunction with 3D-modelling, CT scanning was also a field that we gained extensive knowledge within; We had to figure out how the CT scanner works and how it interprets the output that we received from the scanned objects. All this was important for further development, making sure that we had extensive knowledge about each topic that we were expected to start developing within.

Further on, we learned about software configuration and file structuring. Before we could fully start developing the software, we had to learn about the process behind importing libraries and structuring folders. This was done in order to operate with CMake more efficiently. After this section, we were closing in on the development stage of the product, where we had to acquire some experience with C++ beforehand since this was a language we were not that familiar with. This was accomplished by going through basic tutorials and getting used to the different syntax.

After this step, we started with graphical coding, using OpenGL inside CLion. Here we learned quite a lot about the functionality that OpenGL had to offer, and how 2D and 3D models are made inside the application. After doing research regarding OpenGL and getting used to the software, we had to submerge ourselves into the STEP-example that we received from Jotne. In the example, we learned a lot about AP209 and the EDMsdk system that Jotne currently works within.

8.6 Conclusion

Looking back at the project, we definitely could have done things differently. We have followed a methodical process with the use of kanban and Scrum, which has been educational, but we were unable to complete tasks within the timeframes we had initially set, which in the end resulted with an incomplete software. The cooperation and communication within the group has been good, even though we had certain incidents that affected the result of the project. This was unfortunate, however we are glad that we were able to come up with a result that we were satisfied with.

Bibliography

[1] Wikipedia. 'Open asset import library.' (), [Online]. Available: https://en.wikipedia.org/wiki/Open_Asset_Import_Library. (accessed: 25.03.2022).

[2] Wikipedia. 'C (programming language).' (), [Online]. Available: https://en.wikipedia.org/wiki/C_(programming_language). (accessed: 17.05.2022).

[3] Clockify. 'Clockify website.' (), [Online]. Available: https://clockify.me/. (accessed: 16.05.2022).

[4] Wikipedia. 'Cmake.' (), [Online]. Available: https://en.wikipedia.org/wiki/CMake. (accessed: 9.05.2022).

[5] Wikipedia. 'International organization for standardization.' (), [Online]. Available: https://en.wikipedia.org/wiki/International_Organization_for_Standardization. (accessed: 04.05.2022).

[6] Wikipedia. 'Industrial computed tomography.' (), [Online]. Available: https://en.wikipedia.org/wiki/Industrial_computed_tomography. (accessed: 01.02.2022).

[7] Wikipedia. 'Voxel.' (), [Online]. Available: https://en.wikipedia.org/wiki/Voxel. (accessed: 01.02.2022).

[8] Wikipedia. 'Polygon mesh.' (), [Online]. Available: https://en.wikipedia.org/wiki/Polygon_mesh. (accessed: 01.02.2022).

[9] Wikipedia. 'Iso 10303.' (), [Online]. Available: https://en.wikipedia.org/wiki/ISO_10303. (accessed: 01.02.2022).

[10] Wikipedia. 'Iso 10303.' (), [Online]. Available: https://www.adobe.com/creativecloud/file-types/image/vector/step-file.html. (accessed: 01.02.2022).

[11] R. Lanza, private communication, 18 May 2022.

[12] Wikipedia. 'Opengl.' (), [Online]. Available: https://en.wikipedia.org/wiki/OpenGL. (accessed: 10.02.2022).

[13] J. d. Vries. 'Coordinate systems.' (), [Online]. Available: https://learnopengl.com/Getting-started/Coordinate-Systems. (accessed: 10.05.2022).

[14] O. Velarde. 'What is a wireframe?' (), [Online]. Available: https://visme.co/blog/what-is-a-wireframe/. (accessed: 18.05.2022).

[15] S. Wolhute. 'Wireframing 101: The whats, the types, and the tools.' (), [Online]. Available: https://wearebrain.com/blog/customer-ux-ui/wireframing-101-the-whats-the-types-and-the-tools/. (accessed: 18.05.2022).

[16] M. Rehkopf. 'What are sprints?' (), [Online]. Available: https://www.atlassian.com/agile/scrum/sprints. (accessed: 27.04.2022).

[17] Kanbantool.com. 'Kanbantool.com homepage.' (), [Online]. Available: https://kanbantool.com/. (accessed: 15.05.2022).

[18] J. d. Vries. 'Learn opengl.' (), [Online]. Available: learnopengl.com. (accessed: 05.02.2022).

[19] T. Root. 'The 7 best c++ frameworks for creating graphical interfaces.' (), [Online]. Available: https://terminalroot.com/the-7-best-cpp-frameworks-for-creating-graphical-interfaces/. (accessed: 26.03.2022).

Appendix A

Project Description

The next pages contains the original project description from Jotne.

Case C: Kvalitetssikring av deler fremstilt med Additiv Tilvirkning ved hjelp av CT-scanning

NB: dette oppgaveforslaget er ikke knyttet til BIM.

I romfartsindustrien er additiv tilvirkning (3D-printing) en tilvirkningsprosess som blir mer og mer aktuell etter hvert som teknologien modnes. I forbindelse med ESA-prosjektet **METRIC,** ser Jotne på muligheten for å produsere en satellittdel i aluminium med denne teknologien. Hovedmålet i prosjektet er å bruke State-of-the-Art teknologi for å redusere kostnader og øke kvalitet og kvalitetskontroll til MAIT-prosessene; (M)anufacturing, (A)ssembly, (I)ntegration og (T)esting, for telecom satellitter. I den forbindelse er det ønskelig å gjennomføre en CT-scan av komponenten for å kvalitetssikre intern struktur og måle kritisk geometri. Målet med dette er å validere parametere for tilvirkningsprosessen og å dokumentere at det endelige produktet tilfredsstiller sine mekaniske og strålings krav.

Det er ønskelig å se på hvordan dataen fra skanningen behandles og hvordan den kan brukes i en digital tvilling av komponenten; en digital representasjon i sanntid av den fysiske komponenten. Digitale tvillinger er et fokusområde for Jotne og en sentral del av METRIC-prosjektet. Et viktig moment ved den digitale tvillingen er at den er basert på den åpne standarden ISO 10303 (STEP), som er essensiell i det meste av arbeid Jotne gjør.

Oppgaven vil først og fremst dreie seg om databehandling fra CT-skanning og hvordan dette kan konverteres til nyttig data for den digitale tvillingen. Data fra skanningen blir lagret i et RAW-format, og det er ønskelig at dette formatet undersøkes og beskrives for å danne grunnlaget for utviklingen i oppgaven og å få data lagret i en ISO 10303 database I utgangspunktet skal alle utviklede algoritmer behandle data fra RAW filene. Den nåværende arbeidsflyten ved laboratoriet hos NTNU Gjøvik nytter seg av programvaren VG Studio MAX, og det er derfor også ønskelig å kartlegge egenskapene ved de prosesserte formatene som kan eksporteres derifra.

- RAW filene inneholder CT-skanningens målinger som er materialets tetthet
- VG Studio MAX eksporterer diverse filer hvor RAW filene har blitt behandlet

Utviklingsdelen av oppgaven kan deles opp i følgende hovedpunkter:

- Utvikling av en algoritme som fra RAW data kan produsere et 3D mesh av skallelementer som representerer objektets ytre flater
    - Det er ønskelig at denne modulen også kan håndtere punktskyer
- Utvikling av en algoritme som fra RAW data produserer et 2D mesh (eventuelt bilde) for indre «lag» i modellen der element er assosiert med tetthetsverdiene fra RAW data.
- Utvikling av en komprimeringsalgoritme som komprimerer en voxel modell til et 3D mesh av volumelementer
    - Komprimering vil hovedsakelig være ved å variere oppløsning basert på materialtetthet og detaljer i den scannede delen

Det er ønskelig at algoritmene blir utviklet i C++, og helst som et sett av moduler (libraries) med individuelle omfang; f.eks. data lesing, data prosessering, data skriving. Dette vil muligjøre oversettelse til STEP på et senere tidspunkt (enten som en utvidelse av oppgaven, eller av Jotne)

Jotne vil bidra med:

- RAW data fra CT-scan samt prosesserte formater
- Jevnlig oppfølging av prosjektet
    - Det er ønskelig med en jevn møtefrekvens, f.eks annenhver uke. Detaljer rundt dette avtales i prosjektets oppstartsfase.

Oppgaver:

- Undersøke og beskrive RAW og prosesserte formater fra CT-maskinen (Zeiss Metrotom 1500)
- Konvertering fra RAW data til følgende:
    - 3D skallelement mesh av ytre flater
    - 2D skallelement mesh (eventuelt bilder) av indre lag
    - 3D volumeelement mesh av volumet
        - Utvikling av en komprimeringsalgoritme for dette meshet

Eksempel på del(er) som er ønskelig å karakterisere ved hjelp av CT-scanning

Appendix B

Project Agreement

The next pages contains the project agreement.

Norges teknisk-naturvitenskapelige universitet

Fastsatt av prorektor for utdanning 10.12.2020

STANDARDAVTALE

om utføring av studentoppgave i samarbeid med ekstern virksomhet

Avtalen er ufravikelig for studentoppgaver (heretter oppgave) ved NTNU som utføres i samarbeid med ekstern virksomhet.

Forklaring av begrep

Opphavsrett
Er den rett som den som skaper et åndsverk har til å fremstille eksemplar av åndsverket og gjøre det tilgjengelig for allmennheten. Et åndsverk kan være et litterært, vitenskapelig eller kunstnerisk verk. En studentoppgave vil være et åndsverk.

Eiendomsrett til resultater
Betyr at den som eier resultatene bestemmer over disse. Utgangspunktet er at studenten eier resultatene fra sitt studentarbeid. Studenten kan også overføre eiendomsretten til den eksterne virksomheten.

Bruksrett til resultater
Den som eier resultatene kan gi andre en rett til å bruke resultatene, f.eks. at studenten gir NTNU og den eksterne virksomheten rett til å bruke resultatene fra studentoppgaven i deres virksomhet.

Prosjektbakgrunn
Det partene i avtalen har med seg inn i prosjektet, dvs. som vedkommende eier eller har rettigheter til fra før og som brukes i det videre arbeidet med studentoppgaven. Dette kan også være materiale som tredjepersoner (som ikke er part i avtalen) har rettigheter til.

Utsatt offentliggjøring
Betyr at oppgaven ikke blir tilgjengelig for allmennheten før etter en viss tid, f.eks. før etter tre år. Da vil det kun være veileder ved NTNU, sensorene og den eksterne virksomheten som har tilgang til studentarbeidet de tre første årene etter at studentarbeidet er innlevert.

1. Avtaleparter

Norges teknisk-naturvitenskapelige universitet (NTNU) Institutt:
Veileder ved NTNU: Ivar Farup e-post og tlf.: ivar.farup@ntnu.no | 61135227 | 91695718
Ekstern virksomhet: Jotne IT Ekstern virksomhet sine kontaktpersoner: Henrik Galtung, Tord Kaasa e-post og tlf.: henrik.galtung@jotne.com, 90852108 | tord.kaasa@jotne.com, 90365197
Student: Sander Island Fødselsdato: 1. juli 1997
Student: Eirik Gjertsen Norbye Fødselsdato: 13. oktober 1997
Student: Jens Fossan Tingstad Fødselsdato: 22. august 1993

Partene har ansvar for å klarere eventuelle immaterielle rettigheter som studenten, NTNU, den eksterne eller tredjeperson (som ikke er part i avtalen) har til prosjektbakgrunn før bruk i forbindelse med utførelse av oppgaven. Eierskap til prosjektbakgrunn skal fremgå av eget vedlegg til avtalen der dette kan ha betydning for utførelse av oppgaven.

2. Utførelse av oppgave

Studenten skal utføre: (sett kryss)

Masteroppgave	
Bacheloroppgave	X
Prosjektoppgave	
Annen oppgave	

Startdato: 10. januar 2022
Sluttdato: 20. mai 2022

Oppgavens arbeidstittel er: CT Scanning as a Tool in Additive Manufacturing

Ansvarlig veileder ved NTNU har det overordnede faglige ansvaret for utforming og godkjenning av prosjektbeskrivelse og studentens læring.

3. Ekstern virksomhet sine plikter

Ekstern virksomhet skal stille med en kontaktperson som har nødvendig faglig kompetanse til å gi studenten tilstrekkelig veiledning i samarbeid med veileder ved NTNU. Ekstern kontaktperson fremgår i punkt 1.

Formålet med oppgaven er studentarbeid. Oppgaven utføres som ledd i studiet. Studenten skal ikke motta lønn eller lignende godtgjørelse fra den eksterne for studentarbeidet. Utgifter knyttet til gjennomføring av oppgaven skal dekkes av den eksterne. Aktuelle utgifter kan for eksempel være reiser, materialer for bygging av prototyp, innkjøp av prøver, tester på lab, kjemikalier. Studenten skal klarere dekning av utgifter med ekstern virksomhet på forhånd.

Ekstern virksomhet skal dekke følgende utgifter til utførelse av oppgaven: Reiser. Ca. 2500 kr Potensiell CT scann av delen *DGU*. Ca. 2000kr dersom dette blir aktuelt.

Dekning av utgifter til annet enn det som er oppført her avgjøres av den eksterne underveis i arbeidet.

4. Studentens rettigheter

Studenten har opphavsrett til oppgaven[1]. Alle resultater av oppgaven, skapt av studenten alene gjennom arbeidet med oppgaven, eies av studenten med de begrensninger som følger av punkt 5, 6 og 7 nedenfor. Eiendomsretten til resultatene overføres til ekstern virksomhet hvis punkt 5 b er avkrysset eller for tilfelle som i punkt 6 (overføring ved patenterbare oppfinnelser).

I henhold til lov om opphavsrett til åndsverk beholder alltid studenten de ideelle rettigheter til eget åndsverk, dvs. retten til navngivelse og vern mot krenkende bruk.

Studenten har rett til å inngå egen avtale med NTNU om publisering av sin oppgave i NTNUs institusjonelle arkiv på Internett (NTNU Open). Studenten har også rett til å publisere oppgaven eller deler av den i andre sammenhenger dersom det ikke i denne avtalen er avtalt begrensninger i adgangen til å publisere, jf. punkt 8.

5. Den eksterne virksomheten sine rettigheter

Der oppgaven bygger på, eller videreutvikler materiale og/eller metoder (prosjektbakgrunn) som eies av den eksterne, eies prosjektbakgrunnen fortsatt av den eksterne. Hvis studenten skal utnytte resultater som inkluderer den eksterne sin prosjektbakgrunn, forutsetter dette at det er inngått egen avtale om dette mellom studenten og den eksterne virksomheten.

Alternativ a) (sett kryss) Hovedregel

[1] Jf. Lov om opphavsrett til åndsverk mv. av 15.06.2018 § 1

	Ekstern virksomhet skal ha bruksrett til resultatene av oppgaven

Dette innebærer at ekstern virksomhet skal ha rett til å benytte resultatene av oppgaven i egen virksomhet. Retten er ikke-eksklusiv.

Alternativ b) (sett kryss) Unntak

X	Ekstern virksomhet skal ha eiendomsretten til resultatene av oppgaven og studentens bidrag i ekstern virksomhet sitt prosjekt

Begrunnelse for at ekstern virksomhet har behov for å få overført eiendomsrett til resultatene:
Oppgave/prosjektbakgrunn inkluderer IPR som tilhører andre selskaper her under Jotne, IDEAS, ESA, eventuelt andre. Software/kode, rapporter som studentene utvikler følger alternativ a).
Jotnes IPR:
EDMopenSimDM, Simulation Data Management client/server application; Digital Twin candidate
EDMtruePLM, Product Data Management client/server application
EDMsdk, Toolkit for developing EXPRESS-based applications and for browsing and manipulating EXPRESS-based data
EDMinterface(C++/AP209), C++ API with AP209 specific convenience functions
EDMstepExplorer, Graphical browser of EXPRESS based data instances
EDMconverter (NASTRAN-to-AP209), Data translator of a subset of NASTRAN data types to AP209
EDMconverter (AP209-to- NASTRAN), Data translator of a subset of NASTRAN data types to AP209
EDMconverter (Abaqus-to-AP209), Data translator of a subset of Abaqus data types to AP209
EDMconverter (AP209-to-ANSYS), Data translator of a subset of Ansys data types to AP209
IDEAS IPR:
DGU, DHU og alle tilhørende maskintegninger, modeller og eventuell CT scann med tilhørende filer.

6. Godtgjøring ved patenterbare oppfinnelser

Dersom studenten i forbindelse med utførelsen av oppgaven har nådd frem til en patenterbar oppfinnelse, enten alene eller sammen med andre, kan den eksterne kreve retten til oppfinnelsen overført til seg. Dette forutsetter at utnyttelsen av oppfinnelsen faller inn under den eksterne sitt virksomhetsområde. I så fall har studenten krav på rimelig godtgjøring. Godtgjøringen skal fastsettes i samsvar med arbeidstakeroppfinnelsesloven § 7. Fristbestemmelsene i § 7 gis tilsvarende anvendelse.

7. NTNU sine rettigheter

De innleverte filer av oppgaven med vedlegg, som er nødvendig for sensur og arkivering ved NTNU, tilhører NTNU. NTNU får en vederlagsfri bruksrett til resultatene av oppgaven, inkludert vedlegg til denne, og kan benytte dette til undervisnings- og forskningsformål med de eventuelle begrensninger som fremgår i punkt 8.

8. Utsatt offentliggjøring

Hovedregelen er at studentoppgaver skal være offentlige.

Sett kryss

X	Oppgaven skal være offentlig

I særlige tilfeller kan partene bli enige om at hele eller deler av oppgaven skal være undergitt utsatt offentliggjøring i maksimalt tre år. Hvis oppgaven unntas fra offentliggjøring, vil den kun være tilgjengelig for student, ekstern virksomhet og veileder i denne perioden. Sensurkomiteen vil ha tilgang til oppgaven i forbindelse med sensur. Student, veileder og sensorer har taushetsplikt om innhold som er unntatt offentliggjøring.

Oppgaven skal være underlagt utsatt offentliggjøring i (sett kryss hvis dette er aktuelt):

Sett kryss		Sett dato
	ett år	
	to år	
	tre år	

Behovet for utsatt offentliggjøring er begrunnet ut fra følgende:

Dersom partene, etter at oppgaven er ferdig, blir enig om at det ikke er behov for utsatt offentliggjøring, kan dette endres. I så fall skal dette avtales skriftlig.

Vedlegg til oppgaven kan unntas ut over tre år etter forespørsel fra ekstern virksomhet. NTNU (ved instituttet) og student skal godta dette hvis den eksterne har saklig grunn for å be om at et eller flere vedlegg unntas. Ekstern virksomhet må sende forespørsel før oppgaven leveres.

De delene av oppgaven som ikke er undergitt utsatt offentliggjøring, kan publiseres i NTNUs institusjonelle arkiv, jf. punkt 4, siste avsnitt. Selv om oppgaven er undergitt utsatt offentliggjøring, skal ekstern virksomhet legge til rette for at studenten kan benytte hele

eller deler av oppgaven i forbindelse med jobbsøknader samt videreføring i et master- eller doktorgradsarbeid.

9. Generelt

Denne avtalen skal ha gyldighet foran andre avtaler som er eller blir opprettet mellom to av partene som er nevnt ovenfor. Dersom student og ekstern virksomhet skal inngå avtale om konfidensialitet om det som studenten får kjennskap til i eller gjennom den eksterne virksomheten, kan NTNUs standardmal for konfidensialitetsavtale benyttes.

Den eksterne sin egen konfidensialitetsavtale, eventuell konfidensialitetsavtale den eksterne har inngått i samarbeidprosjekter, kan også brukes forutsatt at den ikke inneholder punkter i motstrid med denne avtalen (om rettigheter, offentliggjøring mm). Dersom det likevel viser seg at det er motstrid, skal NTNUs standardavtale om utføring av studentoppgave gå foran. Eventuell avtale om konfidensialitet skal vedlegges denne avtalen.

Eventuell uenighet som følge av denne avtalen skal søkes løst ved forhandlinger. Hvis dette ikke fører frem, er partene enige om at tvisten avgjøres ved voldgift i henhold til norsk lov. Tvisten avgjøres av sorenskriveren ved Sør-Trøndelag tingrett eller den han/hun oppnevner.

Denne avtale er signert i fire eksemplarer hvor partene skal ha hvert sitt eksemplar. Avtalen er gyldig når den er underskrevet av NTNU v/instituttleder.

Signaturer:

Instituttleder: Dato:
Veileder ved NTNU: Dato: 2022-02-03
Ekstern virksomhet: Dato: 31.01.2022
Student: Dato: 16.02.2022
Student: Dato: 16.02.2022
Student: Dato: 17.02.2022

Appendix C

Project Plan

The next pages contains the project plan.

CT SCANNING AS A TOOL FOR QUALITY ASSURANCE IN ADDITIVE MANUFACTURING

Project Plan

Bachelor project NTNU 2022

Jens Fossan Tingstad
Eirik Gjertsen Norbye
Sander Island

Table of Contents

1 GOALS AND FRAMEWORK

1.1 Background

Jotne EPM Technology is a leader in the development of standard-based software products. They specialize in product data exchange, product lifecycle management, long-term data and product, OAIS archiving, data validation and verification, code checking, rule-based data modeling and cross-platform data sharing within the structure of objects Jotne has produced.

In the aerospace industry, additive manufacturing (3D-printing) is increasingly relevant as technology progresses. In collaboration with the European Space Agency (ESA)-project METRIC, Jotne is looking at the opportunity of producing a satellite component of aluminum using additive manufacturing technology. The main goal of this joint effort is to use state-of-the-art technology to reduce cost and increase quality and quality assurance to the MAIT (Manufacturing, Assembly, Integration, Testing)-processes for telecom satellites. It is desired to use CT-scans of the component to assure the required quality, both internal structure and geometric dimensions. The point of this is to validate and document that the final product satisfies within all parameters.

Jotne wants to see how the scan data is managed and how it can be used for a digital twin of the component. A digital twin is a virtual representation that serves as a real-time digital counterpart of a physical object, in simpler terms, the digital twin is a copy of an object from the real world to the virtual world. Digital twins are a focus area for Jotne and a vital part of the METRIC project. An important addition is that the digital twin is based on the open standard ISO 10303 (STEP), which is essential in most of Jotne's work.

1.2 Project goals

- The primary goal of the project is to process RAW data from a CT-scan of the 3D-printed part and treat these datapoints using algorithmic methods to build a mesh based virtual model matching the original.
- The secondary goal revolves around evaluating the internal structure of the part and making sure the internal defects are within set parameters.
- Lastly, the results must be presented in a visual way, so that someone without an engineering background can make sense of the results.

1.3 Effect goal for client

The client should expect a library of modules, which is able to produce a finished mesh model by applying algorithmic solutions to a RAW data cloud. Optimally, additional functionality will be implemented to allow analyzation of porosity. Jotne will be able to integrate these solutions in their existing system, meaning the results can be used for quality inspection in the future.

1.4 Learning goals

- Learn how to structurally execute a long-term project.
- Learn about CT-scan data and how to read and use it properly.
- Get further knowledge of different approaches and techniques we can implement in our project.
- Get insight into how our client operates in their work environment and what work they do.
- Learn how to work with a scrum board, with weekly timeframes.
- Reviewing every two weeks, resetting and discussing what to do next.

1.5. Project frame

Time is of the essence, and development of this project must be completed between 10th of January and the 1st of May. Primarily to give us time to spend on the report after the main development stage.

1. The code must be written in C++
2. The code must be structured into modules
3. No processing of the data should be needed before using our program
4. The solution should work on all CT-scanned parts
5. The solution must comply with ISO10303

2 SCOPE

We aim to deliver a set of software modules, structured in libraries. In the future, the scope may be expanded to include a relevant and intuitive graphical user interface.

2.1 Subject area

The project will largely be within the fields of algorithmic thinking and computer vision. We classify our problem area as the 3D-printing and CT-scanning space, since our solutions are thought to evaluate their necessity in the aerospace industry in 2022.

We will not be creating a fully simulated digital twin, our focus is primarily on shape and internal structure, not physical properties. This project is only meant to manipulate data, not to visualize it. Therefore, third-party software can be used to visualize the final product.

2.2 Problem area

Our employer JOTNE wants to look at the relevance of 3D-printing in the aerospace industry. In doing so they wish to use CT-scanning to analyze the quality of state-of-the-art printing technology. They have tasked us with manipulating the RAW data from the scanning process and transform it into a virtual model, which they can use to evaluate the part. For us this means building a mesh model replicating the objects surface. Furthermore, our solution should present data on internal defects based on their size and location.

2.3 Problem statement

We aim to create software which can manipulate RAW data from a CT-scanned part to build a mesh model replicating the objects surface. Furthermore, our solution should present data on internal defects based on their size and location.

3 ORGANIZATION

3.1 Responsibilities and roles

The bachelor group consists of:

Jens Tingstad: Project manager.

Sander Island: SCRUM master.

Eirik Norbye: Meeting management & project documentation.

3.2 Employer

Jotne EPM Technologies AS

- Location: Helsfyr, Norway.
- Represented by: Henrik Galtung, Tord Kaasa.

3.3 Supervisor

Ivar Farup, Professor

- Department of Computer Science.
- Faculty of Information Technology and Electrical Engineering.

3.4 Routines and rules

Desired routines:

- Meet up at school/online every weekday 8-9 AM and get at least 6 hours of work done. School or online depending on what is most practical in case of pandemic restrictions or if someone is unable to meet physically.
- Update the scrum board according to progress within the project.
- Update Clockify for tracking work time every day.

Rules:

- Meet up at scheduled time.
- Have an acceptable reason to not work in the scheduled sessions, such as other work, sickness…

If proper rules and/or routines are not followed, the group will meet to discuss solutions:

1. First offence will result in a warning during the subsequent meeting.
2. Recurring offences will result in a formal written warning, signed by the rest of the group.
3. If rules are broken regularly, steps will be taken to ensure the integrity of the project. The supervisor will be involved in handling the situation, and in a worst-case scenario, the subject may be removed from the group.

4 PLANNING, FOLLOW-UP AND REPORTING

4.1. Software development strategy

When researching different development strategies, we had a few demands. The first being it must be an iterative process, meaning we can make changes and updates to the planned schedule as we go. This is especially important when it comes to code heavy projects, since making changes in one section of code might mean having to review other parts of the project. Secondly, we wanted a method that embraced breaking down the project into smaller sizes to be individually reviewed. Since we are planning to rely heavily on employer and supervisor input, we want smaller chunks of work, so it is easily approved/rejected during discussions.

Based on this, we quickly decided agile development methodologies should be investigated. After reviewing multiple different models, we decided on something that matched all our criteria, and formalized the workflow we had already intended.

Since creating an efficient workflow is essential, we decided to use SCRUM, an agile development strategy. We chose this strategy as it focuses on short sprint-based development cycles. Where we set ourselves small goals with short and specific deadlines, spanning at most two weeks. After each sprint we will review our work in collaboration with both our supervisor and employer. Here we will discuss problems and produce viable solutions. The point of this is to keep the next goal as clear as possible, and make sure it is within the realm of reason that the work will be completed in the given timeframe.

4.2 Plan for status meetings and decisions

Status meeting will be performed every other week to keep our employer up to date, besides that we will also have weekly status reports with our supervisor to get feedback concerning changes we make within our project.

5 QUALITY ASSURANCE

5.1 Documentation, standards, configurations management, tools...

As this project contains many different fields, we will be using a variety of software for different applications:

- CLion and/or MATLAB

    - Main coding hubs
- GitLab
    - Communal code sharing and reviews, for collaboration with our client and supervisor.
- ISO 10303
    - Standard for final deliverables, end product must adhere to its demands.
- Code documentation
    - All methods and classes will have clear descriptions, both when it comes to functionality, content and parameters.

As for project documentation we will make use of Stackfield, where we put all of the milestones we need to reach within a certain deadline. Thus, giving us a solid overview of what is already in progress and what's done within our project. The initial milestones are as follows:

- Research – project planning.
- RAW data manipulation.
- Dividing point clouds into manageable chunks.
- Writing algorithms to construct surfaces.
- Building mesh model.
- Evaluate resulting mesh.
- Writing algorithms for internal structure.
- Analyzing internal structure.
- Report sprint.

5.2 Plan for inspection and testing

Inspection and testing will be conducted continuously during the development process, after and during each sprint. We will primarily use GitLab as a resource for inspecting the code while it's under progress with our employer and supervisor. As for testing we will use the built-in debugger tool in CLion, which we have some prior experience with. If this turns out to be troublesome, we will make use of Stack Overflow, which isn't exactly debugging, but more of a feedback/inspection forum where we can get some input about certain changes/modifications that we need to make. Finally, we will also make use of regression testing which entails doing continuous tests after changes that have been made in the code, to make sure it compiles before pushing it into the main branch.

5.3 Risk analysis

In the risk chart below, the perceived potential risks are shown. Likelihood and impact score are on a 1-10 scale, higher numbers translate to more severity.

Situation	**Impact**	**Impact score**	**Requirements**	**Likelihood**	**Risk (impact x likelihood)**
Health problems	Could reduce capacity to meet deadlines,	5	Bad luck	2	10

	crippling progress both internally and towards employer/supervisor				
Missing access to critical resources	Has the potential to stop us from testing and improving our algorithms, in addition to stopping us from visualizing our solutions	6	Rejected access to required computational environments, or rooms fitted with necessary equipment	3	18
Covid restrictions	Certain restrictions may interfere with workflow that impedes progress	3	Severe covid outbreak, lockdowns, campus closed	7	21
Failing to CT scan the physical part	Makes all the algorithmic work done theoretical for it's practical application reducing the impact of our work	5	Failure by employer to deliver required part, failure by NTNU CT-lab to allot time and manpower needed for scanning	5	25
Bad project management	Could disrupt workflow throughout the project, and greatly reduce our capacity to meet our goals	7	Poor communication, insufficient planning, lacking collaboration on difficult problems, no backup	4	28
Imprecise algorithms	Failing to develop sufficiently sophisticated algorithms	9	Lack of cooperation, research, insufficient coding abilities,	5	45

	means the final product has the potential to fall short of the tolerances set for this project, in this case the final product will not be put to use		poor understanding of datatypes, difficulties in adapting existing libraries		

5.4 Risk management

No project is without risk, so we must accept that there is a genuine possibility one or more of the aforementioned risks could come to fruition. What matters is how we prepare to tackle these challenges to mitigate risk.

5.4.1 Bad project management:

To combat this, we have set up several communication channels using different tools, we will use GitLab for coding related work, “Stackboard" for our general project management, timelines and SCRUM activities. In addition, we have scheduled weekly meetings with our supervisor every Wednesday, and our employer every other Tuesday. This way we can keep everyone up to date and make sure we have soft but important deadlines more frequently. Discord is also used as a low effort communication platform internally, and Teams is used to communication between the group and our employer.

5.4.2 Missing access to critical resources:

Regarding resources there is a limit to what we can do, as this ultimately is out of our control. However, what we do within this limit is important. Firstly, we will try to get access to the VR-lab on campus, this lab contains two powerful computers capable of working with complex and large data files. If this fails, we aim to get access to IDUN – a high performance cloud-based computing cluster, where we can run computational tasks remotely. As a last resort, we will invest in hardware ourselves to make sure we are able to reach our goals.

5.4.3 Imprecise algorithms:

This point relates to the first point about project management, as this can lead to severe crippling of the algorithmic work. On its own, however, it is important to realize the difficulty of the subject. Writing good code to manipulate complex data takes a lot of work, both in time and brainpower. Therefore, we will be spending a lot of time initially to understand the data we are working with, research related libraries, and refresh our knowledge of C++. If we get stuck at any point, it is important to take a step back and discuss where the problems lie, if we can divide the problem into smaller chunks or take a

different approach. If this fails, we will discuss our approach with our supervisor, employer, and others. It is of high importance to use all the resources available to us at this stage.

5.4.4 Failing to CT scan the physical part:

Clear communication with the staff at the CT-lab is vital in this case. Communication will happen primarily by email, but there is a plan in motion to get a physical tour of the lab and a tutorial in their workflow. By doing this we get more insight into the entire process, and it might provide relevant information we can use at a later state in development. Should we for any reason be unable to contact the staff by email, we will seek them out in their office to start communication. If our employers are unable to provide the part we need within a reasonable time, we will work with the data we got and make sure the code we run on that data is within the tolerances set by the project.

5.4.5 Health problems:

To avoid any illness, primarily covid related, we are following government guidelines to mitigate as much risk as possible. We all live in communal houses, so there is a chance someone in the household is not following these guidelines. In a worst-case scenario, we have all our project management systems online, and therefore working from home in a digital environment is possible, but not ideal.

5.4.6 Covid restrictions:

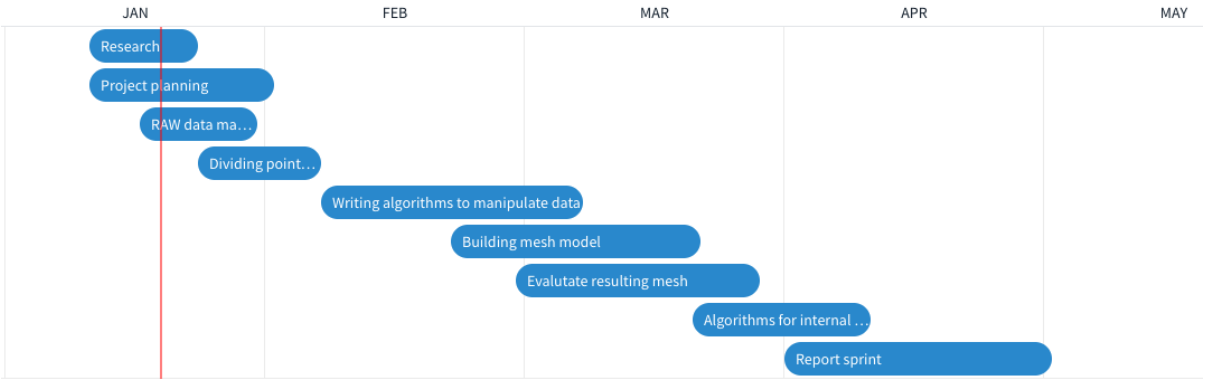
New restrictions are following the infection trend, which is ever-changing. If new regulations demand the closing of campus, we will need to push the deadlines of heavy computational work. By using the SCRUM model, we should have many different tasks that require work at all points in time. Therefore, if campus closes for a few weeks other parts of the project should be ahead of schedule.

6 IMPLEMENTATION PLAN

All work will be organized into small tasks on a Kanban/SCRUM board, with softer deadlines. Major milestones are logged in a Gantt-chart, in which case the deadlines are firm.

Firstly, we will look at only the surfaces of the object, thus we will be neglecting the internal structure of the material in the first phase of the project. We will break down the different phases of the problem in the following way:

- Research data types and existing solutions
- Play around with the data, try manipulating it in simple ways
- Create a 2D sketch of the vertices in a given plane.
- Duplicate this sketch along a set number of planes.
- Create a 2D sketch of the vertices in a plane oriented perpendicularly to the original and duplicate this along a set number of planes.
- Combine the sketches to create a 3D model.
- Convert the model to a mesh based one.
- Review the model.
- Look at internal defects of the aluminum alloy making up the part and set tolerances.
- Check that the internal structure is within set parameters



Appendix D

Meetings

D.1 Møtereferat 01.02.2022

Møte med Jotne.
Alle til stede.

De foreslo å sponse tur til Oslo, sånn at vi kan få sett hvordan de jobber.

Vi fikk opphavsrett til oppgaven, sånn at vi får lov til å publisere.

1600 kr for en CT-scan, stor mulighet for å få støtte fra NTNU, hvis ikke så kunne oppdragsgiveren støtte oss.

Sensorplate, bilde sensor som du har i kamearet ditt. som har ett antall pixler, og så har du en røntgenkilde også har du en kjegleformet stråle som treffer den detektoren. mengde stråling, per areal blir mindre

Stort objekt = dårligere oppløsning.

D.2 Møtereferat 02.02.2022

Møte med Ivar.
Alle til stede.

Vi får bruke VR-lab som arbeidsplass/kontor for å ha tilgang på kraftigere PC om nødvendig.

Trenger vi å bruke RAW-dataen? Hør med Jotne om vi trenger å lage ett program for å konvertere RAW data, når det er allerede finnes flere løsning for dette. Skal vi heller fokusere på å få ut spesifikk info fra RAW data? Som for eksempel porøsitetsnivå osv. på objektet.

Har de egen CT scanner? Dersom de bestiller scans, får de allerede prosesserte filtyper i retur.
Finne metode for å integrere grensesnitt med eksisterende løsninger hos Jotne.
Endre dreining på oppgaven til dataanalyse?

Vi må undersøke RAW data, se om 3GB-filen er den samme som 13GB, men uten støy. Se på parameterene i voxels, se om 8-bit formatet passer parametre. (grey-scale, x, y, z).
Finne startvoxel for modellen, se om den er i senter/kant, hypotese om første genererte voxel. Generere startmodell typ 100x100x100 for å teste endringer (edge detection, mesh generation, remove faces, orientation).

D.3 Møtereferat 07.02.2022

Møte med Jotne.
Alle til stede.

Finn ut hvilke verktøy det finnes i VG STUDIO MAX.

Sjekk om det går an å identifisere enkelte områder, gi input.
Automatisk identifikasjon av porer? Finnes det funksjonalitet i VG STUDIO MAX for å finne porer?

Er det mulig å komprimere filer ved å fjerne unyttige voxler, evt. å slå sammen voxler?

Endre fokus fra å lage mesh etc. til å prøve å oversette til STEP.

Format for mesh og voxler er standardisert i ISO.

D.4 Møtereferat 09.02.2022

Møte med Ivar.
Alle til stede.

Recap om hva vi ble enig med Jotne om: det å ikke bygge mesh, men å fokusere på dataanalyse istedenfor.

Prøve å bruke OpenGL til visualisering.

D.5 Møtereferat 15.02.2022

Møte med Jotne.
Alle til stede.

Oppsummering av hvor langt vi har kommet.

Vi har ikke tenkt så langt med tanke på STEP-konvertering, men vi har sett at det ligger en slags oppskrift på nett for det, med tanke på hva vi driver med i OpenGL.
Akkurat nå er fokus på å få videreutviklet 3D-modellen vi har laget nå, også tar vi det derifra.

D.6 Møtereferat 16.02.2022

Møte med Ivar.
Alle til stede.

Jens har fått covid og er med på møtet digitalt.
Fare for lite fremgang når vi bare blir to de neste dagene.
Oppsummering av hvordan vi ligger an med OpenGL og hva vi har snakket med Jotne om.
Sander og Eirik vil fokusere på gruppearbeid i ingeniørfaglig systemtenkning mens Jens er syk.

D.7 Møtereferat 24.02.2022

Møte med Jotne.
Sander og Jens til stede.

Henrik og Tord er på besøk på NTNU og CT-laben.
Vi spiste lunsj og viste Henrik og Tord litt rundt på campus, samt en tour på CT-laben.

D.8 Møtereferat 09.03.2022

Møte med Ivar.
Alle til stede.

Statusoppdatering; Lite fremgang de siste ukene pga. sykdom og en del press fra andre fag.
Ble gjort et lite retningsskifte i forhold til koding. Har som smått begynt å lage rapporten i Overleaf. Har ordnet litt struktur og har skrevet på noen punkter.

D.9 Møtereferat 15.03.2022

Møte med Jotne, kun Tord.
Alle til stede.

En liten statusrapport om at vi har begynt å få struktur på rapporten, og Jens og Eirik lærer seg OpenGL.

Tord og Henrik vil gjerne holde foredrag/lære oss mer om STEP. Det hadde vært fint for vår del.

D.10 Møtereferat 25.03.2022

Møte med Ivar.
Sander og Jens til stede.

Statusrapport om hvor vi ligger an. Jens og Eirik har kommet et godt stykke med OpenGL. Vi har fått sett på STEP og satt opp litt struktur på rapporten. Sander skal begynne og se på GUI.

Send foreløpig rapport til Ivar mandag 4. april.

D.11 Møtereferat 06.04.2022

Møte med Ivar.
Sander og Jens til stede.

Gikk gjennom et foreløpig utkast av rapport vi hadde sendt inn til Ivar på forhånd.
Små innspill fra Ivar i form av kommentarer lagt inn i PDF-dokument.
Innspill gikk ut på alt fra småting i setninger til struktur på selve rapporten.
Sander har begynt lage GUI i Qt Creator, og vi skal prøve å få koblet GUI-en sammen med koden Jens og Eirik jobber på.

D.12 Møtereferat 12.04.2022

Møte med Jotne.
Sander til stede.

Gitt en oppdatering på hva vi har gjort.
Sander har knotet mye med GUI de siste ukene og sliter med å få det til å funke.

Prøve å få til tur til Jotne siste uka i april.

D.13 Møtereferat 20.04.2022

Møte med Ivar.
Alle til stede.

Vi gikk gjennom at vi hadde tatt kontakt med en på Twitter som har jobbet med Assimp i flere år om han hadde noen tips for utvikling av oversetting til STEP.
Fikk til svar at vi heller burde eksportere en obj.-fil og så bruke FreeCAD til oversetting til STEP, fordi STEP er et monstrøst format.
Gikk gjennom løsninger: bruke et lite subsett av STEP-standarden, så vi ikke bruker hele. Forslag fra Ivar.

Alternativ 2: Lage workflow for Jotne som beskriver fremgangsmåte fra mesh/-voxel til STEP.

Rapportskriving: vurdere om vi skal ha uviklingsrapport, utredningsrapport eller en kombinasjon.

Sett opp fysisk møte med Jotne neste uke (uke 17).

D.14 Møtereferat 21.04.2022

Møte med Jotne.
Sander og Jens til stede.

Gikk gjennom det vi fant ut om STEP-konvertering via en på Twitter som har jobbet mye med Assimp-biblioteket.
Vi gikk også gjennom våre alternativer med å bare bruke et subsett av STEP-formatet og å lage en salgs workflow for Jotne som beskriver fremgangsmåte.
Henrik og Tord tror det er mulig å bruke et subsett.

Viste Henrik og Tord forslag på GUI og fortalte plan for den og at vi sliter med å få ting til å kommunisere.

Det er satt opp et møte med en med kompetanse innen STEP fra Jotnes side i morgen (22. april). Han har nok gode innspill på hvordan vi skal løse dette.

Fysisk møte hos Jotne blir onsdag 27. april klokka 12 i Oslo.

D.15 Møtereferat 22.04.2022

Møte med Jotne.
Alle til stede.

Jotne stilte med IT-mann Remi som har peiling på STEP.
Remi viste et bibliotek som Jotne har som går ut på eksport av STEP-filer og funksjonalitet de har liggende.

Vårt mål fremover blir å få til import av voxel-filer, i og med at vi kun har mesh nå.
Deretter må vi få til eksport av filene ved bruk av Jotnes funksjonaliteter.

Dette er definitivt et møte vi burde hatt my tidligere i prosjektet.

D.16 Møtereferat 26.04.2022

Møte med Jotne.
Alle til stede.

Vi har fått tilsendt et ekesmpelprosjekt av Remi som viser litt hvordan man bruker STEP-konverteringsfunksjonaliteten til Jotne.
Vi har sett litt på det og har delvis skjønt hvordan det er bygd opp, men sliter med å få det til å kjøre.
Det er satt opp møte med Remi i morgen så vi kan få stille eventuelle spørsmål.

Planlagt møte hos Jotne i Oslo i morgen er klokken 11:30.

D.17 Møtereferat 28.04.2022

D.17.1 Møte 1

Møte med Jotne.
Alle til stede.

Møte med Jotne der vi fikk en gjennomgang av Remi hvordan eksempelet hans funker og vi fikk skrevet ut en stp-fil (STEP-fil).
Denne STEP-filen er en såkalt P21-fil, som er en ASCII-fil. P21 kommer fra at det er part 21 av STEP-standarden (ISO 10303-21).
For å få dette prosjektet over til CLion tror Remi at vi må se på preprocessor definitions og precompiled headers.

D.17.2 Møte 2

Møte hos Jotne.
Alle til stede.

Fikk en presentasjon av hva Jotne driver med og det ble mye prat om Jotnes prosjekter.

D.18 Møtereferat 29.04.2022

Møte med Ivar.
Sander og Jens til stede.

Viste frem eksempelet fra Remi og forklarte litt hvordan det fungerer og nevnte at vi gjerne burde hatt møtet med Remi for lenge siden.
Forklarte planene våre om å få til å importere og å skrive ut en P21-STEP fil.

D.19 Møtereferat 02.05.2022

Møte med Jotne.
Alle til stede.

Møte med Arne Tøn fra Jotne, som jobber mye med EDM og kan mye om det.
Prøvde å få hjelp til å få eksempelprosjektet til Remi til å funke i CLion, slik at vi kan prøve å koble det sammen med resten av koden vi har.

D.20 Møtereferat 04.05.2022

Møte med Ivar.
Sander og Jens til stede.

Gikk gjennom at vi har slitt med å få koden til å funke. Jens skal prøve å få til at koden fungerer med voxler.
Begynner å nærme seg at vi må se bort fra utviklingen og kun fokusere på rapporten.
Ivar foreslo at vi sender inn enkelte kapitler for vurdering/gjennomgang, slik at det ikke blir hele rapporten på én gang. Ivar mener også det kanskje begynner å bli gunstig å dreie mere mot en utredningsrapport kontra en utviklingsrapport.

D.21 Møtereferat 10.05.2022

Møte med Jotne.
Alle til stede.

Vi sa vi har sluttet med utviklingsprosessen og har kun fokus på rapporten frem til leveringsfristen.
Siden vi ikke har fått til et fungerende produkt enda, sa vi at vi har lyst til å prøve

videre etter rapporten er levert og frem til fremføringen av prosjektet.
Remi er tilbake fra ferie, så han er tilgjengelig om vi trenger hjelp.

D.22 Møtereferat 11.05.2022

Møte med Ivar.
Alle til stede.

Vi hadd sendt inn et foreløpig utkast av rapporten og fikk Ivar til å lese gjennom og komme med tips.
Han sa det var veldig mye prosessrelevant tekst under Implementation-kapittelet. Dette kan fikses med at vi kaller kapittelet Process and Implementation, eller eventuelt har et eget kapittel om prosessen.
Videre var det mer ting å skrive på diskusjonsbiten om effektmål, resultatmål og prosessen i seg selv.

Snakket om planer om å prøve å fullføre prosjektet etter leveringsfristen, både for vår egen del og for Jotnes del.

Oppsummerte det vi gikk gjennom i forrige møte med Jotne.

D.23 Møtereferat 18.05.2022

Møte med Ivar.
Sander og Jens til stede.

Gikk gjennom en forhåndsinnsendt rapport og fikk feedback:
- Fjern "Functionality"-overskriften, slik at subsectionsa ligger under OpenGL
- Bruk sections i sprintene i Process & Implementation
- Fjern "Structure"-overskriften i Design
- Flytt "Wireframes" fra Design til Theory og skriv litt mer om hvilke designprinsipper som er fulgt i wireframen
- Skriv om hvordan vi har svart på requirements i Discussion
- Gjerne bruk descriptionlist i stedet for subsections

Appendix E

Project Contribution

This table shows how much each member has contributed in each aspect of the project. The table is organized with a scoring system from 1 to 5, where 5 indicates that the member has done a lot of work. A score of 1 indicates that the member has done little work in the given aspect.

Table E.1: Member contribution

Chapter	**Sander**	**Jens**	**Eirik**
Research	5	5	5
Code	3	4	5
GUI	5	1	1
Documentation	4	2	5
Report	5	5	5

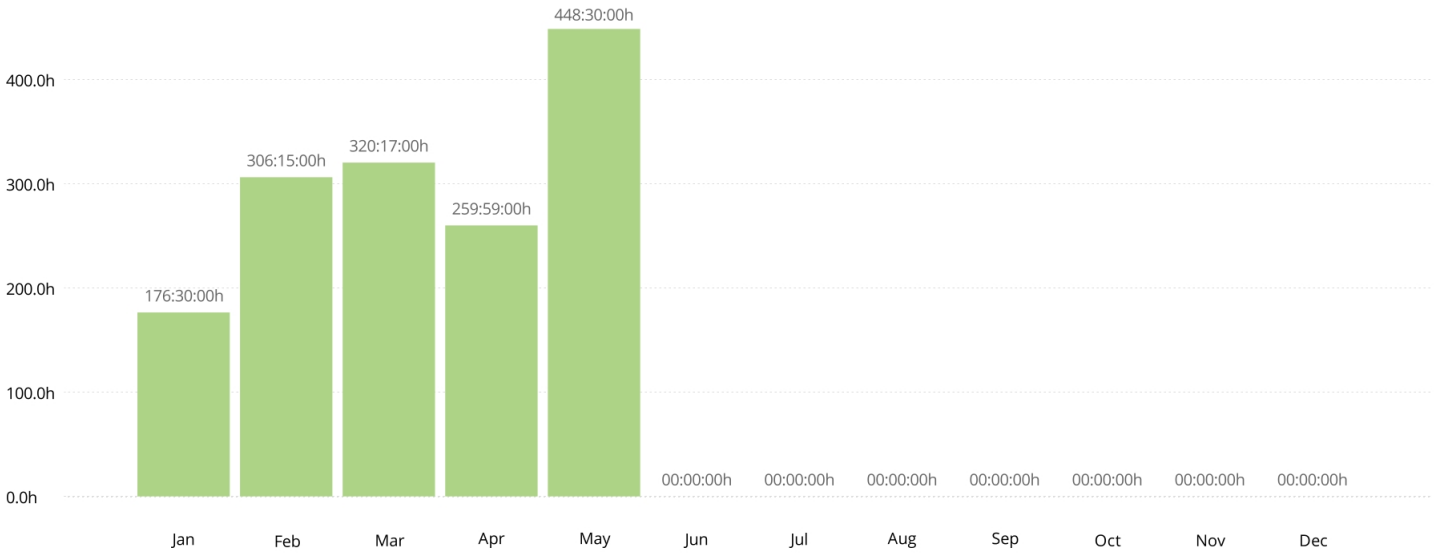
Appendix F

Work Hours

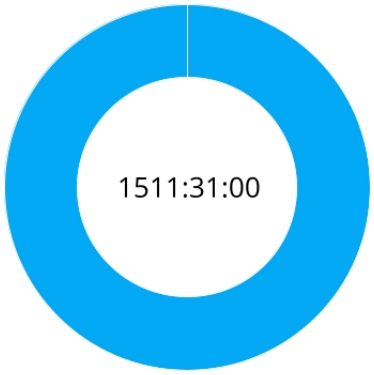
Summary report

01/01/2022 - 31/12/2022

Total: 1511:31:00 Billable: 00:00:00 Amount: 0.00 USD

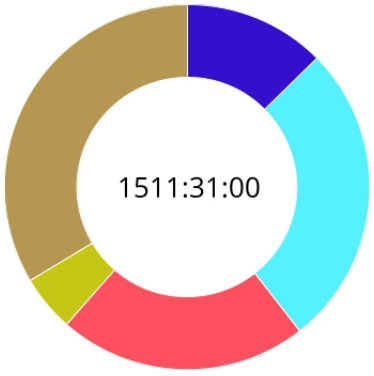


Project



Bachelor	1511:31:00	100.00%

Task



Code - Bachelor	507:15:00	33.56%
Meeting - Bachelor	76:00:00	5.03%
Report - Bachelor	332:45:00	22.01%
Research - Bachelor	405:31:00	26.83%
Start-Up - Bachelor	190:00:00	12.57%

Project / Task	Duration	Amount
Bachelor	1511:31:00	0.00 USD
Code	507:15:00	0.00 USD
Meeting	76:00:00	0.00 USD
Report	332:45:00	0.00 USD
Research	405:31:00	0.00 USD
Start-Up	190:00:00	0.00 USD