NTNU

The Norwegian University of Science and Technology

Visualizing 3D BIM Models in the Real World Using Augmented Reality

Authors

Bjarte Klyve Larsen

Morten Omholt-Jensen

Jørgen Hanssen

Programming

Department of Computer Technology and Informatics

The Norwegian University of Science and Technology, Gjøvik, Norway

Preface

The work presented in this thesis was conducted by Bjarte Klyve Larsen, Morten Omholt-Jensen and Jørgen Hanssen throughout our final semester of the Bachelor's degree in Programming at the Norwegian University of Science and Technology (NTNU). We have worked together on various projects previously and chose to collaborate on the thesis based on our earlier experience as a team and our shared commitment to work towards an outstanding result.

Our collaboration with Rendra AS emerged as a consequence of a close relationship through Morten's mother as a former CEO, as well as Larsen's job there as a part-time developer. Rendra proposed the suggestion for the thesis early in the previous semester, and we found the project intriguing. We accepted the task as it seemed exciting, challenging, and more extraordinary than what was expected for the thesis.

Acknowledgments

We want to convey our highest appreciation to our supervisor Øivind Kolloen, for guiding us throughout the thesis, and to Pål-Robert Engnæs, Jean Niklas Lorange, Simon Žagar and the rest of the team at Rendra for providing us with the opportunity and assistance to conduct such an exciting and unusual project. Thanks to Mariusz Nowostawski for helping us with the technical difficulties that occurred during the project and for stepping in as our substitute supervisor when Kolloen was ill. We would also like to thank all the individuals who tested and reviewed our implementation.

Their sincere belief in the project and our efforts have been incredibly touching.

Abstract

The construction industry is one of the least digitized industries today, which generally has a low allocation of resources for research and development. However, one of the digital advances in the industry has been the adoption of building information modeling (BIM), which is a 3D model-based process that allows workers and stakeholders to manage the construction process collectively and digitally. The objective for this thesis is to ascertain how the implementation of augmented reality (AR) with BIM, can advance the modernization of the building industry, as well as excel StreamBIM as a competing BIM platform; allowing users to work in new productive and interactive ways on site.

This thesis highlights the challenges, solutions, and results of implementing AR in a BIM application for both iOS and Android. The application uses native technologies for rendering and AR, and React-Native for integrating these native implementations in a shared codebase with a uniform UI. This thesis has resulted in an application capable of visualizing and mapping BIM models to real-world buildings using StreamBIM's existing data.

Contents

Abbreviations

BIM	Building Information Modeling
GPU	Graphics Processing Unit
AR	Augmented Reality
UI	User Interface
UX	User Experience
API	Application Programming Interface
QA	Quality Assurance
R&D	Research and Development
HVAC	Heating, ventilation, and air conditioning

1 Introduction

The construction industry is one of the least digitized industries today, which generally has a low allocation of resources for R&D. The construction sector is, therefore, categorized as a low-intensity R&D sector by the Economics of Industrial Research and Innovation (IRI), as it typically uses less than 1% of resources towards R&D. Other sectors in this category include tobacco, mining, and food/drug retailers. In contrast, high-intensity sectors include, e.g., pharmaceuticals, software, defense, and automobile. To put this into perspective, Volvo, an automobile company located in Sweden, spent 14 times as much on R&D than the French construction company Bouygues in 2018, even when the two companies have similar net sale profits [1]. Using data from the IRI, figure [1] illustrates the considerable difference in R&D spendings between the two companies throughout recent years.

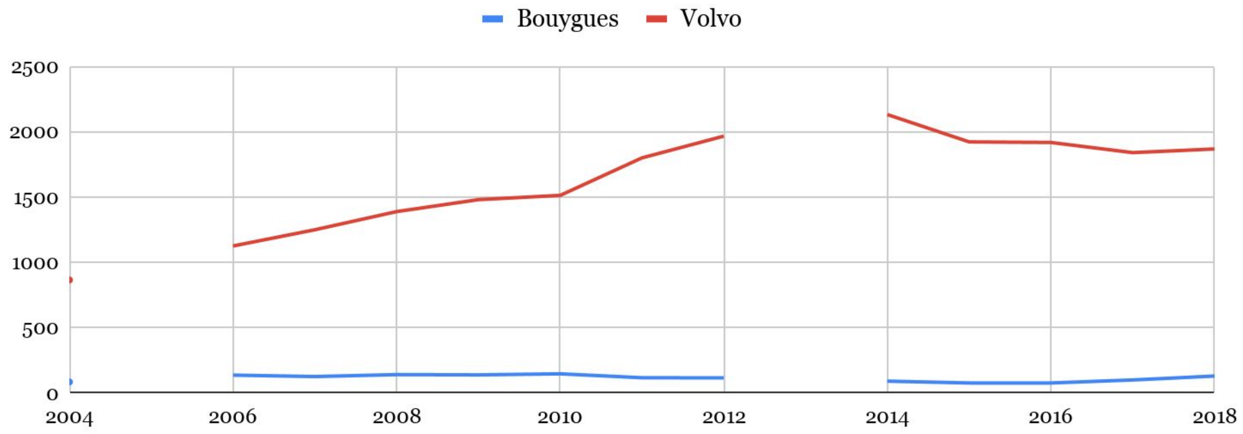


Figure 1: R&D spending (in million) for Volvo and Bouygues in the period 2004 - 2006

Estimations predict that the construction industry could benefit significantly from R&D and digitization; yielding the industry as much as $1.6 trillion through increases in productivity alone [2]. One of the digital advances in the industry has been the adoption of BIM, which is a 3D model-based process that allows workers and stakeholders to manage the construction process collectively and digitally. Today, with the recent advancements in AR technologies, it is argued that AR implementations in BIM software will advance the industry further.

1.1 Project Description

The objective for this thesis is to ascertain how the implementation of AR in the BIM platform *StreamBIM* can advance the modernization of the building industry, as well as excel StreamBIM as a competing BIM product; allowing users to work in new productive and interactive ways on site. This objective will be accomplished by developing a new standalone native AR app for mobile devices running iOS and Android, using data from StreamBIM's existing infrastructure and technologies.

- Considering StreamBIM's state as a web app that currently utilizes web technologies for rendering, native rendering engines will be implemented in the new app. The engines will use StreamBIM's existing model data.
- As requested by Rendra, some specific StreamBIM features will be implemented for testing purposes. These features are *object-selection* and *minimap*. Object-selection will allow users to tap, thus selecting, objects and display information of, e.g., walls, pipes, and beams. The minimap will allow for better navigation and is considered an essential feature of BIM products.
- The app will implement a user-friendly calibration of the building to the environment, i.e., the real world building surrounding the user (AR mapping). The calibration process will consist of scanning an enclosing room and matching the BIM model to that room using the information from the scan.
- The app will be implemented using a shared UI framework. A shared codebase will need to accommodate native bindings with both platforms and manage the platform-specific modules as views.
- Conduct user tests and surveys to assert the value of the implementation. The tests will be conducted by a variance of people who has work experience with the construction industry and BIM.

1.2 Background

1.2.1 The Current State of BIM

BIM as a concept has been around since the 1970s, but the building industry did not adopt it until the early 2000s [3]. Today, BIM is usually preferred over outdated construction processes; shifting focus away from paper-based architectural drawings to improve coordination and collaboration through a digital medium with 3D models. BIM products have become sophisticated pieces of software that provide a rich set of features and utilities to everyone engaged in the construction process.

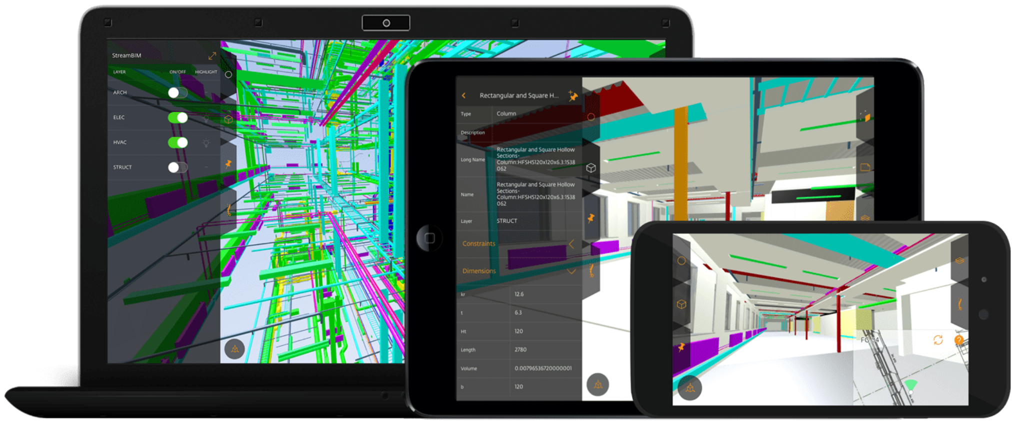


Figure 2: Visualization of different BIM layers on multiple devices (StreamBIM)

The most common approach found in most BIM software today is the visualization of the BIM process in a 3D environment. This approach allows users to observe the different BIM layers of the building from multiple angles, as well as visualizing measurements, materials, and more depending on the product's available features. Figure 2 illustrates how the different BIM layers are visualized in StreamBIM; showcasing the electric and HVAC layer to the left, and the architectural layer to the right. Furthermore, today's BIM software integrates the complete BIM process, which enables users to communicate, document, and highlight problems in a multi-user environment.

1.2.2 StreamBIM

Rendra, the company behind StreamBIM, started as a project conducted by NTNU students in April of 2012. According to an interview conducted with Kristin Omholt-Jensen, the former CEO of Rendra, the main objective was to investigate commercial opportunities with server-side rendering technology. After examining several industries and use cases, they ended up focusing on the construction industry and got an R&D agreement with the Hospital in Østfold the following autumn. This decision resulted in StreamBIM, which has later evolved to be a sophisticated collaboration platform for BIM projects in the construction industry. StreamBIM strives to be a user-friendly BIM platform where the distribution of information, efficiency, and collaboration is in focus; assisting the industry in eliminating mistakes and budget overruns. StreamBIM provides an intuitive user interface, performant streaming of BIM models, as well as many other advanced features. These benefits, as well as smart strategies and good customer relations, have allowed StreamBIM to gain traction in recent years and experience substantial growth in the number of projects throughout 2018.

Currently, StreamBIM has more than 12000 registered users located in Norway, Sweden, and Iceland, and is used by Backe and Caverion, two of the most prominent entrepreneurs in Norway. StreamBIM is also involved in several large construction projects in Norway such as BUS 2, which is the second stage of the children's and youth hospital in Bergen. Much of this acclaim is due to StreamBIM 's excellent user experience that emerged as a consequence of Rendra's close collaboration with BIM users under development. Kristin states that this approach has been of great advantage, as it has allowed for a user experience tailored for on-site workers with tight schedules and inferior technical skills.

JDM Technology Group acquired Rendra in January 2018 and are well known for growing technology companies. JDM affirms that they have an extensive network of companies able to distribute StreamBIM in various countries, which will increase the influence of StreamBIM internationally.

1.2.3 Why AR?

AR is an interactive experience where computer-generated data augment the real world; most commonly by overlaying such data over input from the device's camera. AR is, therefore, a useful and entertaining way of visualizing data that is relevant to the environment around the user. As an example, observing objects and details in the real world would be simpler and more enjoyable if a user could point the camera at the object to get details about it as illustrated in figure 3, instead of looking up the information on an app in the phone. Due to the advantages of AR, there has recently been a significant growth of applications and usages for AR technologies in mobile applications, such as IKEA's furniture app, Gatwick airport passenger app, and several games and entertainment products [4,5].

Figure 3: Use of AR to visualize public places

As discussed earlier in *1.2.1*, the modern building industry visualizes constructions in a 3D environment through BIM software. This visualization, which is based on the real world, makes AR ideal for BIM and it is argued that all areas of the construction process will benefit from AR, including design, construction, maintenance, and renovations [6]. AR will allow designers to make design decisions confidently by observing their design in correlation with the real world; improve construction workers' understanding of buildings, as well as improving their efficiency; make it easier for renovators to locate hidden infrastructure such as HVAC and sanitary [6]. Moreover, several StreamBIM users have shown positive interest in an AR module for the platform, which will allow for the development of immersive and more intractable ways of using BIM.

1.3 Task Details and Scope

1.3.1 Native Rendering

Native rendering utilizes a target platform's native technologies and hardware components to render a scene. In this thesis, native rendering is a requirement set by the limitations of current AR solutions and the rendering methods used by StreamBIM. StreamBIM currently uses web-based rendering, which integrates poorly with native AR solutions and is limited to run on top of a browser. Also, there are no existing cross-platform hybrid AR solutions that would be viable for this project due to their limitations and lacking performance.

Outcome

- Performant rendering
- Provide information about native rendering, as Rendra has expressed a desire to research the benefits and disadvantages that native rendering entails

Goals

- The same camera controls found in StreamBIM with comparable movement sensitivity
- Use StreamBIM's existing data for rendering
- Ability to stream models and render a building while it is streaming/loading

Out of Scope

The streaming of BIM models is the main selling point of StreamBIM and allows mobile devices with inadequate memory to render large BIM models. However, the implementation of streaming is complicated, time-consuming, and is not too relevant in regards to the thesis' scope. Therefore, streaming has initially been left out of the scope, so it could later be added if there is enough time after the finalization of prioritized tasks.

1.3.2 Object-Selection

Object-selection allows users to select an object or a section of the building by touching it on the screen. The selection will highlight the object and present information about the selection, which could include, e.g., name, description, type, dimensions, materials, and problems highlighted by other users. This function already exists in StreamBIM and is an essential feature for the users. We hope to integrate this feature to ensure that the user can continue normal work operations without having to leave the application.

Outcome

- Provide information about if and how this feature is enhanced in AR
- A better user experience
- Supply the user with all required information on-demand

Goals

- Find the correct object that is being selected and retrieve information about the object using the StreamBIM API, then display that information
- Must work in virtual and AR mode

Out of Scope

Object-selection is a complex feature in the StreamBIM app, as it in addition to providing information about an object also enables users to communicate and upload images about the object. As this task mainly centers around the benefit of selecting objects in AR, this feature will be as simple as possible. For this thesis, the selection will only display an object's name, type, and description as a proof of concept.

1.3.3 Minimap

The minimap is an essential feature that allows users to navigate the environment. The minimap projects the 3D environment as a 2D map seen from above and marks the user’s position. The map is draggable, moving the virtual camera in relation to where the user’s position is moving. This feature allows the user to navigate more quickly around the building to find the correct location quickly. The 2D map is based on the floor height; as soon as the user changes the floor in the virtual environment, the map has to change accordingly.

Outcome

- Simplistic and intuitive navigation in the app
- Recognizable navigation for StreamBIM users
- Provide information about how this feature can be implemented in a native app

Goals

- The same minimap controls found in StreamBIM with comparable movement sensitivity
- Use StreamBIM’s existing tile-server and floor image data
- Must be dynamic, i.e., display the correct floor plan in regards to a user's lateral position.

Out of Scope

It is possible to implement a minimap that is shared between the platforms in a unified UI. However, this is out of scope for this thesis, and the minimap will be platform-specific.

1.3.4 AR Mapping

AR mapping is the process of finding a transformation to apply to a rendered model to make it seem in-place with the real world in AR. In our case, we are mapping a room of a rendered BIM model to the same room in the real world; a user can then move throughout a building under construction and see the rendered building overlayed correctly over the unfinished building, which gives the impression of being inside the rendered building. AR mapping is the most crucial process in the application, as it is the gateway between virtual and AR mode and ensures that the building is overlaid correctly; the AR feature is useless if this process is complicated, slow and inaccurate.

Outcome

- Good user experience
- Accurate, fast and uncomplicated mapping

Goals

- The same user process on both platform
- The process must work reliably and in most buildings

1.3.5 Shared application framework

The native-specific code, mainly rendering and AR on both platforms will be integrated into a shared codebase; based on an application framework. Integrating with an existing cross-platform framework that supports native bindings gives the developers the ability to focus on the platform-specific code, and develop the UI in union. Performance is critical on the 3D side, which means native-code is required. the existing frameworks for a shared codebase provide good enough performance on the UX side.

Outcome

- Reduce the allocation of resources needed to develop overlapping code.
- Reduce time spent designing and implementing platform-specific UI components
- Unified design language for both platforms

Goals

- Make the native codebases implementable with the selected Framework
- A uniform and consistent API for both native codebases
- Merge the two platform-specific codebases into one shared codebase

1.3.6 User QA and Testing

QA is a process in which mistakes and bugs produced during the development process are reduced. This process is performed at an early stage of the process and every subsequent stage in the development process. Combining QA with UAT, which is also called a beta, will ensure that the developed product is right for the user and works as the user expects. During the UAT phase, the application is tested with users from the correct audience. This stage will be the closest we get to real-world environments and use cases and allows the developers to iron out the last defects in the application before releasing it to the public.

Outcome

- Reduced the risk of an unused product
- Ensure the quality of the software is up to par with StreamBIM
- Determine the value of an AR implementation in StreamBIM

Goals

- Lower the risk related to development and release for Rendra
- Create a viable product for Rendra
- Create an enjoyable user experience; that is simple enough to where the construction workers will use the application for their daily work.

1.4 Expectations

1.4.1 The Team's Expectations

The team's initial expectations were to be able to finish a working AR module for StreamBIM on both Android and iOS devices using native rendering. Initially, the team expected to spend a large part of the development process making the rendering process work as intended in a 3D environment, and a large part of process managing to anchor and translate the rendered building in an AR view. Regarding time management, the team expected to have a working prototype at least a month before the deadline for the thesis, to be able to spend some time working on unexpected problems and documentation. One of the team's goals was to have an excellent working product which would result in a top grade for the assignment. This is why the teams chose such a large and complicated task.

1.4.2 Rendras's Expectations

Rendra did not expect a fully functioning AR implementation going into this collaboration. However, as AR is a challenging and high-risk project for Rendra, they expressed a desire to obtain as much information about the process, solutions, and challenges of developing an AR implementation going forward. In the best case, Rendra was hoping to see the collaboration result in an exciting demo showcasing a StreamBIM model visualized correctly in a physical building. Summarized, Rendra wanted a proof-of-concept and informative insight into the process and experience of building an AR solution for the StreamBIM platform.

In regards to the technical aspect of the project, Rendra was hoping that the application could be consolidated into a cross-platform codebase for both platforms. Additionally, Rendra wanted the platform-specific code to be as similar as possible. Rendra bases these wishes on the resources needed to continue complicated development on two different native platforms

2 Project Outline and Requirements

2.1 Existing Products and Research

2.1.1 Competitive Products

The construction industry is on the rise for tech startups, and a lot of new products emerge monthly. Most products struggle with their mobile solutions due to the share size of the models. Architects and owners have had access to their BIM models for years through desktop software like Revit and other BIM tools. With the new trend of empowering the workers and putting the models in their hands, a race between the BIM companies has emerged, and they aim to provide the best and easiest solution for their customers. AR is a very new technology in the mobile industry, and the business just recently started developing BIM oriented AR applications.

TwinBIM by Dalux

Dalux is a company located in Denmark and has worked with the construction industry for over 13 years; making them Rendra's biggest competitor within the BIM market. Dalux has a wide range of product for the construction industry and offer products similar to StreamBIM as well as a solution named TwinBIM, which is a cross-platform AR solution offered as a free feature for all its users [7]. In conversations with a user of TwinBIM, it emerged that the solution was challenging to use, did not work with larger models, and generally contributed to bad user experience. As of writing, the TwinBIM application does not support model streaming. The users are required to download the BIM models to run the application.

Gamma AR

Gamma is a BIM product developed in Germany and has a feature called Gamma AR which implements AR in much the same way as TwinBIM [8]. Gamma is a smaller company with less recognition than Dalux. Gamma shares much of their feature set with TwinBIM, neither support streaming of IFC models and their alignment process is simplified and very manual work. With Gamma being one of the new competitors on the scene there is barely any information about the company and the product itself on the internet; however, they are far along when it comes to the speed and the functionality of the application.

2.1.2 Related Work

We found it hard to find useful resources and examples of existing implementations or thoughts around this subject. The construction industry is very outdated technology-wise, which makes what we do unique. Very few companies have tackled AR for BIM, and the solutions we have found have had very poor adoption amongst the construction workers. We believe this might be due to immature tech and the low usage of existing technology in the field.

Utilizing BIM with AR and Location-Based Services [9]

This Masters thesis talks about the possibility of implementing an AR-based system. Back in 2013, the technology was limited, and the only way to accomplish such a task was to do most of the calculations on the server side. The server had to run image processing and send object locations back to the client. This is done through a technique called Lifecycle Building Card [10]. However, today, a solution like this would not be feasible; the clients' devices are stronger than before, and the camera and sensors are more accurate than previously. We rely on fast updates, and because of network limitations on the building site, the application needs to work both offline and online.

BIM with Microsoft hololens [11]

This Masters thesis implemented a BIM AR application for Microsoft's *Hololens*; using the embedded cameras, they made a proper representation of the building to map the BIM model against this representation. In this project, they used Unity as Hololens supports it. The application itself required the models to be implemented inside of the application with no support for streaming or downloading models on the go. There are some issues with the solution presented in this thesis; they have not solved the problem of rooms looking the same ex: hospital rooms.

2.2 Technologies

2.2.1 Rendra’s Existing Technologies

StreamBIM uses a lot of different technologies and libraries bundled together to bring the application to the market. We were required to implement some of these alongside their API so that the experience would be the same on the AR application as it was in their current web application. The backend and API are primarily written in golang and is using JSON:API as the API standard [12]. Rendra also follows a few other specs for streaming of their data that we had to follow to be able to read and parse the 3D data.

JSON:API

We worked with the Rendra’s JSON:API endpoints defined in appendix (See Appendix A). REST has no standard when it comes to how the API response is formatted, the goal of JSON:API is to define a common syntax for responses. A client built around this concept will implement more aggressive caching, and eliminate network requests. Through linked objects and relationships, the client will be able to fetch all the data associated with the response from different sources. Rendra uses this with Ember JS which provides a wrapper around the JSON:API client. The web app can ask for a resource and request all the associated data to that resource, with no extra code required. Due to the sparse JSON:API client in Ember JS Rendra has not been able to implement compound documents or sparse fieldsets. This means that even though it can fetch the resources, it will have to perform multiple requests to do so. During our research, we found no viable JSON:API client for iOS and Android and decided to treat the API as REST.

Protocol Buffers

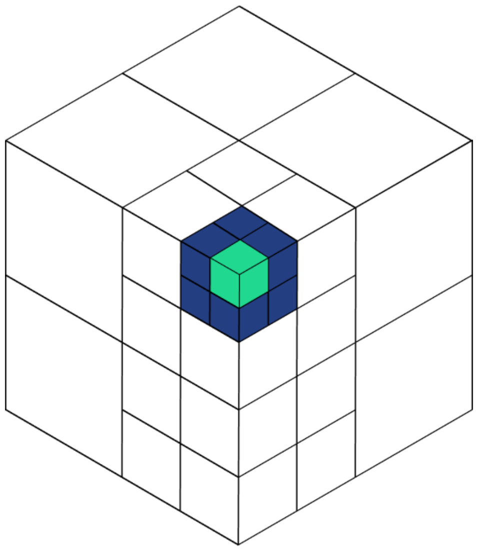
Protobuf (Protocol Buffers) is a language-neutral, platform-neutral message protocol developed by Google [13]. It can encode a variety of data into an optimized response. To parse this data, we required the proto file written by Rendra. When this is run through a compiler, it will output an encoder/decoder for the language in question. Rendra uses these proto files to send the .obj file data. They do this to speed up the process of loading 3D models and because they combine and add extra information about the layers. With these things in place, a request to a Protocol buffer endpoint is just as easy as fetching data from a JSON:API.

Materials

For coloring and shading Rendra provides an API endpoint to download the materials used in the BIM model layer. The data from this endpoint is a standard wavefront material file. With this file, we can give the 3D nodes the right colors and light. This is further explained in *3.1.1*.

Octree Data Structure

Rendra uses an octree data structure for 3D positioning of elements. An Octree is a Node where each node can have up to 8 children nodes [14]. This will result in a tree-like structure of nodes with 3D objects attached to it. Each node contains a bounding box, information about what protobuf file attached to the node, and the type of the node. A node can be of different types and are handled differently by the rendering engine. Some objects will be instanced while others might not contain instancing at all.



A closer look into a octree node from the StreamBIM API

```json
{
  "info": {
    "created": "1558159369",
    "version": [4, 4],
    "type": "hybrid",
    "files": ["325276.obj"],
    "detailLevels": ["oct_", "box_"],
    "translated": [...],
    "density": 0.35618538,
    "numberOfNodes": 9,
    "numberOfNodesWithGeometry": 9,
    "depth": 2
  },
  "id": "0",
  "bbox": [...],
  "gbbox": [...],
  "tris": [1178, 1130],
  "insts": [0, 0],
  "density": 0.56859505,
  "children": [...]
}
```

Figure 4: Octree structure

2.2.2 Programming Languages

Choice of programming languages for the project was decided based on the requirements of the finished product. The product must be running on mobile devices and must be able to run and perform well in both 3D and AR mode. This limits the choice of languages to the native languages of the two mobile platforms with support for AR: Android and iOS. For the Android part of the project, we decided to use Kotlin over Java and other languages. Compared to Java, Kotlin is more concise, safer, and has better support for functional programming [15]. For the iOS counterpart, we decided to use Swift over Objective-C because of Swift's benefits in performance and ease of use [16]. The two languages were chosen in tandem due to the fact that they are very similar resulting in a codebase that can be understood by both teams.

2.3 Rendering

There are multiple ways to implement native rendering in an application; however, it is often dependent on the situation of the project. For example, a simple application rendering 3D assets stored on the device would most likely benefit from using an existing rendering solution, as this is the typical use-case. In contrast, this app use streamed geometry data and may later implement the advanced rendering features found in StreamBIM, such as clip-planes and custom shaders. To display the 3D geometry, we have no use for advanced shaders and have therefore not included any shaders at all in the project code. Rendra does much work with clipping and measurement, which requires the use of shaders; it is necessary that our chosen technologies support adding this feature in the future.

2.3.1 Custom Rendering Engine

It is a viable option to build a custom rendering engine for the project. The engine could use the shaders and rendering techniques found in the StreamBIM app, even if it uses web technologies. A custom engine would be made greenfield, thus allowing it to be tailored to this specific project. Furthermore, it would enable StreamBIM to remove dependencies on rendering libraries such as ThreeJS as well as increase performance in their web app; The engine could be compiled to web-assembly and implemented in the web app, improving performance and lower the bundle size of the application.

Currently, a complete cross-platform rendering engine would not be possible to build, due to Apple’s exclusion of OpenGL ES on its platform in favor of its low-level high performance rendering API Metal. Apple thought OpenGL was missing the core integration with the operating system, hardware, and software thus decided to take a clean-room approach and design a graphics API for Apple’s hardware. This result increased the number of draw calls being made by 10x compared to OpenGL ES [17]. With this change came some changes to shaders and how they work as well requiring developers to learn Metal shaders and adapt the existing shaders to work with Metal. To resolve the rendering API issue, a wrapper in C has to be written to work as an intermediate layer between the two platforms. API calls to the rendering engine would be abstracted and there would be an identical rendering API on both platforms. This would make the rendering code between the platforms similar. With this in place, the only difference between them will be the shader library. This API is desired by both the team and the project owner.

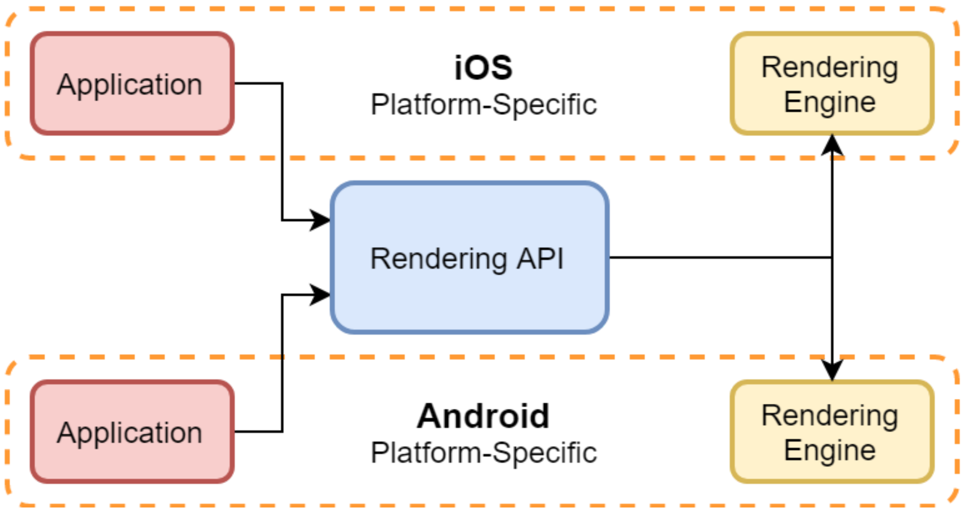


Figure 5: A uniform rendering API for both platforms

Exploring the Concept

The team conducted a minor prototyping session during the planning phase of the project. The objective was to build a basic rendering engine from scratch. The session yielded an educational insight into rendering concepts and the process of developing such an engine, as well as creating a basis for comparison. The rendering engine was not as performant as we would like, and it showed us the amount of work required to make a complete engine from scratch. With the results from the prototyping session, we decided against creating the engine from scratch. The time constraints we had made it hard for us to set aside the time required to make an engine that was viable for AR.

2.3.2 Existing Rendering Engines

After evaluating other 3D frameworks, it would be useful to analyze already existing rendering engines and whether or not they would suit our needs. The main problem or skepticism we had when it came to using existing rendering engines is that they are primarily suitable for games [18]. This would not necessarily have been a huge issue, but it would entail building on top of an extensive and complex rendering engine, with various features we would not have any use of such as different types of lighting, textures, and physics. Therefore it would seem logical to develop a rendering engine from scratch which only handles our specifics needs which can scale and be extended as feature needs arise. Keeping that in mind, there are a few rendering engines which could have been relevant for us to use.

For Android, the most logical choice would be the new rendering engine developed by Google called Filament. Filament is a performant rendering engine which has a significant focus on rendering techniques such as lighting, textures, and materials. Filament would be a viable choice for us because of its continued development by Google, ease of use and performance.

While Google has been pushing its rendering engine for Android rendering, Apple has only been recommending SceneKit for iOS rendering. It is possible to use Filament on iOS also, but this is very experimental. Excluding Filament, there are a few free options for cross-platform and iOS development such as Antiryad Gx, BatteryTech, Corona SDK and EdgeLib. The main disadvantages with these engines are that they are primarily optimized for game development and as we discussed earlier, this would entail including a vast library with a lot of unneeded features into our project [19]. These existing rendering engines are also challenging to customize for our specific use cases which could make present and future development more difficult.

2.3.3 Existing 3D Frameworks

A 3D framework is a high-level layer built on top of a rendering engine and includes features and abstractions related to the creation of comprehensive scenes. Therefore, 3D frameworks usually include a scene graph, camera controls, and physics, but also incorporate additional tools for building realistic scenes, e.g., particle systems, lighting, and animations. While there are many viable 3D frameworks for this project, the selection was narrowed down to the two official 3D frameworks for each platform, as well as Unity. The choices were limited because Apple, Google, and the only community highly recommend the official frameworks. Unity was discussed online as a possible cross-platform competitor, and its viability piqued our interest.

SceneKit for iOS

SceneKit is a feature rich 3D framework developed by Apple and is designed as a high-level, easy-to-use framework for iOS devices. SceneKit includes all the essential features of a 3D framework in addition to various features such as material shading and geometry manipulation. These additional features allow models to be created dynamically using vertex data and custom materials, which makes the framework highly compatible with the data used on the StreamBIM platform. Furthermore, SceneKit is extremely performant, which is a requirement in this project. For future development on the iOS platform, scenekit can be kept as it also exposes the Metal API, thus giving the developer the ability to write as low level as required. This will yield a performance benefit in the long run once shaders and custom rendering programs are introduced.

Sceneform for Android

Similarly to SceneKit, Sceneform is a 3D framework with the essential features needed for this project. The framework is developed by Google and was initially made exclusively for AR apps. However, Sceneform could be used as a rendering-only solution in non-AR apps as of September 2018. Sceneform is built on top of Filament, which ensures a performant experience on most Android devices with various specifications. Sceneform achieves this by adaptively throttling resolution in favor of a higher refresh rate. Filament provides access to the underlying OpenGL ES engine allowing shaders to be added to the rendering loop. One of the downsides is how

Android manages memory for their applications, so an NDK supported rendering engine and scene graph would need to be implemented to improve the overall Android experience.

Unity

Unity is first and foremost considered as a game engine and is used by game developers to create advanced games. Unity is one of the most popular game engines today and is expected to experience a growth rate at over 39% over the next two years. Furthermore, Unity is used by 90% of AR/VR companies worldwide [20]. Unity provides most of the same features as both SceneKit and Sceneform, as well as being a cross-platform solution with native AR integration on both iOS and Android. These benefits make Unity a compelling option for this project. While providing a very high-level API for both platforms in C#, Unity also has the ability to hook in to each platform giving the developer more fine control over the rendering process.

2.3.4 Conclusion

All the presented solutions are viable options for rendering in the project. However, SceneKit and Sceneform were chosen as they are high-performance and minimalistic solutions that provide the features required for this project while being scalable. Also, they are the official 3D framework solutions for their dedicated platform and has direct integration with the native AR solutions for their platform.

While a custom rendering engine, if developed properly, would yield a great tailored rendering solution as well as the benefits discussed in *2.3.1*, it would introduce the risk of consuming too much of the limited timeframe allocated for this project. Also, this thesis focuses on the AR aspect of the project and not the rendering. Therefore, this option was eliminated in favor of using an existing solution.

The argument is the same in the choice of using an existing framework or building a custom framework on top of existing rendering engines. All the 3D frameworks presented have the features needed to carry the project forward, and a custom framework would therefore not provide any additional benefits on the cost of consuming time.

In many ways, Unity would save time already being cross-platform and provider a richer feature set than SceneKit and Sceneform. However, using Unity would introduce a radically new element to the StreamBIM technology stack as well as include a large redundant feature set. Furthermore, Unity is known to consume large amounts of memory; memory is essential in this project as BIM models need a substantial allocation of memory. Being limited by device models and hardware, we decided to try out Scenekit on the iOS platform. After working with Scenekit, we found that Scenekit's integration with ARKit to be so beneficial that we wanted to try out Sceneform on Android as well.

2.4 How to Implement AR

While there exist multiple AR solutions, research showed that there were no cross-platform AR solutions viable for this project; a combination of React Native and libraries such as *react native ARKit* and *react native ARCore* is possible, but these implementations are inefficient and highly experimental [21]. The most common approach to implementing AR in apps today is by using the native AR development kits for both platforms. These kits are *ARKit* for iOS and *ARCore* for Android. Fortunately, the rendering solutions chosen for this project provides direct integration with the kit for their respective platform.

ARKit

ARKit is an AR module developed by Apple for iOS devices, and this project will implement the second version of the module named *ARKit2*. ARKit2 adds additional features over ARKit, e.g., simultaneously shared AR experiences, persistent tracking, and improved object detection. Moreover, the newer version enhances performance and general AR functionalities [22]. ARKit combines various methods of tracking to ensure an immersive AR experience. These methods include device motion tracking, camera scene capture, and advanced scene processing. ARKit uses these methods to map and track the real-world user space and match it to the coordinate space rendered by SceneKit. This mapping allows objects rendered within SceneKit to be displayed correctly over the real-world userspace [23]. ARKit uses the *visual-inertial odometry* technique to create this mapping, which is a technique that calculates the 3D translation and rotation of a moving camera relative to its origin, using visual features such as interest points.

ARCore

ARCore is Google's answer to Apple's ARKit and is a platform for building AR experiences. Similarly to ARKit, ARCore is also feature-based and tracks the device's motion using *concurrent odometry*. ARCore further uses this information to control a dedicated Sceneform AR camera and maps the real-world user space to the Sceneform coordinate space, much in the same way as ARKit [24]. Moreover, ARCore also provides light estimation, which allows for the estimation of current lighting conditions in the environment. Light estimation makes a scene more realistic by allowing rendered objects to interact with the lighting such as shadows and colored light. ARCore also provides the ability to create collaborative shared AR experiences like ARKit through ARCore's Cloud Anchor API. With Cloud Anchors, environmental features are shared between devices, enabling apps to render the same scene simultaneously.

Google announced ARCore just weeks after Apple's release of ARKit, which was around the time Google released its previous AR project *Tango* into beta. Tango is a more advanced version of ARCore targeted for devices with advanced AR-specific hardware, such as the ASUS Zenfone AR. While ARCore works perfectly on a table and other flat surfaces, Tango excels at mapping more extensive or irregular shapes, specifically rooms and buildings. Tango actively uses specialized hardware to create a 3D environment instead of tracking planes and surfaces, which improves its ability to keep objects in place and remember large mappings. However, Google stopped support for the project in December of 2017 in favor of ARCore, which gave AR to more users without special devices. Tango would be particularly suitable for our project, but due to discontinued support, it will not be looked into in this project [25].

Limitations

Both ARKit and ARCore use feature-based tracking, making it reliant on a detailed environment with distinct features to achieve proper tracking; a blank indefinite surface or low lighting conditions will, therefore, reduce their ability to track correctly. This is a limitation of most AR integrations today, as feature-based tracking is the most common solution. Furthermore, occlusion, the process of concealing rendered objects that should be behind real-world objects, is not a feature provided by SceneKit nor Sceneform. However, occlusion is not a critical obstacle in this project, and it will most likely be addressed in the future with further advancements in AR.

2.5 Architecture

Each platform's architecture is centered around their best practices, yet keeping the code and structure as similar as possible on both platforms. The architecture chosen for each platform was well defined and researched before we started the project, and lead to the creation of a set of rules to follow during development to ensure that the end product was compliant to the spec defined in the project plan.

2.5.1 Android

The high-level Android architecture is based on the official recommendations from the Android developer website, which is followed alongside other acknowledged best-practice patterns for Android. This ensures that the app is robust, scalable, in spec, and that onboarding new developers will be easy.

The architecture follows the *separation of concerns* principle which, according to the official Android architectural guide, is the most important principle to follow [26]. By following this principle, the app is split up into modules focused on a specific concern, e.g., UI, state, and data fetching. This allows for more manageable integration- and functionality tests at a later stage, as well as providing a satisfactory user experience and manageable app maintenance.

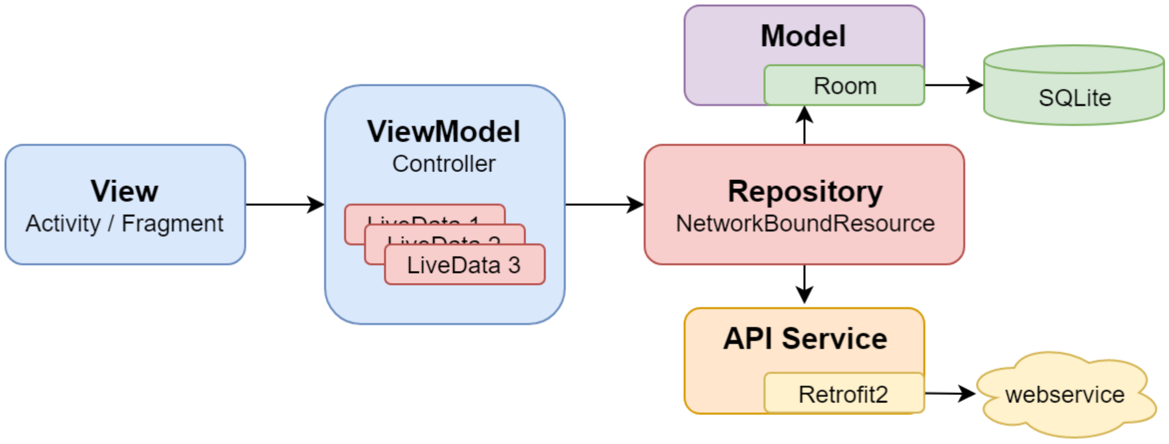


Figure 6: The official recommended app architecture for Android

The main recommended way of communication between modules is by using the LiveData implementation, which is a lifecycle-aware data holder based on the observer pattern. LiveData is a simple class that contains data that can be observed. When the data is changed, the class notifies and forwards the new data to its observers. Being lifecycle-aware, LiveData will only notify active lifecycle observers, known as LifecycleOwners such as Views. However, LiveData can be observed forever, which is useful if the observer is not a LifecycleOwner. A Resource class is often used in conjunction with LiveData, which is a generic class that contains data and a status, e.g., loading, success, and error. Using this in conjunction with LiveData makes it ideal for asynchronous data fetching.

View

Views are modules that manage the UI and often represent pages in the application, which is in Android known as Activities. However, Views can also represent smaller modules of the UI, called Fragments. A View is responsible for drawing and presenting data to the user, as well as handle user interaction and managing system communication such as permission requests. Views are owned by the OS, which makes them vulnerable to destruction based on user interactions with the OS or system conditions such as low memory. Therefore, it is advised that the app stores its state outside the view in a module that is owned by the app [26].

ViewModel

ViewModels are modules that manage the app’s state, supply Views with their data, and contain the logic to communicate and manipulate that data. A ViewModel is usually assigned to one specific View but does not know about UI components, making it unaffected by configuration changes resulting from recreating activities after, e.g., device rotation.

Repository

Repositories are modules that provide a clean API for providing data and handle data operations. A Repository is not attached to a specific View nor ViewModel but focuses on a specific domain, e.g., user, project, building.

Repositories handle data fetching independently, which means that they individually manage the data-fetching method. The recommended way of fetching data is using NetworkBoundResource, which is an implementation that combines persistent storage through SQLite and Services. This implementation caches data in the persistent disk on the device and uses the cache as a placeholder. This makes the app more responsive, especially at lower internet speeds. When a ViewModel requests data, the repository invokes a call to a NetworkBoundRequest, which returns a LiveData containing cached data. When the updated data is retrieved, the new data is cached, and the NetworkBoundRequest updates the LiveData with the new data; this, in turn, is observed by the View or ViewModel.

However, NetworkBoundRequest, as is, can not be used in this project alone, as the app fetches an immense amount of BIM and geometry data. As caching multiple gigabytes on a device is unfeasible, NetworkBoundRequest has been extended to allow data-fetching through a remote data source or cache only. This extension allows the app to keep all data-fetching through the NetworkBoundRequest implementation, which avoids different implementations.

Model

Models, or Room entities, are objects that hold data and are based on Room, which is a library for abstracting SQLite and provide database access on the device. Models describe how data should be serialized and stored in a database, but can also be used as data objects throughout the application.

API Service

The API services are modules that focus on fetching data from a remote data source and are, alongside repositories, focused on a specific domain. The services utilize Retrofit, an HTTP client for Android, to communicate with the StreamBIM REST API. Retrofit has multiple converters to parse formats such as JSON and XML to Java objects, but most importantly, it has a converter for protocol buffers. This making Retrofit ideal, as StreamBIM use protocol buffers for the geometry data.

2.5.2 iOS

Apple provides little to no best practice guides to follow for iOS swift development. They released something they call Apple MVC, which is a modified version of the MVC architecture. The core concepts of this is good but it is a little outdated compared with the growth of the Swift language. Swift and iOS development is very community-based and therefore lacks great resources to follow for the architecture outline. However, with Swift 4, there are some new patterns to use which we chose to implement alongside Apple MVC to ensure that the structure of the code was up to par with Android. We tried to model the architecture to be closer to Android instead of just relying on the standard MVC pattern.

Doing this we distinguished between Controllers, View Controllers, Models, Views and Networking. Splitting the Networking and data fetching into its own segment allowed us to make use of the powerful protocol syntax in Swift, and keeping the models strictly models holding data. A lot of the structure and the best practices followed in the application have been from John Sundell [27]. John is an avid Swift developer and has set the bar for how to develop on the iOS platform.

The main goal with the architecture is to promote a quick development process and eliminate bugs, while still keeping the system simple and easy for new developers to get started with the project.

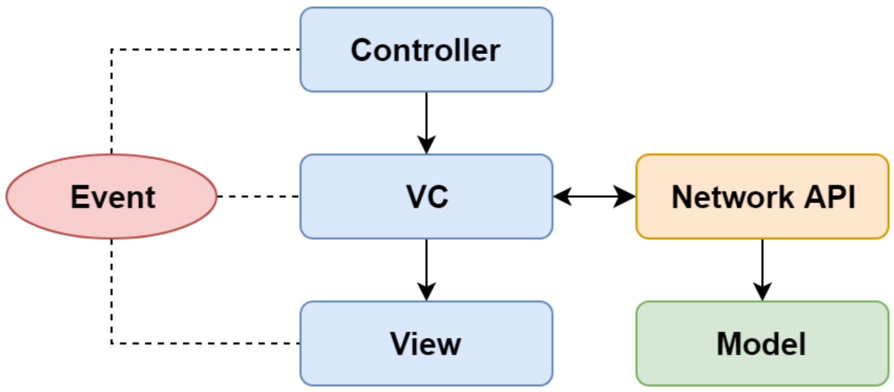


Figure 7: showing the base structure for the iOS architecture

Controllers

The job of the controllers in the iOS application is to structure and handle the dataflow inside of the application. This creates an isolated flow that ties multiple view controllers and models together to make the application work as intended. The controller holds the state of the current view and will mediate between view controllers and pass data to them if they require some shared data. Controllers often pass itself as a delegate down to the view-controllers allowing the view controllers to communicate with the main controller through events. This is used so a view controller can trigger events application-wide instead of being isolated to the view itself.

View Controllers

A view controller will hold all the data for the view it's controlling. It has its own set of views that it will handle and can also be a subview controller. Splitting each screen up into multiple view controllers allowed us to focus on the core functionality of each view controller and creating reusable view components for the iOS application. Some components have been used in multiple locations throughout the application and ensure consistency in functionality and quality throughout the application.

Views

Although most of the application was initially designed using the storyboard, we decided to transition into using shared framework as the UI layer for the application. We had to refactor the storyboard elements into code-form. With each view extending the base UI components, we could create reusable view components in code. The view itself is considered a dumb component and will only act upon the data passed to it. The events produced by the view will be handled by the View controller implementing the view.

Models

The main function of a model in iOS architecture is to hold data. A Model has no way of fetching the data, instead the main purpose is to serialize and deserialize data for the network layer. Most models extend the Codable protocol to work with different encoders and decoders in Swift. The protobuf model is built from a proto file extending the Protobuf protocol.

Protocol based networking

The APIClient protocol controls data fetching inside of the application [28]. It is a generic protocol that takes in the required request, decoder, and a completion callback. By default, the APIClient lives on the main async thread. This is to boost performance and load times for the data that is required to load as soon as possible for the user. Protobuf and larger requests live under the global user-interactive thread. Being outside of the main thread improves the user's experience in the application. Once the data is processed and ready to be displayed, it's fused with the main runtime.

User defaults

Swift and iOS come with a good way of handling user defaults out of the box, but it was never built for ease of use. The user defaults presides under the standard application namespace and can be fetched based on keys for these defaults. To solve this issue we wrapped a class around the user defaults objects provided by iOS. The main function of this wrapper is to ease the use of UserDefaults. We try to keep it as easy as possible and link them to data models. This gives the developer the ability to store data models in the cache. The cached objects can be fetched with the name of the Model.

Events

With inspiration from the likes of redux, the team set out to develop a fully detached system; We chose to go with an event-driven architecture for the application. A controller or View Controller can create events that other parts of the system can listen to. This created a less cluttered codebase and a simpler debug process. This is primarily done with the minimap and the 3D camera itself; When moving either camera around they will send out an event telling the rest of the listeners that it needs to update.

Events are running outside of the main thread to improve the experience for the user, however, this can result in a small delay between the action is performed until the other components update. Testing the difference between a closely integrated loop and a separated one for 3D view and minimap showed little to no impact in performance, thus being safe for use in something that needs to update quickly.

2.6 Work Methodology

We decided to use an agile work methodology since it provides flexibility in the development process in addition to the project itself being difficult to plan statically caused by unknown variables in the team's experience and knowledge regarding technologies such as AR and rendering. Of the various different agile development frameworks, we decided on using Scrum as our main development method. Scrum is a lightweight framework which fits small teams from three to nine developers [29]. The team members are also very familiar with the Scrum development framework from earlier projects.

We have also been using Git as a tool for collaborative work, version control, and issue tracking. When using Git, we have been using a workflow known as Gitflow in combination with feature branches for consistency and structure. "Feature branches" is a concept within Git branch modeling where branches are named based on features. Gitflow is a tool for branch modeling in Git which eases parallel development and collaboration by enforcing feature branches in addition to having a *development* or *staging* branch which the team merges their respective feature branches with. Only at the end of a sprint will the *development* branch be merged with the *master* branch.

The Scrum method has been followed in a manner to ensure maximum efficiency. There has, therefore, been some methods within the development framework which has not been followed such as an official Scrum master. We have also been using external tools to assist the development process such as an online Scrum board. The online tool *RealtimeBoard* was used for managing a Scrum board, roadmap, burndown charts, sprint planning, agendas and diagrams such as Gantt diagrams when needed. Furthermore, have been planning and executing two-week sprints, which is one of the most common sprint cycles.

2.7 Unified UI Library

We quickly decided using a unified UI language was something that was required for this application. The extra work required to make sure the UX is the same on both platforms is not worth it in the long run. The user expects the application to look and feel the same regardless of which platform they are on. We looked into the top 3 alternatives for simplifying the UI code and making it into one platform with native bindings for each. The team carefully looked at each of them and benchmarks between them before deciding on the right solution for us (See Appendix B).

2.7.1 Xamarin

Even though Xamarin is slowly becoming less popular among developers it's still a product that should be brought up. Xamarin is a cross-platform platform framework using C# as its core language and native platform libraries wrapped in a .NET layer [30]. It provides near native speeds with C# being a multi-paradigm programming language and a strong contender to the other languages used in native development today. Xamarin does not officially support bindings to Swift, but it is possible to leverage their bindings for Objective-C. Because of the language choices and the trends in the community; Combined with Rendra's existing stack we decided against using Xamarin for this project.

2.7.2 Flutter

Flutter is a new cross-platform UI toolkit using Dart from Google [31]. As a compiled language this is one of the more performant cross-platform application frameworks. It provides native bindings with each platform and good support with open-source components. Flutter comes with a good set of base components to do the normal operations required in an application, while still supporting the use of native platform-specific code if it's required. Dart bindings can easily be set up for native code thus making code sharing easy across the platforms. Flutter provides a good set of developer tools such as live and hot reloading of components, giving the developer a good suite of tools to work with. Flutter also promotes ease of development and design as their main focal points. With support for material design right out of the box and a lot of components pre-defined it is simple to get up and running in a short manner of time. Flutter is still in its early stages and it would, therefore, be too much of a risk to go with this platform during the duration of this project.

2.7.3 React Native

React native has been one of the longest standing and most used Application frameworks [32]. Since its release in 2015 it has gained a lot of traction. React native allows the developers to use Javascript to write the application logic and UI while still using native components. Using a framework like react-native can boost the productivity of a team that already relies heavily on web technology as most of the components can be re-used in the native application. React native like the others, has some downsides to it. Javascript is not the perfect language when it comes to development and it is prone to bugs and mistakes in the code; Therefore using typescript will be critical in the application.

React native supports code from the NPM repository allowing developers to choose between millions of pre-existing packages to use in the application. While flutter might be more performant due to react-native's javascript pipeline; Promotes better code reusability, React native has proven itself to be a viable technology that many fortune 500 companies rely upon.

3 Implementation

3.1 Rendering

The first step of the implementation was to simulate already a simplified version of the existing StreamBIM application. In other words, we needed to create a 3D rendering environment where we could visualize the BIM models with similar navigation as StreamBIM. This is because we wanted to use the 3D navigation to better estimate how to translate the models in AR later. The first step in this process was to get access and parse the BIM models in a format which we could use in a scene graph. To get access to BIM models, we could load them statically, but we wanted to use StreamBIM's API to better present to Rendra how to implement native rendering.

3.1.1 Parsing Data from StreamBIM

StreamBIM utilizes their backend which supplies BIM models from the previous format called IFC to a new format called protobuf. The StreamBIM API also supplies metadata about the object within the models such as material information and various rendering data such as submeshes and instances. A submesh is a method of separating objects based on materials, and instances is a method to re-render already rendered objects based on position. Using instancing there is no need for the vertex data every time the object repeats in the model.

The first data which was parsed was the material information files for a building. The material files follow the wavefront format so there was no need to use any external libraries to parse this information, so a material loader was written on both platforms from scratch. It is also worth noticing that this particular code could be written in a similar manner for both platforms enabling code reuse. A request to a Material file will look like this; Kd and Ks are colors, and d is dissolve.

```
# Rendra Alias|Wavefront exporter v.0.7
newmtl surface-style-12436-tre-bordkledning-st_ende-brun
Kd 0.101647064 0.09847059 0.06670588
Ks 0.09 0.09 0.09
newmtl surface-style-288963-tre-valn_tt-horisontal
Kd 0.39654902 0.19827451 0
Ks 0.05 0.05 0.05
```

Figure 8: A material wavefront file containing material information

The next step was to parse the data required to render the models. This step was considerably larger as it required several API endpoints and different file formats. The first piece of information required to render the data was a file called a manifest file. This file describes the relation of the octrees in a building along with other data such as detail level and bounding boxes. The manifest file is specific to each architectural layer to enable toggling between layers.

The main data source used in the rendering process is the octree files themselves. These contain all the data required to render a model. The octree files include vertex data, indices, merged facets, and other data which are all used by the 3D frameworks to render the models. The octree file uses the protobuf format and needs a special method of parsing. To save time, external libraries were used on both platforms to parse the protobuf files. After this step was done, we had all the data required to achieve a 3D rendering environment for the BIM models.

3.1.2 Scene and Nodes

At this point, we had all the data available to start the rendering process. This part of the project would use all the data parsed from StreamBIM' API and the 3D frameworks on each platform to create renderable nodes which are attached to the scene graphs. A scene graph has a top-level node called a "scene". This scene is used by the 3D frameworks to render data. A scene has a root note and a camera node. The rendering modules purpose is to supply the scene's root node with renderable nodes from the render data. This entire rendering process is based on the architectural layers. To render the vertex data from the octree files, we need to have a reference to both the material file and the manifest file for a particular architectural layer. Therefore, using callbacks, the material file is fetched first, then the manifest file, and when both these files are fetched, the rendering module fetches the octree file and initializes the rendering process.

The 3D frameworks require data which differ somewhat from the data we have parsed from StreamBIM's API. The 3D frameworks require the vertices and indices of a model to successfully render the model. To render materials they also need specific values from the material file, however, the material is parsed in such a way that all of these values are easily available. The vertices and indices are divided in the data, are easily combined correctly and is inputted in the 3D frameworks. When the frameworks have access to all the data available, they will create render nodes which can be attached to the root node of a scene graph. It is also worth to notice that when rendering several octree nodes, it can't be done at the same time on Android. Passing that amount of data to memory causes problems with memory clamps, so to effectively render all nodes, they should be recursively fetched, effectively streaming the nodes one at the time.

The specific code for this process is quite different on each platform. This is mainly caused by the difference in the Sceneform and Scenekit APIs. However, the underlying data flow is very similar. The code differences are unavoidable because of the two different scenegraph libraries, but the code base on one platform can be used to understand the other if there is a basic understanding of both libraries.

Scene Graph

The structure of a scene graph used to render a BIM model.

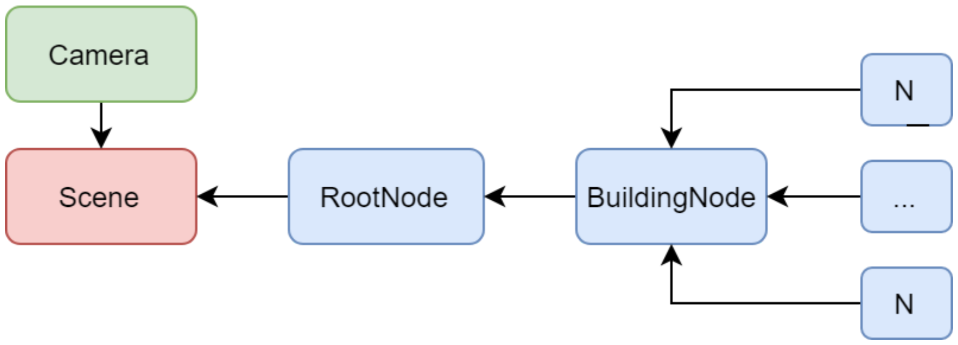


Figure 9: The scene graph architecture

Figure 9 shows how the scene is configured. Initially, it only contains a root node and a camera node which is a viewport for the user. Every node has a renderable attribute along with geometric values to describes its position and rotation in a rendering environment. A node's position can be manipulated using 3D vectors, and a node's rotation is described and manipulated as quaternions. On iOS, however, it is also possible to manipulate rotation using Euler Angles.

In this project, we have additionally added a building node which holds the BIM models which are attached to the root node. This is done to make translations easier when rendering in AR. This is further explained in part *3.2.2* regarding mapping models to the real world in AR.

3.2 AR

When rendering was accomplished, and buildings were loaded correctly, the next step was to implement AR. Implementing AR was a straightforward process, as the rendering solutions for both platforms are highly compatible with the chosen AR solutions; the user can toggle AR on or off with the press of a button. However, this naively places the building in AR without any relation to the real world; the model is moving and tracking correctly with the device's movements, but at this point, the integration was useless as the building must be correctly mapped to the room it reassembles.

3.2.1 Toggling AR Mode

Both SceneKit and Sceneform allows for a simple way of toggling between virtual and AR mode by providing an AR-specific *ARScene* in conjunction with the non-AR scene mentioned in *3.1.2*. By initializing two scenes, one for virtual and one for AR, the root node can easily alternate between the two scenes. The ARScene is mostly different from a regular scene in that the device's camera entirely controls the attached camera-node, as well as that the scene's coordinate space is mapped to the real-world userspace.

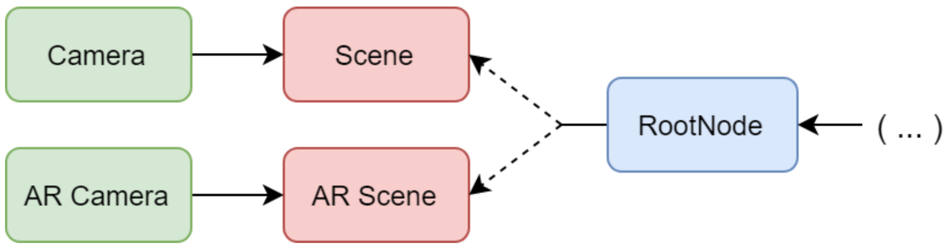


Figure 10: The scene graph architecture with an AR scene and virtual scene

Alternating between virtual mode and AR mode is as straightforward as setting the root node's parent to the desired scene; the 3D frameworks manages all aspects of the process. When the root node is assigned to the ARScene, the building is automatically placed into the real world; the colored background in virtual mode is changed to a video stream of the device's camera, giving the building the illusion of existing in the real-world

```
// RenderingManager
fun toggleAR(visible: Boolean) {
    if (visible) {
        rootNode.setParent(arScene)
    } else {
        rootNode.setParent(scene)
    }
}
```

Figure 11: How the root node is assigned to the scenes in Android

3.2.2 Mapping Models to the Real World

At this point, the model could be placed and appear in-place in the real world. However, the pose of a model was currently static; a building would constantly appear in the same place in virtual mode, as this coordinate space is abstract. In contrast, the ARScene has a coordinate space that is directly correlated with the real world and the device's start-pose; the real world defines the XZ-plane of the coordinate space, and its origin is defined by the device's camera position when the AR session is initialized, as illustrated by figure 12. Moreover, the device's forward start-direction defines the coordinate space's Z-axis direction; the Z-axis' direction is initialized in the direction of the device's camera.

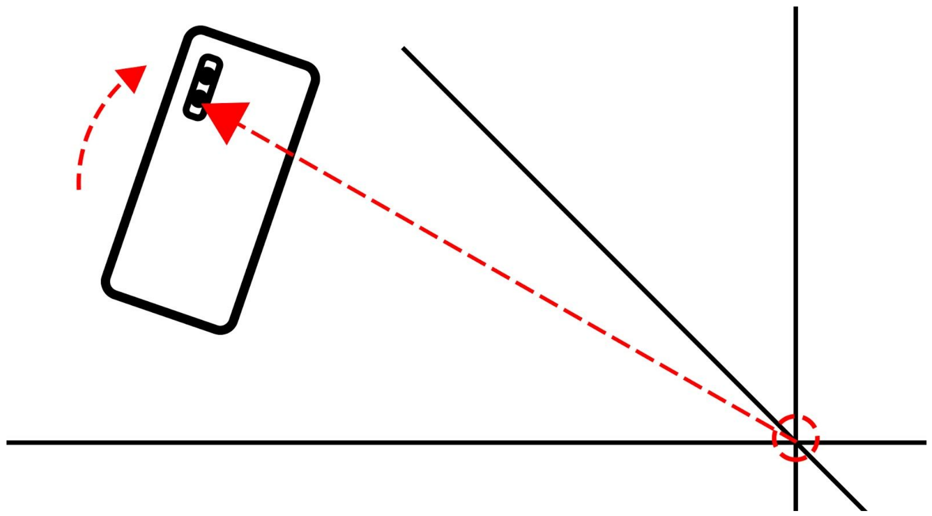


Figure 12: Showing how AR coordinate space stays consistent when the device has moved.

As the models' pose was static, and the AR space is dynamic, a model's placement in AR was decided by where the device started the AR session. This was a problem, as the desired outcome of this AR solution is that the pose of the building is the same as the building in the real world. The model had to, therefore, be applied a transformation to match the real-world building. For such a transformation to achieve this outcome, it would need to contain information about how to rotate and translate the model to be correctly matched with the real-world building's pose.

At first, a direct camera-camera mapping was feasible to implement. A direct camera-camera mapping transforms the building so that the phone's camera, when invoking AR mode, is at the same position as the virtual camera in virtual mode. However, this mapping is highly inaccurate as it is based on the user's calibration, but it is simple to implement and worked as a temporary solution until a world-aware mapping was implemented.

Implementing World-Aware AR Mapping

As a direct camera-camera mapping is very inaccurate, a mapping using information about the real world to align the BIM model was essential. Fortunately, both ARCore and ARKit retrieve information about all detected real-world surfaces, and BIM models usually include information about the rooms' geometry, called *space geometry*. StreamBIM already had this geometry data available on their backend; however, we had to request that Rendra made this information retrievable through their API, as the data was available on StreamBIM's backend only. Rendra made the data available and it was easily parsable to the same data as the detected surfaces by ARCore and ARKit. From here, there were several methods of obtaining a transformation using this information. It is also worth noting that a limitation in Sceneform, which is that there is no difference between a node's local and world rotation, implied that the scene graph had to account for this. The solution was to add a second node for translation only, as translating the root node would shift its center of rotation. This new rotation-specific node is the *BuildingNode* found in figure [9].

The Brute-force Approach

This method first aligns the model, represented in figure 13 with an orange color, to a hypothetical real-world grid calculated using the surfaces that define the room, which is represented with blue; by using the difference in rotation between the surfaces and the coordinate space axes, the model can be aligned with the real-world coordinate system.

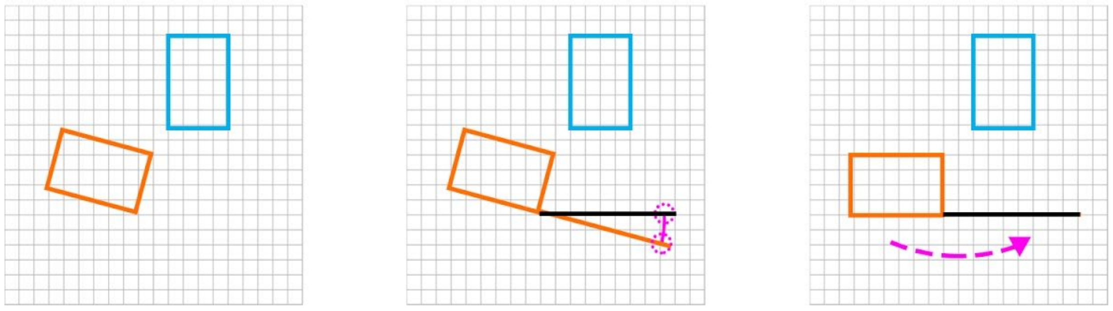


Figure 13: How a model can be aligned with the real-world coordinate space

The method then rotates the model continuously until it finds the most likely rotation based on the number of matching vectors between the room's corners. It then finds the same corner in the real world and the model and translates the model to match the same corner in the real world.

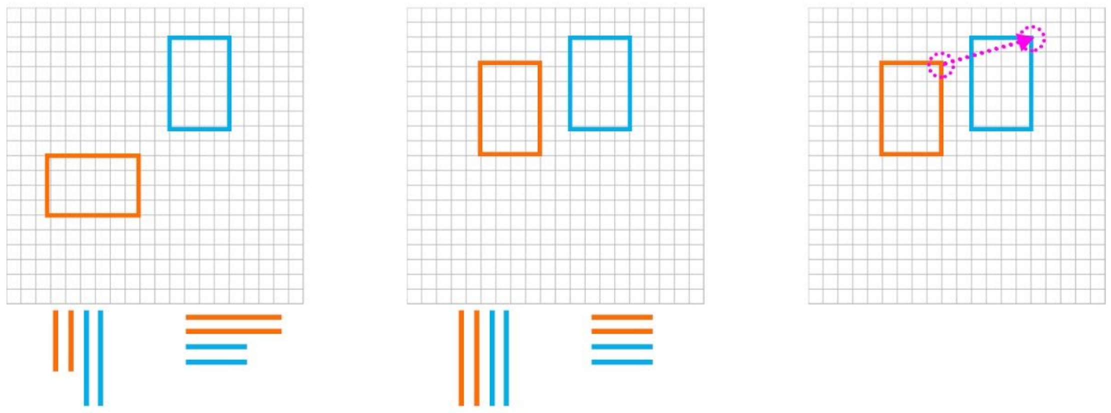


Figure 14: Illustration of the brute-force method's rotation and translation process

This method is nice as the user only needs to be in the same room in the virtual environment before entering AR without needing to worry about camera positions. However, the user also needs to scan multiple surfaces before the transformation could be correct. Also, a room would have to be unique enough to only be correct in one orientation; all rectangular rooms have the same shape when rotated 180 degrees.

The Camera-Aware Approach

This method is much like the brute-force method but uses information about the virtual camera and the device's camera to better determine the model's rotation; the user points the virtual camera in the same direction as the device's camera in the real-world building. This method rotates the model correctly, but the user is still required to scan multiple surfaces before the transformation can be obtained.

The Progressive Approach

This method finds a start-transformation and continuously maps the model as information about the world is gathered; we based the start-transformation on our initial direct camera-camera transformation. The algorithm then iterates over all surfaces detected by the AR module and compares them with the equivalent surface defined by the space geometry. The method then matches the rotation of these surfaces and finds the translations for these surfaces to overlap. The rotation and translation are then applied to the BIM model. This method required us to implement the Observer pattern so that new mappings could be performed when new AR surfaces are detected. Nonetheless, it allows the user to observe the building in AR without having to scan multiple surfaces beforehand, which could make the calibration process appear faster than it is.

Addressing the Methods

We considered all the methods and after implementing and testing all of them, we decided that progressive mapping allows for the best user experience as well as being the simplest to implement. Furthermore, even though the bruteforce method is more automated than the other methods, we discarded it due to its high processing cost for complex rooms and its inaccurate rotation.

3.3 React native iOS binding

Implementing a native binding

There are multiple ways to create native bindings in react-native. The simplest way to achieve this is to create a wrapper around the application as a UI Component. We followed the pattern displayed in figure 15, as it required the least amount of work. React-native does not support bindings for swift, we are therefore required to create an objective-c binding for our swift code. As shown in figure 15, this is the flow required to implement the component for swift. We followed the steps provided by Facebook to accomplish this [33].

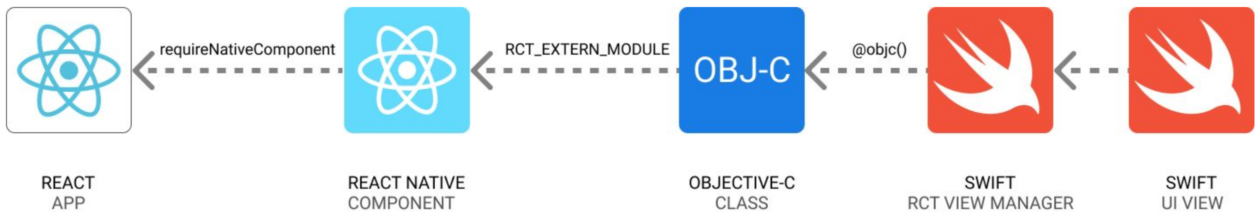


Figure 15: React-Native implementation on iOS

We only exposed the minimum data required to load the 3D models into the application (See Appendix C). To load 3D models; The API needs to know which project and building to target and from which user. The toggle between ar-mode and 3D is done through a flag. We decided to let react-native handle loading and off-loading of the component, meaning there are only two UI states.

With all this in place, we were able to import the external module in our react-native codebase.

3.4 Obstacles and Difficulties

3.4.1 Sceneform

Working with native Android development has mostly been a decent experience. However, there has been some issues specific for device memory and libraries we have used for the Android platform. One of the larger issues the team had to solve was the level of maturity and poor documentation for the Sceneform rendering library. Working with Sceneform has been a lot easier and less time-consuming than developing rendering engines from scratch. However, Sceneform seems to be intended for much simpler use cases than ours; Sceneform was initially intended to be a rendering tool for ARCore and was not able to render a 3D environment on its own until late September 2018.

While Sceneform was initially made for loading static object files, what we needed to achieve was dynamic rendering based on vertices, indices, and material properties. This functionality was poorly documented, and it seemed like no one else in that community had done anything similar to us. We had to figure everything out ourselves using the little documentation that we had available, but we managed to accomplish our goals in the end. In addition to poor documentation, some features for 3D graphics were missing or not yet implemented in the Sceneform library.

The nature of Sceneform and JVM combined also resulted in high-level code complexity. This is not a problem for an end user, but our team's initial requirement was to have some level of code similarity between iOS and Android for code reuse, which was no longer possible in the rendering section of the code. The immaturity of Sceneform combined with our specific use case leads to some significant differences in the rendering method on iOS and Android.

3.4.2 Camera Translation

An early issue that occurred was camera movement. On iOS, this problem was solved rather quickly, because SceneKit has the possibility of using Euler Angles instead of Quaternions. Euler angles are considerably simpler to implement, considering quaternions are one of the most complicated mathematical concepts in 3D relevant programming. The team had no previous experience with graphics programming nor quaternions, which lead to a faulty implementation on Android. The camera did not translate correctly after the building had been rotated. There was also no way to differentiate between local positions and world positions in Sceneform at the time, nor local rotation and world rotation. This combined with our lack of 3D graphics knowledge, lead to the team implementing a workaround for this issue. Instead of moving the camera, we moved the building since the building nodes had a viable local position and rotation.

3.4.3 Minimap

A tile map is a collection of tiles or images mapped in a specific order. StreamBIM uses their API to fetch tiles for floors in buildings and then uses the OpenLayers map library to stitch the tiles together and handle navigating using the minimap module. In our project, we chose not to use a map library, mainly because OpenLayers does not exist for mobile development and we did not want to include a more extensive map library for handling tiles. Therefore we implemented a tile service on Android and a tile module on iOS which has three main tasks to do. Firstly, they fetch the correct tiles for the current floor in the current building in the project. Then they stitch the tiles together to form a minimap with the tiles. Lastly, they create render nodes with the 2D tile map which is later used to render the minimap to a scene graph. On Android, creating a 2D tile map in a scene view proved difficult. This is mainly because of the lack of documentation and the cumbersome method Sceneform renders 2D images using ViewRenderables and static image modules. On iOS, this part of the minimap implementation was developed in a short time without any significant problems. On Android, this part took a bit longer but ended up working as intended.

Minimap Controlling the 3D Camera

The next step in the process was to enable touch events on the minimap to control the 3D rendering camera, and enabling the 3D camera to move the minimap. The camera implementation on iOS was straight forward; we use events to pass state between the main camera and the minimap camera. These events are handled within the camera controller class itself thus keeping the two cameras separated. To accomplish the same effect on Android, we first had to rewrite the structure of the camera modules. This was mainly because we had a poorly implemented camera solution in our 3D view. We decided that this issue had to be fixed and restructured before starting the minimap camera translations. Where we before had one rendering manager class which handled the cameras, we now had a camera class which handles the main camera methods along with two subclasses: a main (3D) camera class and a minimap (2D) camera class.

We implemented the minimap in such a way that the minimap is a SceneView with a dedicated camera. This method allows us to move the minimap camera in relation to the 3D camera instead of moving the minimap in opposite relation to the main camera. This rewrite on Android took a bit longer than expected because of some mathematical and logical errors in the camera translation and rotation handling. This was mainly caused by complicated mathematical concepts such as quaternions.

After the structure was improved on Android, the next step was to implement methods for handling the translations between the 2D minimap and the 3D main view. Firstly, we tried to implement an algorithm to calculate how much the cameras had to move in relation to the other. The problem was that different floors and different buildings had different sized minimap tiles, which made the current solution useless as it only worked for some of the buildings. After a meeting with the product owner, they referred us to their current web solution and explained some more of their API endpoints which we could use which lead to the next attempted solution.

The next solution we tried to implement was using a transformation matrix supplied by StreamBIM's API for each floor. To do this, we had to implement a few mathematical algorithms to handle matrix calculations and other 3D mathematics. After fixing some logical and mathematical errors, the solution worked across every project as long as the transformation matrices were correct. One smaller problem we had, was to start the minimap camera at the correct initial position in relation to the 3D camera. We solved this by using the initial translated positions we use to position the 3D model in a method which converts 3D to 2D positions. After the initial positioning was correct, the last step was handling the different resolutions the different buildings' tilemaps had.

The implementation of the minimap took a lot longer than initially planned, which was caused by a few factors: firstly, the first notable rewrite to improve how we handled the cameras on Android, took a long time. Next, the solution which did not work across all buildings, and lastly, fixing some mathematical and logical bugs which took a long time because of the difficulty debugging 3D translations. In the end, we had a working solution and were able to use some of the knowledge regarding translations to help with the 3D to AR view translations.

3.4.4 AR Mapping

Addressing the Implementation Problem

The main problem with the implementation was that we were unsuccessful in getting it to work, or rather the translation of the model; we implemented all the solutions, but the problem persisted across all. The algorithm consistently found the correct rotation for the building as well as the same corner in both the room's geometry and the real world, but translating the model from point A to point B turned out to be a struggle.

The Debugging Process and Solution

A fundamental difficulty of debugging AR is that it has to be done by using the app, which is very tedious for our specific AR edge-case. The debugging cycle consisted of building a version of the mapping, trying it, then interpreting logs and attempting to fix the problems. The cycle was very time-consuming as much time was spent using the app and interpreting the vague logs containing geometry information.

After a prolonged period of debugging, we achieved a working implementation using the progressive method. The solution maps the model correctly to the floor, however, we found that the mapping of walls does not work due to StreamBIM's initial rotation of the building; StreamBIM rotates the model by 180 degrees on the z-axis due to an old implementation mistake. Nevertheless, StreamBIM does not flip the space geometry the same way as the model, which is why the mapping of the floor works and not the walls. A working implementation should be implemented within a short time-period now that we know the root of the problem.

3.4.5 Memory Management

Another issue we have encountered is a memory specific problem on Android. Android devices have a maximum heap size which is different on various devices. Since the team has elected not to implement streaming, because of time restriction, entire BIM models are loaded into memory for rendering on both platforms. On iOS, apps can allocate most of the memory, but this is sadly not possible on the Android platform. Since the Android memory is limited, some larger models are too big to be rendered on Android.

There are mainly two practical solutions to this problem when developing apps on Android that require a large amount of memory. The first one is to split the rendering into different processes. A process on Dalvik (Android's VM which executes applications) has a small heap limit, but if we could make several of them to handle the rendering together, we would solve the memory issue. The problem with this solution is how to pass renderable nodes through IPC (Interprocess communication) pipes. For this to be possible, we would have to serialize the Sceneform nodes which would be a huge task because of the complexity of these nodes.

The other popular solution for a fixed heap size is to use Android NDK (Native Development Kit), which gives applications access to unlimited memory, which was our initial plans for rendering. The problem with this solution is that Sceneform implementations do not support the use of NDK. Using NDK to store model data would solve the memory issue, but this would be hard to do alongside Sceneform.

The team has not prioritized fixing the memory issue on Android because we can load in decently sized models. The problem only arises when trying to load in every layer of a complex building. To showcase our AR implementation we do not need to resolve this issue, but the memory issue is a problem which should be fixed if the project is to be handed over to Rendra at a later date.

The last memory related problem on the Android platform is currently unsolved. The problem occurs during the loading of a model. It seems as if the memory continues to increase and is not properly de-allocated. We have spent some time trying to resolve this issue, and it is still unclear is it is a memory leak in our code, or if is an unavoidable effect caused by Sceneform. There has been reported a few memory leaks on Sceneform on the official Github page, but we have not yet excluded the possibility that there may be a minor error in our code causing the problem. This problem is unsolved because it has not hindered the loading of most models and after some evaluation, we decided that there were more pressing concerns to prioritize. If the problems do not reside within Sceneform, it is probable that implementing one of the solutions for the problem described above would also fix or identify this memory issue.

4 Testing and Quality Assurance

4.1 Documentation

We have followed several standards in regards to documenting the source code. There has also been written separate documentation to ease collaboration or the handover, to Rendra. A well-planned documentation strategy will help provide consistency and structure to the project. Linters has been used to enforce code standards which the team has followed for both the Android and the iOS parts of the project. The team initially planned to write tests, wherever feasible, for all code submitted to the developer branch of each respective repository. However, this requirement has not been prioritized equally to the rest of the project caused by the time restriction.

To provide proper documentation of code, we have been writing block comments for all the code in the project. This is to ensure that the code which is written is understandable and readable especially since there has been a lot of complicated math related code in regards to rendering, projection, translation and camera movement in this particular project. Block commenting has increased productivity in code reviews in the same regard. Furthermore, we have written different types of documentation, such as installation guides and other various documents to aid Rendra in taking over the project after project end. Hopefully, these steps will help to ease the transition where the code base and our obtained knowledge is handed over to Rendra AS after the thesis is completed. Proper documentation will also hopefully help Rendra avoid problems which we have experienced throughout the project, and we hope that they may implement one of several suggested solutions to problems we have not had the time to solve towards the end of the project.

4.2 Tools

Some external libraries and tools have been essential to achieving proper documentation throughout this project. As mentioned above, linting, testing, git hooks, and more have been used to improve documentation in our project. These methods are made a lot simpler using good existing libraries and tools. In this chapter, we will list and describe the tools we have used in this process.

4.2.1 Linting

All code has been inputted through linters for both platforms throughout development. The libraries we have used for the Android and iOS platform are called KTlint and SwiftLint, respectively. Both linting libraries enforce a good code format and style which can also be manually configured to ensure that it fits our specific code. In the beginning, we expected the use of linters to help with code reusability considering customized rules we specified in early development. Although this has helped somewhat towards that goal, other problems described in 3.4 has lead to reduced code reusability, which is regrettable.

4.2.2 Version Control

Throughout development, we have used Git actively as a tool for collaborative work, version control, and issue tracking. While using Git, the team has been using a workflow known as Gitflow in combination with feature branches for consistency and structure. Feature branches are a concept within Git branch modeling where branches are named based on features. Gitflow is a tool for branch modeling in Git which eases parallel development and collaboration by enforcing feature branches in addition to having a development or staging branch which the team merges their respective feature branches with. Only at the end of a sprint has the development branch been merged into the master branch. This gives us a good overview of every sprint by reviewing the git log in the master branch and allows us to easily roll back to earlier versions of the master branch to review or test different aspects with earlier implementations.

4.2.3 Pre-commit Hooks

When working together as a team, it's important to have the same coding style and make sure code which is pushed to the repository is not broken. As mentioned in *4.2.1*, the team decided on a code-style early on in the project and we set up linting tools to handle these checks for us. To automate the linting process we created a few hooks for git. The main function of a git hook is to hook in to a part of the process and to run a set of tasks. Naturally for linting and testing the hook in question is a pre-commit hook. Git hooks are not pushed to the repository so they have to be specified by each developer. We kept the hooks themselves on the repository as a flat file and required the developer to link them to their local repository.

The hook itself presides under the file *.git/hooks/pre-commit* and can be any executable script. The flexibility of git hooks allowed us to automate a lot of work that we would otherwise do manually. An example of the pre-commit hook used for the iOS platform is attached (See Appendix D).

4.3 Quality Assurance

4.3.1 Functional and Unit Testing

As mentioned throughout this thesis, this project is centered around developing a complete stand-alone application. The primary purpose of this project is to gather information about various subjects such as native rendering and mainly AR in BIM. Although we wanted to develop a robust AR implementation, the highest priority was to get the functionality we needed to implement AR. Therefore, unit testing has not been highly prioritized after development started. However, critical parts of the project have been unit tested, particularly in the parts which handle AR geometry, translation, and other mathematical operations.

Unit tests like these were essential to us because of the problems we had regarding translations in AR. The unit tests ensured us where the errors were not so that we could rule out some mathematical and some logical faults. These tests also help validate the quality of our code and are useful when the product owner takes over the project. It would be beneficial to have maximum test coverage, and this may be implemented after the project ends to ensure good stability of the module. As far as functional testing is concerned, there has not been any. We deprioritized functional testing in our scope; however, functional testing may also be implemented after the completion of the project.

4.3.1 User Test and Surveys

We conducted a series of user tests and surveys to validate the usefulness of AR for BIM technologies. This validation is vital for the project, as we have little knowledge about how construction workers conduct everyday tasks. Getting an idea of how the end user perceives the AR implementation is very important for this project so that we can validate our current implementation, get suggested changes, and validate the core concept of using AR in various job positions throughout the building industry.

There were mainly two types of tests we wanted to conduct. Firstly, we wanted to take our AR implementation to a construction site and ask a construction worker to use our product and explain what they think about the idea, and give us constructive criticism about the user experience. The second type of test we wanted to conduct was general surveys for different people working in different jobs in the building industry. People in leadership positions could, for example, explain if there are any financial benefits to be gained using AR, or if a company would use such a product in new projects or for the rehabilitation of existing buildings. Sadly, we were not able to organize a meeting at a construction size because of time restrictions and that such meetings require adequate time for scheduling. We were, however, able to survey a variance of people in the construction industry in the form of interviews and surveys.

5 Results

5.1 The Resulting AR Application

This project has resulted in a cross-platform mobile application that is capable of visualizing StreamBIM's BIM models correctly in the real world. The application enables users to log in with StreamBIM credentials, browse projects and buildings, and observe these projects in both a virtual environment and in AR.

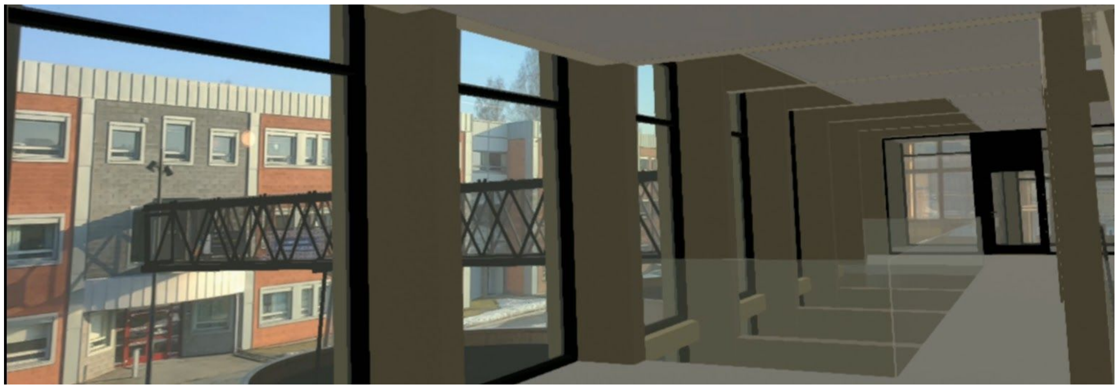


Figure 16: The NTNU S-Building's Architectural Layer in AR in the Application

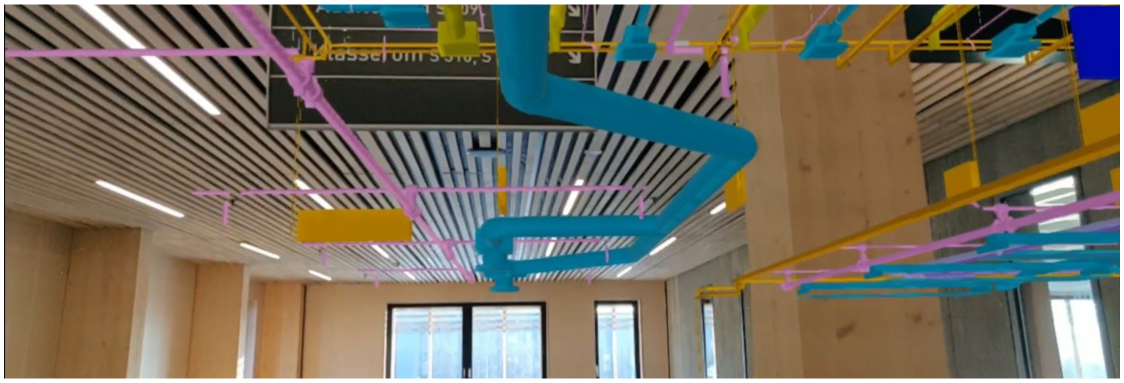


Figure 17: The NTNU S-Building's HVAC and Electrical Layer in AR in the Application

Moreover, the application implements a dynamic and functional minimap for localization both in virtual and in AR mode that also allows for more straightforward navigation in virtual mode. Also, users can select objects such as walls and pipes in both virtual and AR mode, and retrieve information about their name, type, and description. The finished prototype satisfied our initial scope and expectations, and it performs rather well.

5.2 Target Audience Response

The responses are primarily based on surveys from people in various positions that use BIM regularly, e.g., educators, construction leaders, and engineers. Most of the participants have earlier experience with StreamBIM, but there was some shortage of experience with AR; most of the participants had, however, used some form of AR application, such as the IKEA furniture app or AR measuring tools. Moreover, several of the participants have used Dalux's AR BIM implementation.

According to the surveys, the ability to observe BIM content in the real world using AR is a desired feature that may improve several aspects of BIM. First and foremost, the target audience agrees that AR will simplify the usage of BIM, provide a better perception of buildings, and contribute to better educational aids. Furthermore, aspects of BIM such as documentation, service and maintenance, and renovation is especially expected by the participants to be more helpful in AR. In regards to object-selection, all of the results stated that AR would improve object-selection. However, an unexpected factor emerged from the survey. Some participants stated that workers, especially foreign workers outside of Norway, could perceive AR as a gimmick. They further stressed that it is essential to showcase what use-cases are improved with AR to ensure that it can be a successful feature in future BIM technologies.

Although an AR feature can be recognized as a gimmick, most participants asserted that they would use the implementation actively if the calibration process is tolerable; participants agreed that around 4 or 5 seconds, depending on the room, is a quick enough calibration process. However, the participants working on-site stated that poor lighting conditions are common in the industry; many countries experience dark winters, and artificial light is commonly added later in the building process. This entails that the calibration could be worse in such conditions, yet further development of mobile and AR technologies will only improve in the future, and hopefully reduce the consequences of poor lighting and similar conditions.

5.3 Value for Product Owner

The thesis has primarily provided the product owner with desired information and research about the process, solutions, and challenges of developing an AR implementation, as well as a working demo that Rendra was hoping for. Moreover, the response from testers has ascertained that object-selection is a better experience in AR, which indicates that more features may prove beneficial in AR as well.

The most valuable element of this thesis is the research and information that was gathered when developing the AR application. The information will allow Rendra to more easily make decisions going forward in regards to a possible migration to a standalone StreamBIM application with native rendering and AR. Rendra has expressed a positive reaction to the results of the thesis, especially in the positive feedback from the target audience and the possibility to keep much of succeeding code uniform under a shared codebase with React-Native.

Furthermore, even though the Android application is experiencing critical memory problems, Rendra is optimistic in regards to the native rendering on both platform; they were impressed that larger models were able to be rendered with the Android heap limitation, especially without existing StreamBIM optimizations such as occlusion culling, detail management, and filtering.

5.4 Differences Between the Platforms

We have conducted a series of benchmarks to test and differentiate technical, functional, and performance-related aspects of the application on the different platforms. We have benchmarked the application in neutral environments, and we have attempted to ensure that the testing conditions are as equal as they can be for both platforms. This is done to ensure unbiased or inaccurate results. When benchmarking performance, we used the same internet and the same building with all layers loaded. The building with all layers is barely under the heap restriction on the Android device used for testing, and it contains 270 nodes.

Even though we ensured an equal testing ground, the specifications of the devices used to conduct the benchmarks are different. We used an iPhone XS to benchmark the iOS application, and a OnePlus 6T to benchmark the android application. Both devices are flagships for their respective platform, but the iPhone is objectively more powerful; the iPhone's A12 Bionic processor is, according to the Antutu ranking board, around 20% more powerful than the OnePlus' Snapdragon 845 [34].

Model Load Time

As expected, the iOS application is able to load the model significantly faster than Android; the model is completely loaded on iOS after 35 seconds, while the model needs 60 seconds to be fully loaded on Android.

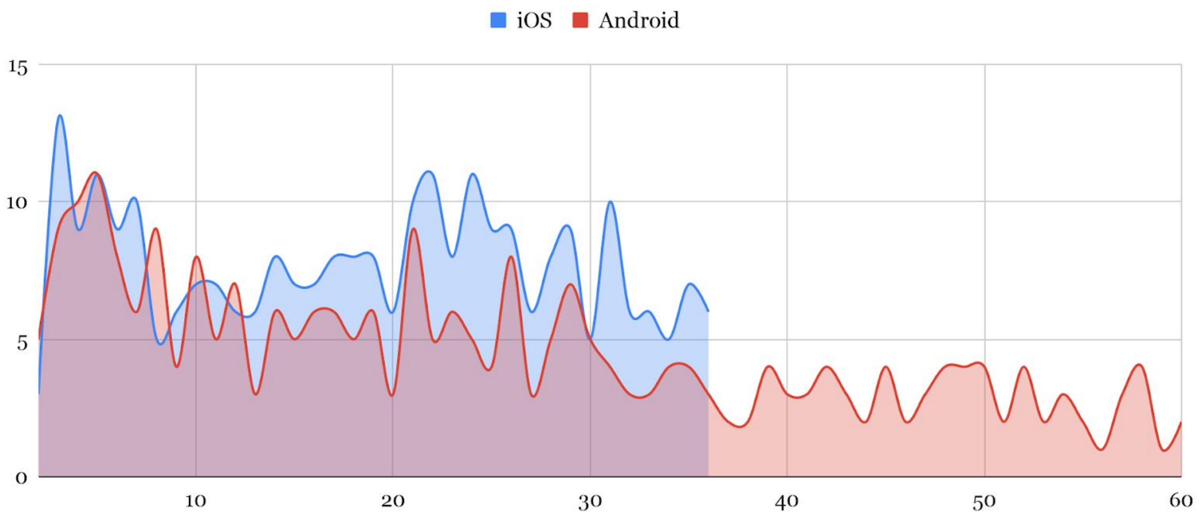


Figure 18: Histogram of Processed Nodes per Second When Loading the Model

Furthermore, iOS has a consistent rate, while the Android rate slowly declines overall throughout the loading process. The reason for Androids declining rate and slow loading time is that nodes on Android are processed succeedingly for each layer. Most of the smaller layers, such as architecture and electronics, was fully loaded on Android between 10 and 30 seconds. The remaining nodes processed between 30 and 60 seconds are nodes from layers with significantly more nodes, such as HVAC; these larger layers need more time to process, as many nodes are queued to be processed. In contrast, the iOS implementation can process all nodes from all layers in parallel, which allows for much higher efficiency and faster load time.

Memory Usage

The memory consumption for both platforms when loading the model is vastly different. The iOS application allocates and keeps resources when loading the model, while the Android application frequently reclaims allocated resources through garbage collection. The increasing memory usage on Android suggests the possibility of memory leak, as discussed in *3.4.5*. The memory usage does, however, stay consistent after the application has completely loaded the model.

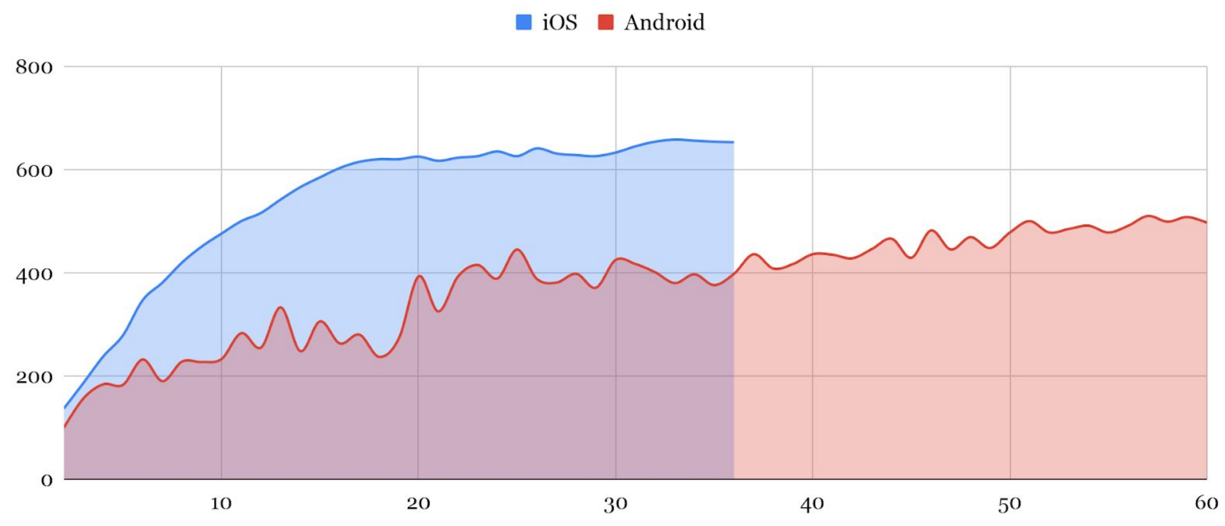


Figure 19: Histogram of Memory Usage When Loading the Model

FPS

A very noticeable difference between the platforms when loading the model is their framerate. The framerate on the iOS application stays consistent and high throughout the process, while the framerate on Android is stuttering prominently. Most of the framerate on Android is acceptable, but it is sometimes very noticeable and unenjoyable when the framerate drops to below 25 fps. Figure 20 illustrates these points well, but the figure describes the average number of frames per seconds, thus not showing the fragments when the framerate dropped below 20. Furthermore, the framerate is consistent with the number of nodes being processed, as described by figure 18; the frame rate is at its lowest when the highest amount of nodes are processed simultaneously, which is when multiple layers are processed, i.e., 0 to 30 seconds.

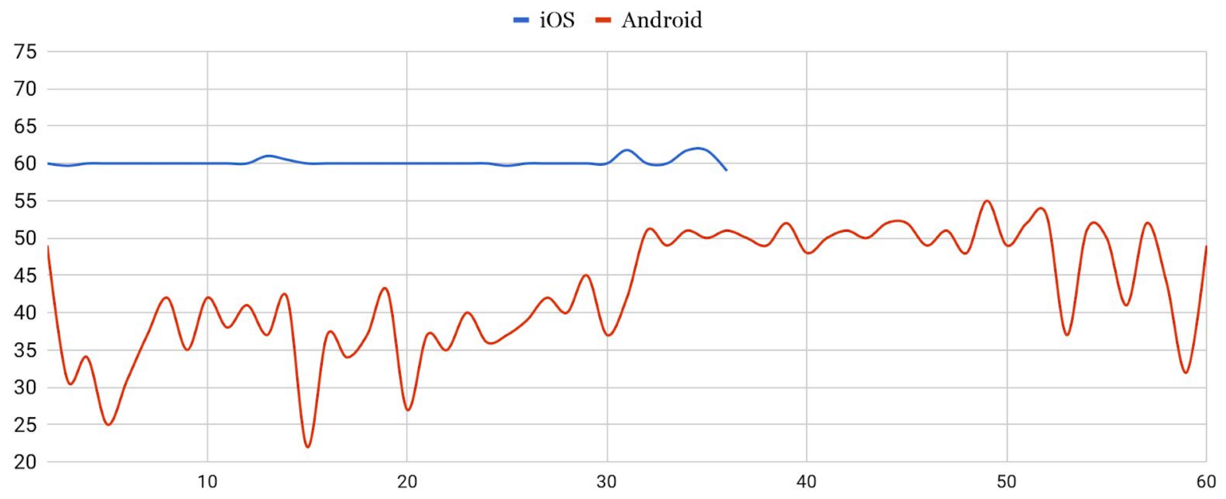


Figure 20: Histogram of FPS when Loading the Model

5.5 Deviations

Functional

For the most part, the application's functionality is in spec with minimal deviations, as well as exceeding the product owner's expectations; the project resulted in a fully working AR application that matches the project's description and scope. However, the most significant deviation in the app's functionality is its inability to load large models on Android, due to the unforeseen memory restrictions present on the platform.

Technical

It was expected that placing models in AR would require a more advanced integration process than what was needed. The project plan took into account that the process of displaying models and calculating camera movement had to be done manually; however, the AR frameworks handled this automatically.

At the beginning of the project, a custom rendering engine seemed more intriguing than using a 3D framework. Despite our desire and the benefits of building a rendering engine from scratch, the focus changed in favor of 3D frameworks to preserve time.

It was initially planned to create two separate native applications that used an intermediate layer to simplify and unify rendering calls. However, the apps were merged with React-Native as it was achievable and allowed for a cross-platform application through a shared codebase.

Work Methodology

Even though the teams responsible for their respective platform should collaborate and work on the other team's codebase occasionally, the teams mostly stuck to their addressed codebases throughout the project; code reviews were, however, conducted across the teams.

The developers on the Android team were assigned unique roles, primarily lead developer and UI/UX designer respectively; however, the focus on these roles decreased during the project, as there was close collaboration between the team members and decisions were able to be achieved collectively within the team.

Throughout the project, the focus on planning and routines became slightly relaxed, and communication between the Android and iOS team was also declining somewhat due to illness. However, meetings at Rendra in Oslo were carried out consistently and continuously throughout the project.

Quality Assurance

While the project plan specified that the whole codebase should be thoroughly tested with both unit tests and functional tests, we decided to only test crucial elements of the codebase such as advanced math functions and utilities; only testing crucial elements eliminated the risk of difficult debugging, but allowed for more time developing the application. Furthermore, we decided that a thoroughly tested codebase is not vital for this project, as this project mostly is for research purposes.

As described in the project plan, all code written must have documentation in the form of block comments. This is the case with the final codebase, but the process of documenting code has been inconsistent during the project; some sprint tasks has solely centered around documenting undocumented code.

5.6 Uniqueness

At the end of development, we wondered how unique our finishing application turned out to be. There are not many other available solutions today, so in that regard, our core concept is somewhat special. However, other competing products have been listed in *2.1* and comparing our results to these shows that in functionality, they are quite similar. This is because there is not much flexibility within the scope of implementing AR for BIM models, yet there are differences in usability, stability, user experience, and maturity. The product we are comparing ourselves to, are production ready, fully developed application. So a direct comparison may not be completely fair. However, keeping in mind our time restrictions, we argue that our application is able to compete with these products. The only feedback we received from a user of the TwinBIM solution by Dalux, was that it was not very user-friendly and struggled with aligning larger models. The method we use for AR mapping is not affected much by the size of the models, yet we do require that the user detects at least the floor and one or more walls. This part of our application has not been tested as thoroughly as we would like, but we feel confident that our resulting application is quite unique in regards to the user-friendly UI, object selection, and in general stability for the models we have used for testing.

5.7 Comparing the Results and Expectations

In favor of both the team and the product owner, a working AR implementation was able to be demoed towards the end of the project period; this sat us in a position to start the thesis writing early, fix bugs, and conduct testing with on-site workers. Even though the organization of user tests failed, due to lack of time and the substantial preparation required, the results from surveys were positive and suggested that AR is expected to be a useful tool for BIM solutions.

Furthermore, Rendra specifically wanted the application to be as similar on the two platforms as possible, and we have achieved that by using React-Native. In spite of the rendering and AR aspects of the applications being platform-specific, the shared codebase allows much future code to be shared between the platforms. Also, there are ways for Rendra to do these platform-specific modules more similar, which we further discuss in *chapter 6*.

Moreover, we believe that this project has provided sufficient information to help Rendra make decisions going forward; Rendra expected information about platform-specific challenges, solutions, and research, which this project has presented. The project has also, through surveys, ascertained that AR is a useful feature for Rendra to consider and that AR is implementable using StreamBIM's existing data.

6 Future Work

Towards the end of project development, it is essential to discuss the project's future. Throughout this thesis, we have described the task and goals of this project, and we have established that Rendra will take over the project and will use the knowledge we have obtained to evaluate if they will implement AR in their application. In this section of the thesis, we will discuss Rendra's handover of the project, future work for Rendra, and future work for the group. We will also discuss some aspects of the project which the team would have liked to implement given more time before the deadline of this thesis.

After the project is finished, Rendra will use the research and knowledge we have obtained throughout development and the code we have written to evaluate if they are going to implement the main features of our application such as native rendering and AR, and then perhaps implement a similar solution. For Rendra to make the best possible decision, they have asked us to provide a comprehensive list of what has been challenging and what type of work they can expect when developing a similar implementation. Furthermore, Rendra has also expressed that they want to know what kind of new libraries which may aid them through development or to avoid mistakes or obstacles we have encountered.

6.1 Resolving Obstacles

The mistakes or obstacles we have faced throughout development has been thoroughly described in 3.4, so to recapitulate the points most relevant for future work:

Sceneform Maturity

Sceneform is a relatively recent project, especially in regards to as a standalone rendering framework (disjoint from ARCore). This has lead to some difficulties throughout development in regards to both the rendering and the 3D navigation. If it is preferable to continue working with the 3D frameworks in the future, it will be necessary to evaluate the use of Sceneform in a professional context and whether or not Sceneform is mature and stable enough. In our experience, Sceneform works well enough for our tasks, but we also predict problems if more complicated features are required, such as rendering which requires shader programming. If Sceneform proves too challenging to work with, it would be more relevant to look into a custom rendering engine.

AR Mapping

AR mapping has been the most significant obstacle for us to overcome. For future work, it may be pertinent to reevaluate this process. Although we feel confident that we have implemented an adequate solution to this problem, it may be relevant to rethink the issue and see if there is a better solution. For this issue, there are no tools or libraries which can make this process easier; it mostly revolves around using mathematical methods to align the renderable Scenform nodes with data from ARCore.

Android Memory Management

Memory management on the Android platform will be somewhat of a challenge in the future. For this issue, we have described several possible solutions which may be implemented in the future. However, in the future, it would be more pertinent to implement streaming properly on both platforms. This will reduce the amount of memory required on mobile devices, and it may not be necessary to change the current memory management.

6.2 Future Rendering Decisions

One of the more extensive discussions about future work on this project would include whether or not to continue using the 3D frameworks or step back and develop custom rendering engines. The 3D frameworks have been beneficial in this thesis to get 3D rendering working as soon as possible. They are also quite performant and requires less knowledge about 3D graphics. In the future, we believe that it could be beneficial to take the time to implement a rendering engine customized to the requirements of the application. One of the reasons we suggest this is because of the current maturity of Sceneform and uncertainty if Sceneform on Android will be suitable for more complicated features such as clipping, occlusion culling and general filtering which Rendra uses in their current web-based solution.

Rendra has also expressed some difficulties and higher expenses regarding development across two platforms. It would be much more advantageous to have a cross-platform solution for rendering. This is another reason we think it may be beneficial to develop a custom rendering engine. For a custom rendering engine to work cross-platform, it would have to be written in C. On iOS, it would be possible to include C headers directly into the project. However, Sceneform would have to be removed from the Android project as it does not support the use of an NDK which is necessary to have native C code included in a project on Android. This would then also entail rewriting the minimap module as this also uses Sceneform's rendering capabilities.

6.3 Desired Features

Throughout development, there have been some features which the team has wanted to implement to improve the user experience or solely for optimization reasons.

Occlusion Culling

One of these features which we would like to implement is occlusion culling, which is a rendering method that disables the rendering of an object which the user cannot see. This would both optimize the performance of the rendering and improve the user experience in such a way that when looking through walls in AR, one could not see objects which are in a different place in the building.

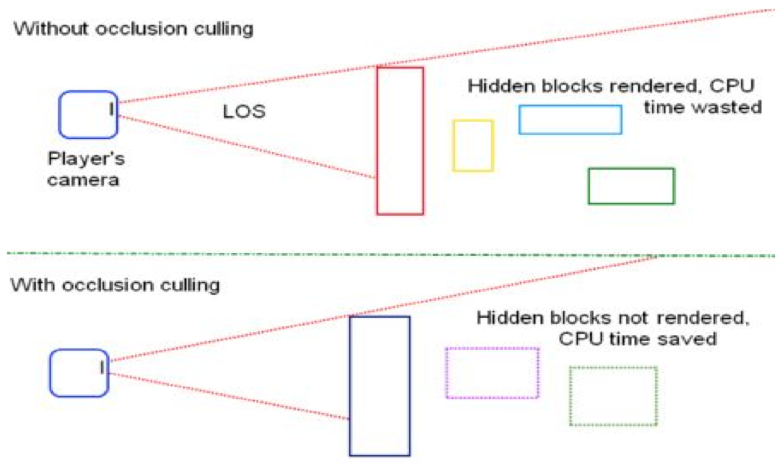

Without occlusion culling
LOS
Player's camera
Hidden blocks rendered, CPU time wasted
With occlusion culling
Hidden blocks not rendered, CPU time saved

Figure 21: The effects of occlusion culling

https://feedback.bistudio.com/T66269

Clipping Planes

Such an effect could also be achieved with clipping planes. Clipping planes are a feature which StreamBIM uses to allow users to clip through objects and walls. They work by removing data which lies behind the clip plane and only renders objects in front of it. We could achieve an effect similar to occlusion culling by automatically creating a box of clipping planes around rooms so that when a user uses AR to look at pipes through the architectural layer, they would not be able to see the pipes which lay beyond the clip planes. If in figure 21, the darker blue rectangle would be moved behind a wall in the model, the pipes and objects inside the wall would be rendered, while the objects in different rooms would not be. This could help a user get a better visualization on sub-layers without being distracted by objects in different positions all around the model. It would also help users with specific jobs such as electricians and jobs regarding ventilation, which is often laid behind walls in buildings.

Shaders

To implement one of these solutions would require the use of custom shader programming. On iOS, we already know that this would not be a problem, but we have not tested this on Android with Sceneform. Sceneform is supposed to support custom fragment and vertex shaders through a material file which is supposed to work quite well. However, some research may be required to see if this feature works as intended.

Cloud Anchors

Cloud anchors could be implemented in the future, to improve the AR experience for users. This would require modifications to Rendra's backend services along with changes in our AR application. If cloud anchors had been implemented in the future, it would open the door to many potential features. Users of the application could share data, create shared points, or visualize issues or problems on the construction site. If an AR module is adequately implemented in StreamBIM in the future, we recommend implementing cloud anchors as a natural next step to improve the user experience. Figure 22 shows an example of two users seeing the same object in an AR environment. This is a reasonably simple example which can easily be applied for several features in our AR application.

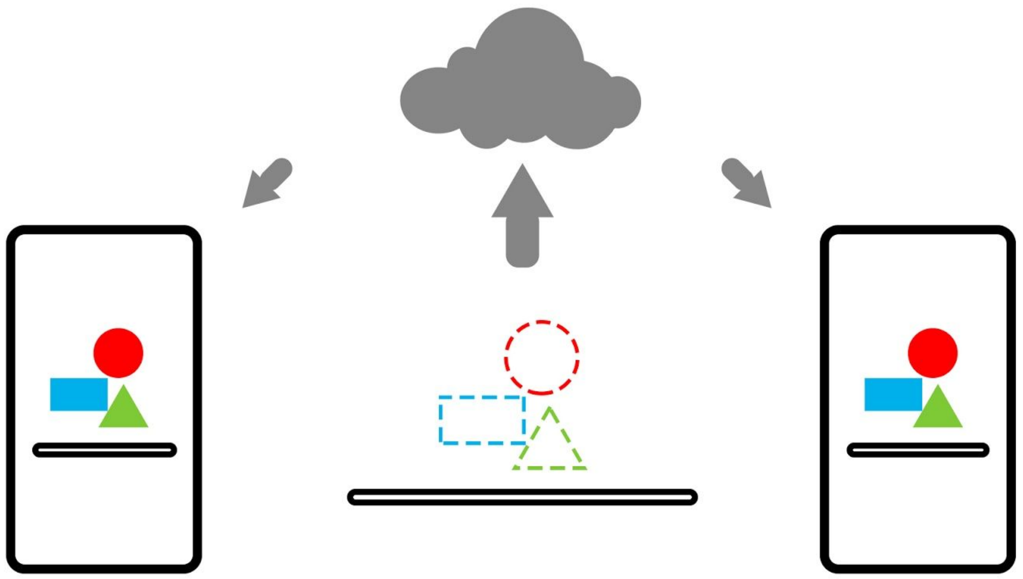


Figure 22: Shared AR experience using ARCore Cloud Anchors

AR Measurements

Another feature which may be implemented in the future is measurements. In StreamBIM, a user can measure lengths and dimensions of the models. This is a handy feature for construction workers. This could be implemented in the current version of our AR application. AR measurement tools already exist in abundance, and to combine our current AR functionalities with a measurement tool would be an excellent way for construction workers to take exact measurements without any other tools than their phones.

AR Issue Tracking

In the current version of StreamBIM, a user can create issues for every member in the project and mark it with a specific place in the model along with pictures and text describing the issue. This would also be a feature which could be improved by AR. Combined with object selection, we could implement a solution where a user can look at a place in the model and mark it as an issue in our AR application. AR could be used to capture pictures of the problem in the real world and inside the model. Also, if cloud anchors would be implemented, other users could see the same issue in the same location in AR and be able to comment on them.

7 Discussion

7.1 Learning Outcome

We have worked on this project for a long time, and as we initially chose a large task, we have been developing actively since the project start. Throughout the project, we have learned a lot, touching on several different subjects regarding the organization and developing of this massive project. Throughout the project, we have found that we can obtain much knowledge and become proficient in multiple areas in a short time, while also being able to apply it productively. To properly discuss our full learning outcome, it is relevant to describe our fundamental knowledge about such a project, or what we knew going into the project.

Before the project start, we had done individual research about a range of topics such as mobile development in Kotlin and Swift, BIM data and data formats, Rendra's existing technologies, and some 3D graphics. Although every team member had some previous experience working with iOS and Android, there is still a lot to learn throughout the project. Two of the team members had also worked closely with Rendra earlier and had a good idea of their product and their customers. All team members also had a basic understanding of AR and had a decent idea of how extensive the assignment would be.

Although the team members had some basic knowledge and experience about StreamBIM, AR, and mobile development, in hindsight, there was a lot we did not know beforehand. Firstly, none of the team members had worked with Kotlin earlier. However, the team has experience in Java, but the team saw the syntactical advantages that come with using Kotlin and saw the usefulness of learning a new language for Android development. Furthermore, even though we had some experience with Android development, we had no experience dealing with memory-extensive applications. Two of the team members also had no experience with Swift as a language or iOS mobile development. Therefore we arranged a weekend where the whole team developed rendering related functionality in Swift and the rendering library Metal.

In regards to actual rendering, AR, and translations, we had no previous experience at all. Every team member has focused application development and had never worked with 3D graphics, rendering, or the mathematics that is required within these subjects as many of these appear in game development and not application development. An excellent example of this is the use of quaternions for rotation. Other mathematical subjects like these were essential for us to learn in order to achieve smooth camera movement, positioning, and other calculations. Several other subtopics within 3D graphics and rendering were also completely new to us, such as translations, perspectives, projections, and AR mapping. We did know what AR anchoring was but had no experience mapping objects to the real world in the way we did in this project.

From the beginning, we knew there would be a lot of research and learning required to complete such a task, and although it seemed like we were in over our heads in this assignment, we managed to obtain the knowledge and experience necessary to finish the task we chose. Close to project end, we are all left with a lot of new experiences and knowledge which is great for young developers to have. We have all become proficient in rendering technologies, 3D mathematics, mobile development, and project management. From another perspective, we have learned a lot about architecture, development patterns, and best practices on two different mobile platforms. We have organized meetings and kept summaries as well as worked closely with a professional company which has taught us a lot.

We have also obtained knowledge and experience in regards to organizing a small team working with a large project over a lengthy period. We have made mistakes and corrected them as we went along and were able to plan and execute a project plan even though there have been several deviations from that original plan. A rather unexpected area of which we have learned a lot about has been StreamBIM’s users in the building industry. To ensure good user experience and to make sure our product would be useful for the customers, we had to have a good idea of how the users would utilize the product. We have also learned something about architectural layers and BIM technologies and how such technologies are being used today as well as the digitalization of the building industry in general.

We, as a team, are confident in that the knowledge and experiences we have obtained throughout this thesis will help us become better developers and researchers in the future. As mentioned earlier, the knowledge we learned has already helped us develop and improve the application, and the organizational skills we have learned will help us work better in teams and make it easier to plan more substantial projects in the future. In this manner, this project has been an important step towards us becoming more proficient and experienced developers, and we will take the knowledge with us in later projects and work.

7.2 Reflection

There are various aspects of the project worth reflecting on that centers around the achieved results and our process of conducting this project. Nevertheless, we are generally pleased with the results and the work process.

7.2.1 The Results

As a team, we are delighted with the results emerging from this project. First and foremost, our AR application is within its extensive scope, exceeds expectations, and has yielded positive feedback from people interested in the thesis, testers, and the product owner. We are happy with how our application turned out; it works well on both iOS and Android, has excellent performance, and we generally think that it a cool and useful addition to BIM. As developers, we liked the straightforward connection between the native rendering libraries and the AR modules for each platform, as they allowed us to integrate AR in the app easily. Moreover, and especially on Android, we liked that the strict code structure enforced by Kotlin lead to a better code structure and architecture. There are, however, aspects of the application that we dislike. First off, the memory restriction on the Android platform has restricted us from being able to render the largest BIM models on Android. Rendra has expressed that they are not too worried about the matter, but we dislike this and would love to find a solution to this problem. The memory problem has further forced us to throttle model loading on Android to conserve memory, which we find unfortunate and upsetting. Also, we dislike the structure and architecture of the Android application that resulted from the best practices, recommended architecture, and suggested libraries such as Retrofit. We found this architecture to cause a significant amount of boilerplate, and it was also very different from the architectures we had previously built and enjoyed during our studies.

In regards to the feedback from testers and Rendra, we are pleasantly surprised that the majority has found our implementation useful. It seems like a lot of our peers, teachers, and people working with BIM technologies are very excited about the idea of AR in BIM, and several of them were happy to answer surveys regarding the subject. We are also pleased with the results of the surveys conducted. We had a good group of subjects with different relationships with AR and BIM, and who also had different opinions regarding an AR implementation in BIM. We received great feedback, and the subjects were almost all positive to our core concepts and were surprisingly accepting of different problems with AR technologies today, such as poor calibration under poor lighting. The positive feedback from the survey helped us get an idea of how the typical users of our application would react to it, and it also helped validate our core concepts and our implementation in general.

7.2.2 The Work Process

We have, for the most part, experienced the work process as positive; it has yielded educational learning outcomes and great insight into the development process with a remote product owner on a large-scale project. We are grateful for the experience that we have obtained in this project, and the close relations made with the developers at Rendra; we have found this collaboration fun and enlightening. Furthermore, we found the task extremely exciting and had a positive attitude going into this project, even when we knew that the task would be hard to complete. We did, however, finish a fully functional proof-of-concept prototype, and we are very proud of what we have achieved during this semester. One of our most positive experiences during the development was when we finished a working prototype and tested it on the NTNU S-Building; the model mapped perfectly, and we enjoyed seeing the application come to life. We are glad that Rendra, our supervisor, various teachers, and others has shown genuine interest in our thesis. We are confident in our choices during the process, and we argue that the headstart we received at the start of the project is one of the many reasons contributing to the completion of the application.

There has, however, emerged a few unpleasant aspects during the process leading to bad experiences and low morale. First off, this project has been incredibly challenging due to the advanced math involved in rendering concepts and geometry in general. Furthermore, our lack of knowledge about these advanced topics made the process more challenging. These reasons also made the app hard for us to debug, which especially was the case with the AR mapping. Moreover, we argue that much of the annoyance regarding the work process were Android related. We found Android Studio, the IDE used for developing apps on Android, tedious and frustrating to use; the UI design process in Android Studio has been sluggish at best, and we were forced to restart the IDE multiple times due to false negatives such as false compiler errors.

One of the negatives regarding Android development is that Android Studio is forced in such a high degree as the recommended IDE. Android Studio has some nice features which other IDEs has struggled to implement, however, it performs poorly and has several annoying aspects to it, especially for developers who are used to lightweight IDEs such as Visual Studio Code and Atom. Shortcuts in Android Studio are different, which will take some work rebinding, it is full of useless features and it is an incredibly large and slow program. During development, a group member spent some time setting up an independent development environment using the command line scripts and Visual Studio Code. However, it seems that there will always be some issues with Android development outside of Android studio which causes more problems than it solves. After exploring other options, we figured it was little point not using Android Studio, but in some aspects, it has a tendency to slow down development, especially on older and slower computers.

Moreover, we are disappointed in our inconsistency with Scrum, and think we could have avoided this inconsistency even when the team members were ill. Also, even though we had consistent meetings with Rendra in Oslo, we should have reached out for more help and programming sessions with them; Rendra also said, at the end of the project, that they would have appreciated more joint work sessions.

7.2.3 In Retrospect

Although it has been an overall great experience working with this project, there has been several obstacles and difficulties which we had to solve or live with throughout development. Therefore, during development, we made some thoughts about how we would have done it everything differently if we had started over having the knowledge and experience we have now. We believe that we could have avoided some of the obstacles given better planning in some areas or more experience in others. There have been some mistakes that have been caused by us directly, but also some obstacles which were introduced by libraries and tools which we have used. Therefore, we could have avoided used such tools if we knew about their shortcomings beforehand.

A thoroughly discussed subject throughout this thesis has been rendering. We have earlier discussed our choices and the advantages and disadvantages in regards to using existing 3D frameworks or developing a rendering engine from scratch customized for our needs. Keeping in mind the problems we had working with Sceneform on Android, and our newly obtained knowledge about rendering, we would have made different choices in regards to rendering methods, if we had started over today. For example, it could have been better to choose a custom rendering engine given the experience we have now.

An interview with the CTO (See Appendix E) of Rendra told us that Rendra would have difficulties dropping their web-based solution for a non-cross-platform solution because of the extra expenses and organization this would require. They would either continue development on one platform or try to find a cross-platform method of rendering. Given that Rendra has expressed a desire to see a rendering engine combined with our difficulties regarding the 3D frameworks, it would seem it would have been more advantageous to develop a rendering engine in C, Metal and OpenGL ES. A rendering engine could have been combined with an intermediate rendering layer for both platforms and an NDK on Android to achieve cross-platform rendering.

Keeping this in mind, we believe that we made the correct choice in not doing this with the limited knowledge and experience we had at the project start in regards to rendering. However, if we were to start over, we would take the time to create such a solution. Another benefit from this could be that Rendra could keep their web-solution and port a C++ rendering engine to Web Assembly to get a substantial increase in rendering performance.

Next, we would most likely have decided to only focus on one platform, which would ensure a better, less experimental application, and would allow us more time to explore and document the more challenging parts of the process. In the beginning, the scope of this assignment felt a bit large, but we did manage to exceed expectations and develop a working solution in time. However, it could be better to have a more limited scope, in order to research some subjects more in-depth. It would also allow us to explore some subjects we left out of our scope, such as streaming and give us more time to quality ensure the code with functional testing and more unit testing.

Regardless of the hypothetical changes that we would have done differently, we believe that Rendra will be able to avoid making the same mistakes going forward. Furthermore, if we decide to continue working with this project, we will probably implement some of these changes to the existing application to make a future version better and more stable.

7.3 Main Conclusion

To conclude our findings, the finished application we have developed throughout this assignment has turned out successfully, even though the scope was initially considered quite large. However, after countless work-hours and much hard work, we managed to exceed both the product owner's and our own expectations and ended up with a fully working prototype. All team members has worked extremely hard throughout the project with over 1600 work hours combined across the three members. This thesis has touched on exciting concepts which have provoked interest by teachers, CEOs of BIM technology companies, and our peers. We had our problems during development and project management, but at the end of the project, we are incredibly pleased with the result of this thesis. Furthermore, throughout development, it became clear that both AR and some of the available rendering tools we used, were not particularly suited for our comprehensive requirements. This lead to us having to experiment and come up with clever solutions without much external help or available resources. We started out with limited experience and knowledge regarding the BIM industry, AR and 3D rendering, yet we managed to obtain all the knowledge required to finish development in good time. Because of these reasons, we have all learned a great deal in several areas of programming, project organization, and even the building industry, and at some points felt like we explored little researched areas of technologies. We think that the application we have developed is worth continued development with many exciting potential features, optimizations and opportunities. Furthermore, we are thrilled with the fact that we have made an MVP and gathered research which has actual value for Rendra and that we were able to contribute to the further development of their product StreamBIM.

8 Sources

1. COMM/JRC/KFG/. IRI - The 2018 EU Industrial R&D I [Internet]. [cited 2019 May 2]. Available from: http://iri.jrc.ec.europa.eu/scoreboard18.html

2. Re-Modelling Construction: How the Industry is Digitizing [Internet]. Geospatial World. 2017 [cited 2019 Apr 25]. Available from: https://www.geospatialworld.net/article/how-re-modelling-construction-digitizing-industry/

3. Building information modeling - Wikipedia [Internet]. [cited 2019 May 1]. Available from: https://en.wikipedia.org/wiki/Building_information_modeling

4. Takahashi D. AR companies have grown 50% since the end of 2017 [Internet]. VentureBeat. 2018 [cited 2019 Jan 23]. Available from: https://venturebeat.com/2018/07/12/ar-companies-have-grown-50-since-the-end-of-2017/

5. Marr B. 9 Powerful Real-World Applications Of Augmented Reality (AR) Today [Internet]. Forbes. Forbes; 2018 [cited 2019 Apr 3]. Available from: https://www.forbes.com/sites/bernardmarr/2018/07/30/9-powerful-real-world-applications-of-augmented-reality-ar-today/

6. BIM and Augmented Reality – An enhanced view for enhanced outcomes [Internet]. Geospatial World. 2018 [cited 2019 May 2]. Available from: https://www.geospatialworld.net/blogs/bim-and-augmented-reality-an-enhanced-view-for-enhanced-outcomes/

7. TwinBIM - Augmented Reality - Dalux [Internet]. Dalux. [cited 2019 Apr 24]. Available from: https://www.dalux.com/dalux-field/twinbim/

8. Home | GAMMA AR [Internet]. GAMMA AR. [cited 2019 Apr 25]. Available from: https://gamma-app.de/sample-page/

9. [No title] [Internet]. [cited 2019 May 20]. Available from: https://buildingsmart.no/sites/buildingsmart.no/files/2013_ntnu_martin_victor_nagy.pdf

10. [No title] [Internet]. [cited 2019 May 20]. Available from: https://d2f99xq7vri1nk.cloudfront.net/legacy_app_files/pdf/SimAUD2011-LifeBCa.pdf

11. [No title] [Internet]. [cited 2019 May 20]. Available from: https://pdfs.semanticscholar.org/aafb/1b13efd7fc6b916e148f49425a8cb4696544.pdf

12. JSON:API — A specification for building APIs in JSON [Internet]. [cited 2019 May 20]. Available from: https://jsonapi.org/

13. Website [Internet]. [cited 2019 May 20]. Available from: [https://developers.google.com/protocol-buffers/

14. Contributors to Wikimedia projects. Octree - Wikipedia [Internet]. Wikimedia Foundation, Inc. 2004

[cited 2019 May 20]. Available from: https://en.wikipedia.org/wiki/Octree

15. Kotlin vs. Java: 9 Benefits of Kotlin for Your Business | Udemy for Business [Internet]. Udemy for Business. 2018 [cited 2019 Jan 23]. Available from: https://business.udemy.com/blog/kotlin-vs-java-9-benefits-of-kotlin-for-your-business/

16. Wodehouse C. Swift vs. Objective-C | What Is The Difference? [Internet]. Hiring | Upwork. 2015 [cited 2019 Jan 23]. Available from: https://www.upwork.com/hiring/mobile/swift-vs-objective-c-a-look-at-ios-programming-languages/

17. Apple Inc. Keynote - WWDC 2014 - Videos - Apple Developer [Internet]. Apple Developer. [cited 2019 May 20]. Available from: developer.apple.com/videos/play/wwdc2014/101/

18. Mobile game development: best tools and advice | Thinkmobiles [Internet]. Thinkmobiles. [cited 2019 May 20]. Available from: https://thinkmobiles.com/blog/mobile-game-development-tools/

19. Mobile game development: best tools and advice | Thinkmobiles [Internet]. Thinkmobiles. [cited 2019 May 20]. Available from: https://thinkmobiles.com/blog/mobile-game-development-tools/

20. Unity Public Relations Fact Page [Internet]. Unity. [cited 2019 May 15]. Available from: https://unity3d.com/public-relations

21. Molenaar R. Augmented Reality & React Native [Internet]. Medium. Lightbase; 2018 [cited 2019 Apr 3]. Available from: https://medium.com/lightbase/augmented-reality-react-native-6385b6c2fad2

22. Fitzsimmons M. What is ARKit 2? Here's what you need to know about Apple's latest AR update [Internet]. TechRadar. TechRadar; 2018 [cited 2019 May 15]. Available from: https://www.techradar.com/news/what-is-arkit-2-heres-what-you-need-to-know-about-apples-latest-ar-update

23. ARKit | Apple Developer Documentation [Internet]. [cited 2019 May 15]. Available from: https://developer.apple.com/documentation/arkit

24. Fundamental concepts | ARCore | Google Developers [Internet]. Google Developers. [cited 2019 May 15]. Available from: https://developers.google.com/ar/discover/concepts

25. Contributors to Wikimedia projects. Tango (platform) - Wikipedia [Internet]. Wikimedia Foundation, Inc. 2014 [cited 2019 May 15]. Available from: https://en.wikipedia.org/wiki/Tango_(platform)

26. Guide to app architecture | Android Developers [Internet]. Android Developers. [cited 2019 May 10]. Available from: https://developer.android.com/jetpack/docs/guide

27. Swift by Sundell [Internet]. Swift by Sundell. 2019 [cited 2019 May 20]. Available from: https://www.swiftbysundell.com/

28. Mohamed A. Protocol-Oriented Network in swift [Internet]. Medium. SwiftCairo; 2018 [cited 2019 May 20]. Available from: https://medium.com/swiftcairo/protocol-oriented-ios-network-layer-422575314cc2

29. Scrum (software development) - Wikipedia [Internet]. [cited 2019 Jan 23]. Available from:

https://en.wikipedia.org/wiki/Scrum_(software_development)

30. Xamarin App Development with Visual Studio | Visual Studio [Internet]. Visual Studio. [cited 2019 May 20]. Available from: https://visualstudio.microsoft.com/xamarin/

31. Stoll S. In plain English: So what the heck is Flutter and why is it a big deal? [Internet]. Medium. Flutter Community; 2018 [cited 2019 May 20]. Available from: https://medium.com/flutter-community/in-plain-english-so-what-the-heck-is-flutter-and-why-is-it-a-big-deal-7a6dc926b34a

32. Eisenman B. Learning React Native. O’Reilly Media, Inc.;

33. Native UI Components · React Native [Internet]. [cited 2019 May 20]. Available from: https://facebook.github.io/react-native/

34. Ranking - AnTuTu Benchmark - Know Your Android Better [Internet]. [cited 2019 May 19]. Available from: http://www.antutu.com/en/ranking/rank1.htm

9 Appendixes

9.1 attached zip file

- Reports
- Meetings with supervisor
- Meetings with product owner
- Status reports
- User survey
- Full benchmark sheet
- Interviews

Appendix A

Main gateway

Login (return: json)

The login route takes a username and password and returns a JWT. This JWT is then attached to every request done from the application.

Project-links (return: json)

A request to this endpoint was made to display the projects a user has access to. This route returns the project name and ID.

From project endpoint (Requires project-id)

buildings (return: json)

Returns all the buildings from a given project-id

floors (parameters: floorID, return: json)

Return all the floors for the project, we used this alongside the modifier filter[building] to ensure we only got floor data for the building we selected.

floorplan (parameters: floorID, return: json)

Floorplan returns information about the tiles used. We use this to ensure that the tile map works correctly.

tile (parameters: floorID, return: json)

The tile itself, this is a small tile picture for the floorplan minimap.

info (parameters: layer, return:json)

Returns the octree model for the building.

node (parameters: layer, detail, id, return: protobuf)

Returns the 3D Data for a given octree node

materials (parameters: layer, return: plaintext)

Returns the materials for the 3D objects

Appendix B

	Xamarin	Flutter	React Native
Backing company	Microsoft	Google	Facebook
Opensource	SDK's only	Yes	Yes
Language	C#, with limited support for native bindings	Dart, with native bindings	Javascript, with native bindings
Architecture patterns	MVC, MVVM	MVP, MVC, MVVM, Flux	MVP, MVC, MVI, MVVM, Flux
Hot reload	No	Yes	Yes
Application performance	4/10	9/10	5/10
Libraries and tooling	6/10	5/10	7/10
UI	4/10	9/10	7/10
Documentation	10/10	10/10	10/10
Community	6/10	7/10	9/10
Maturity	10/10	5/10	9/10
Overall score	40/60	45/60	47/60

Appendix C

```
#import "React/RCTViewManager.h"

@interface RCT_EXTERN_MODULE(ARViewManager, RCTViewManager)
  RCT_EXPORT_VIEW_PROPERTY(arMode, BOOL)
  RCT_EXPORT_VIEW_PROPERTY(building, STRING)
  RCT_EXPORT_VIEW_PROPERTY(project, STRING)
  RCT_EXPORT_VIEW_PROPERTY(token, STRING)
@end
```

Appendix D

```
#!/bin/bash

#Path to swiftlint
SWIFT_LINT=/usr/local/bin/swiftlint

#if $SWIFT_LINT >/dev/null 2>&1; then
if [[ -e "${SWIFT_LINT}" ]]; then
    count=0
    for file_path in $(git ls-files -m --exclude-from=.gitignore | grep ".swift$");
do
        export SCRIPT_INPUT_FILE_$count=$file_path
        count=$((count + 1))
    done

##### Check for modified files in unstaged/Staged area #####
    for file_path in $(git diff --name-only --cached | grep ".swift$"); do
        export SCRIPT_INPUT_FILE_$count=$file_path
        count=$((count + 1))
    done

##### Make the count avilable as global variable #####
    export SCRIPT_INPUT_FILE_COUNT=$count

    echo "${SCRIPT_INPUT_FILE_COUNT}"

##### Lint files or exit if no files found for lintint #####
    if [ "$count" -ne 0 ]; then
        echo "Found lintable files! Linting and fixing the fixible parts..."
        $SWIFT_LINT autocorrect --use-script-input-files --config .swiftlint.yml
#autocorrects before commit.
    else
        echo "No files to lint!"
        exit 0
    fi

    RESULT=$?

    if [ $RESULT -eq 0 ]; then
        echo ""
        echo "Violation found of the type WARNING! Must fix before commit!"
    else
        echo ""
        echo "Violation found of the type ERROR! Must fix before commit!"
    fi
    exit $RESULT

else
#### If SwiftLint is not installed, do not allow commit
    echo "warning: SwiftLint not installed, download from
https://github.com/realm/SwiftLint"
    echo "If you have Homebrew, you can directly use `brew install swiftlint` to
install SwiftLint"
    exit 1
fi
```

Appendix E

Interview

Pål-Robert Engnæs

This interview was conducted to obtain information about future plans for the work conducted in the thesis, technical expectations and general technical information that could be applied in the thesis report. The interview was conducted 07.05.2019 with Rendra CTO Pål-Robert Engnæs. He has been involved with Rendra since January 2013 and has had the roles of CTO and software developer focusing on backend, devops and 3D processing. The interview was conducted in Norwegian.

Hva var forventningene fra Rendra sin side i starten av prosjektet? (eks kvalitet, ferdig mvp, osv...)
Vi så et utfordrende prosjekt med høy risiko, hvor viktigste forventning var å få avklart flest mulig av snublesteinene og utfordringene som må løses på veien mot AR i et B2B-produkt for ikke-tekniske brukere. Vi forventet, eller kanskje håpet, at gruppa ville komme fram til å kunne vise en StreamBIM modell noenlunde riktig forankret med det fysiske bygget, i kombinasjon med en solid liste med "future work".

Var det meninger om at oppgaven hadde et veldig mye bredere scope enn hva som var forventet av en gruppe studenter for en slik oppgave?
Ledende spørsmål. ;-) Gruppa gikk nok optimistisk ut, man kunne ha startet med en enkel statisk modell istedenfor octtree fra eksisterende StreamBIM servere, man kunne ha valgt én av iOS eller Android - det var tidvis to samarbeidende grupper her, én på hver mobilplattform. Forenkling her ville nok gitt mer tid til å fokusere på forankring, som var forventet å bli det vanskeligste temaet. Men dette blir etterpåklokskap.

Var det forventet en fungerende MVP på slutten av oppgaven? Hvis ikke, hva var forventet?
Nei. Med MVP, "minimum viable product", forstår jeg noe som kan lanseres ut til kunder; smalt scope men med god kvalitet på det som tilbys. Proof of concept, med en dose lærdom og erfaringer, er mer beskrivende for hva vi forventet.

Endret forventningene seg underveis i prosjektet? Isåfall hvordan?
Jeg heller mot å si nei. **Forhåpninger** endret seg, konkret falt håpet om felles kodebase for to plattformer fort bort, deretter viste forankring av modell mot virkelighet seg å være enda vanskeligere enn håpet.

Er det noen aspekter av oppgaven dere mener burde vært mindre eller mer prioritert?
Vi har aldri tenkt at gruppa burde prioritert annerledes, men med etterpåklokskap ser vi at prosjektet kunne ha blitt bedre med lavere ambisjoner på integrasjon med StreamBIM og mobilplattformer og mer fokus på forankring. Et tidligere dykk i teori bak forankring opp mot særegenheter ved domenet (bygningsmodeller) kunne ha hjulpet.

Er det noe som har kommet frem i arbeidet som har vært spesielt positivt?
Selv om gruppa har revet seg i håret over f.eks. minnebegrensninger på Android synes vi det er lovende hvor store modeller man kan vise selv uten noen som helst form for optimalisering. Med strømming, occlusion culling, level-of-detail og filtrering på toppen - som vi bruker i produktet - ser vi optimistisk på håndtering av selv de største byggeprosjektene.

Er det noe som har kommet frem i arbeidet som har gjort dere bekymret? (eks: Android minne restriksjoner).
Det er fortsatt en lang vei over hvite deler av kartet for å komme fram til en AR-løsning som er enkel og kjapp nok til at bygningsarbeidere gidder, samtidig som den er nyttig og robust nok til at de ser verdien.

Hva synes Rendra om det utførte arbeidet, og hva mener dere om samarbeidsprosessen? Hvordan kunne dette eventuelt blitt gjort bedre?

Gruppa har jobbet overraskende nær hva vi forventer av egne ansatte. Det gjelder hvordan man setter mål og gjør teknologiske valg, arbeidsprosesser når flere utviklere jobber på samme kodebase, jevnlig vurdering av status og justering av retningen videre. Jeg tror både gruppa og vi i Rendra hadde fått enda mer ut av prosjektet hvis vi hadde jobbet mer på samme sted.

Hva synes Rendra om gruppens valg av teknologier i dette prosjektet?

Jeg mener gruppa har gjort godt funderte valg hele veien. Det betyr ikke at alle valg har vist seg geniale, men at de har satt seg inn i og vurdert alternativer og vurdert dem mot hverandre ut fra hvilke kriterier som er viktige for dette prosjektet. Ingen hopping etter "the latest greatest" på buzzword-toppen.

Hvordan tenker Rendra å bruke arbeidet videre? (eks: om det er verdt for dere å lage en native løsning videre, om dere ville lagd en custom cross-platform rendering engine som kunne blitt implementert i NDK på Android for å slippe minne restriksjonene, osv...)

Det er litt tidlig å si. Men det sitter langt inne å gjøre tung utvikling parallellt i flere plattformspesifikke kodebaser. Native rendering engine under Javascript har litt bedre odds enn andre alternativer; cross platform kode er bra, men selv native blir det mye plattformspesifikt. Vi kan ende med å måtte velge én plattform.

Er det noe spesifikt Rendra vil ha utdypt i "future work" seksjonen av oppgaven?

Hva er spesielt vondt og vanskelig med hver av plattformene. Tips om eventuelle nye verktøy/løsninger som bør undersøkes før vi tar teknologivalg. Idéer om forankring og UI.

Hva synes dere om arbeidet dere måtte utføre fra deres side under prosjektet? (eks: eksponering av space-geo, og problemer med en IFC fil, eller annet)
Jeg hadde egentlig forventet flere ønsker om features og felles arbeidsøkter. :-)

Med erfaring etter dette samarbeidet med en bachelorgruppe, er det noen råd dere ville gitt en ny gruppe med studenter dersom det skulle blitt utført en ny bacheloroppgave hos dere?
Én: kutt scope til det minste tenkelige, til det gjør vondt i hjertet, og fjern deretter halvparten. Scope creep midtveis er lett som en plett dersom man har kuttet for mye.

To: prøv å veve prosjektet tettere sammen med det daglige arbeidet i bedriften så blir det naturlig mer kontakt og læring begge veier. Ok, dette er vanskelig å få til, men verd en ekstra runde i tenkeboksen.

Kan du kort utdype din nåværende rolle og eventuelt tidligere roller i Rendra?
Har hatt cirka samme rolle hele tida, nærmer seg 5 år i Rendra. Bruker dagene mest som senior utvikler, fokus på backend, devops og 3D-prosessering. Kombinert med tittelen CTO som er nynorsk for "han som må gå i flest møter". Som bonus har jeg en ekstra finger på rattet ifht roadmap og teknologivalg.