NTNU – Trondheim
Norwegian University of
Science and Technology

The Potential of Utilizing BIM Models With the WebGL Technology for Building Virtual Environments

A Web-Based Prototype Within the Virtual Hospital Field

Lars Madsen Hestman

Master of Science in Informatics
Submission date: July 2015
Supervisor: Theoharis Theoharis, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In 2011 a standard for 3D rendering in web browsers saw the light of day. The WebGL technology allows the browsers to utilize the hardware on the device to accomplish hardware accelerated graphics. As GPUs are constantly getting more powerful, both on desktops and mobile devices, this opens up a whole new world of possibilities for developing complex web based 3D application. A large increase in the adoption of Building Information Modeling (BIM) in the construction industry over the last decade means that there exist a large amount of 3D models of real buildings. The question is whether these models can be utilized for other purposes than building planning.

In this thesis a web based prototype was developed using the WebGL technology, with BIM models of a building at St. Olav's University Hospital in Trondheim as the foundation for the 3D scene. By testing the prototype on different platforms with a selection of different web browsers this thesis evaluates to which extent the technology is mature enough to apply complex BIM models in a game-like environment, and how it can be utilized in a virtual hospital context.

Sammendrag

I 2011 så en standard for 3D-rendering i nettlesere dagens lys. WebGL-teknologien gjør det mulig for nettlesere å utnytte maskinvaren på enheten til å oppnå hardware akselerert grafikk. Ettersom GPUer stadig blir kraftigere, både på stasjonære maskiner og mobile enheter, åpner dette opp en helt ny verden av muligheter for å utvikle komplekse nettbaserte 3D-applikasjoner. En stor økning i anvendelse av bygningsinformasjonsmodellering (BIM) i byggebransjen det siste tiåret betyr at det eksisterer en stor mengde 3D-modeller av virkelige bygninger. Spørsmålet er om disse modellene kan utnyttes for andre formål enn byggeplanlegging.

I denne avhandlingen ble en web-basert prototype utviklet ved hjelp av WebGL teknologien, med BIM-modeller av en bygning ved St. Olavs Hospital i Trondheim som grunnlaget for 3D-scenen. Ved å teste prototypen på ulike plattformer med et utvalg av ulike nettlesere vil denne avhandlingen evaluerer i hvilken grad teknologien er moden nok til å anvende komplekse BIM-modeller i et spillignende miljø, og hvordan det kan bli utnyttet i en virtuelt sykehus kontekst.

Preface

This is a master's thesis in Informatics, within the specialization Game Technology. It has been written at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU) in Trondheim, during fall of 2014 and spring of 2015.

First of all I would like to thank my supervisor Frank Lindseth, Senior Research Scientist at SINTEF Medical Technology and associate professor II at IDI, for his guidance and discussions throughout the process. I would also like to thank Tor Åsmund Evjen, at St. Olav Eiendom, for his input in the discussion and assistance in acquiring the BIM models. Lastly I would like to thank Professor Theoharis Theoharis at IDI for assuming the role as my formal supervisor.

Lars Madsen Hestman
Oslo, June 2015

Table of Contents

List of Tables

List of Figures

Listings

Abbreviations

3D	=	Three-dimensional space
Add-on	=	Extension to a software
API	=	Application Programming Interface
App	=	Application (mobile application)
ARK	=	Architectural BIM discipline
BIM	=	Building Information Modeling
CAD	=	Computer-aided design
CPU	=	Central Processing Unit
CSS	=	Cascading Style Sheets
FPS	=	Frames per Second
GNU GPL	=	GNU General Public Licence
GPU	=	Graphic Processing Unit
HTML	=	HyperText Markup Language
IARK	=	Interior design BIM discipline
IDE	=	Integrated Development Environment
IE	=	Internet Explorer
IFC	=	Industry Foundation Classes
JS	=	JavaScript
JSON	=	JavaScript Object Notation
Lag	=	Latency / delay / slow response
MIT	=	Massachusetts Institute of Technology
NTNU	=	Norwegian University of Science and Technology
OS	=	Operating System
Plug-in	=	Extension to a software
RAM	=	Random Access Memory
RIE	=	Electrical BIM discipline
Spec	=	Technical specification of a machine
WebGL	=	Web Graphics Library

Chapter 1

Introduction

1.1 Motivation

"Users can accomplish many tasks on today's Web, from purchasing products to interacting in real-time with users throughout the world. However, one key element has yet to make its mark on the Web: 3D. Today, 3D is primarily used online in applications such as games and virtual worlds, which are rendered using powerful computers and specialized software. (..) Users want their browser-based experiences to be more like those they have on a PC. Consumers are becoming more accustomed to 3D content because of the use of the technology in movies, videogames, and other types of entertainment.
(..)
There is thus demand for more and easier-to-access 3D content on the Web. (..) And the better the browser experience, the more potential revenue that online content could generate for providers and others. However, 3D on the Web remains primitive today because the complex technology has been difficult to use with typical PCs and browsers. (..) In fact, browsers generally cannot natively run complex 3D content or offer either high frame rates or full-screen graphics.
(..)
Now, though, several organizations are working on technologies that may finally widen 3D's presence on the Web by transforming browsers into more powerful computing platforms that can deliver a PC-like experience, including the playing of 3D content. This would enable applications such as product modeling, presentation, and configuration; 3D online meetings and worker collaboration; the simulation of processes such as surgery or mechanical procedures; virtual tours; and augmented reality. Nonetheless, 3D on the Web will have to clear some obstacles before the technology can become reliable

and mainstream."

These words are from an article (Ortiz, 2010) in IEEE Computer back in 2010. One year later the world saw the release of WebGL, which became the standard for 3D rendering in web browsers. Still, 3D content is not something you see all over the web today, and the most noticeable 3D applications and games are still designed as stand alone applications. Is there a reason to believe it will stay like this in the future, or will more and more 3D applications be available in the browser?

Accurately predicting trends in the future is not always easy, and especially in the field of IT where the next big thing may be just around the corner. Regardless of that, the probability that web development will only become more popular is something numerous people in the IT sector believe. Glenn Romanelli of Ligthous Design, Inc claims that native mobile apps are on the decline, and that their clients find it more cost effective developing website applications that works across platforms (Stangarone, 2013). One of the main drawbacks of web development historically have been the lack of performance and robustness of browser applications. With the continuous increase in computer hardware's performance, and the release of technology like WebGL, are we finally at a point where web browsers, both for desktop and mobile, are mature enough to render complex 3D scenes at an acceptable frame rate?

Figure 1.1: Virtual operating room in Second Life, from Imperial College London's virtual hospital

Another concept that gets more and more popular and integrated in education and training is virtual reality. Flight simulators are examples of virtual training that have been around for ages. In later years use of a virtual environment for training purposes is getting more and more common in other professions as well. Armed forces, emergency services and the health sector are examples of fields were the potential of virtual training is looked upon with great interest. The ever increasing interest for web technology and virtual reality, and how these can be combined, has been the main motivating factor for this work.

1.2 Initial project ideas

The objective of this thesis started out quite loosely, but with a aim of combining WebGL with actual building data from St. Olav's University Hospital in Trondheim to create a recognizable authentic virtual environment playable from the browser. One of the main ideas in the starting phase was to build on the already existing *Virtual Operating Room*, worked on by former and current master students at NTNU (Kleven, 2014), created in the online virtual world application Second Life. The purpose of Virtual Operating Room is to provide training for nurse anesthetists through a role-playing game. This solution has already been put to use at St. Olav's. For this project the idea was to port this concept to a browser application, and use real building data from St. Olav's to get closer to the actual surroundings than possible in Second Life.

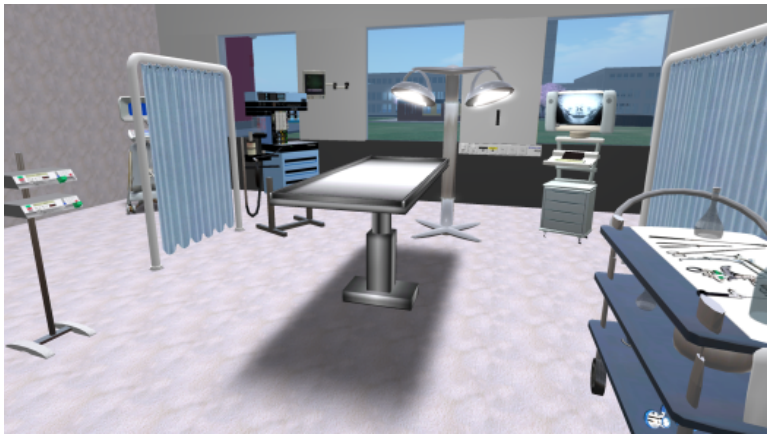


Figure 1.2: Virtual operating room in Second Life, from St. Olav's Hospital (Kleven, 2014)

Along the way this project had to part from the Virtual Operating Room idea as St. Olav Eiendom, responsible for exercising the strategic ownership of real property on St. Olav's, could not provide 3D models for rooms/buildings related to surgery or other clinical facilities. St. Olav Eiendom were however interested in the project and throughout meetings it was decided to create an prototype using 3D models of an administrative building at the hospital.

1.3 Project description

In this thesis the focus is on exploring the possibilities WebGL brings to modern web development, and how BIM models can be utilized and integrated in this context. The main goal is to develop a web-based prototype using the WebGL API and use BIM models as the foundation for the 3D scene. The prototype will work as a proof-of-concept of possibilities 3D WebGL applications can offer in the virtual hospital context. It will include objects to

interact with that likely could feature in the virtual hospital arena, either in the category of logistics, optimization, education, training or simulation.

1.4 Research questions

Besides creating a prototype this thesis will look at a number of research questions related to WebGL development and the browser technology used to present WebGL applications.

1. In what way is use of BIM models suitable with WebGL?
2. The major web browsers compatibleness with WebGL:
   (a) Which support the technology?
   (b) How do they compare performance-wise rendering 3D using WebGL?
3. What exists of higher-level WebGL frameworks today, and how do they compare?
4. How does the WebGL technology adapt to the wide range of devices of different performance capabilities?

1.5 Research method

The work behind this thesis started out with a literature study to get a good understanding of the technology that were to be used for developing the prototype, and look for related work. The main sources gathered in the literature study consists of books, articles and technical reports, and are for the most part located by using Google Scholar. For some of the more narrow technological facets discussed in the thesis there was not much academic published work, thus blogs etc. from developers within the field were used.

The next step after the literary study was to develop the prototype. An analysis made of existing higher-level WebGL frameworks was taken into account when choosing the development approach and technology for the project. After the development process was finished the prototype was used to conduct both usability tests of the prototype, and benchmark testing of desktop browsers. A less comprehensive test for mobile devices was also carried out. The results from these tests together with the sources gathered in the literature study formed the basis for the discussion and conclusion.

1.6 Report outline

This report starts with presenting the results of the preliminary study, in **chapter 2**. The preliminary study is a literature review which aims to elaborate on WebGL and BIM, and look at related work. Finally a selection of higher-level WebGL frameworks will be presented and compared.

Chapter 3 will focus of the developmental part of this thesis, describing the purpose of the prototype, which technology and tools that have been used, how the prototype is implemented, and address topics that needs to be handled when developing a web-based 3D application.

The results of this thesis will be presented in **chapter 4**. This chapter will give a description of the prototype developed, and presents the results of a usability test, a benchmark browser test and a mobile test. The results will also be discussed in this chapter.

Chapter 5 will bring a further discussion with the defined research questions in section 1.4 as the main focus. The results from the previous chapter will also be used to underpin the statements made.

Finally, **chapter 6** will deliver the conclusion of this thesis, and address future work related to the thesis.

There are three appendices to the thesis. **Appendix A** presents screenshots from the prototype, the questionnaire from the usability test is presented in **Appendix B**, while the source code is provided in **Appendix C**

Chapter 2

Preliminary study

As a starting point for this master thesis a preliminary study have been conducted. The purpose with this study was to get to know the WebGL technology before starting the development process. This chapter will first examine and define the concepts of WebGL and BIM, and then look at previous work related to the topic. Finally there will be an thorough study of high-level WebGL frameworks.

2.1 WebGL

Web Graphics Library, better known as WebGL, is a low-level JavaScript API for rendering 3D and 2D graphics in the web browser. It is developed and maintained by the Khronos Group who, on their website, defines the technology in this manner:

> *"WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES 2.0, exposed through the HTML5 Canvas element as Document Object Model interfaces. Developers familiar with OpenGL ES 2.0 will recognize WebGL as a Shader-based API using GLSL, with constructs that are semantically similar to those of the underlying OpenGL ES 2.0 API. It stays very close to the OpenGL ES 2.0 specification, with some concessions made for what developers expect out of memory-managed languages such as JavaScript."*

The history of WebGL goes back to 2007 when software-engineer Vladimir Vukicevic, at Mozilla, started working on a prototype, called Canvas 3D, for the upcoming HTML <canvas> element (Cantor and Jones, 2012). His work led to a project at the Khronos Group, where the goal was to create a specification for granting web browsers access to the Graphic Processing Units (GPU), thereby achieve accelerated 3D graphics on the web.

This project resulted in an initial release of WebGL in March 2011, while the first stable release came 2 years later, in March 2013.

As the release dates indicates WebGL is a quite modern technology, and it is also most likely a game changer for web development. But rich graphics and 3D rendering have been done in web browser way before WebGL came around, so what makes this technology special? First of all, rich graphics technology prior to WebGL needed the user to install plug-ins or browser extension to work. According to Vukicevic many users finds plug-ins inconvenient to install, troubleshoot and manage, and therefore prefer not to use them (Ortiz, 2010). WebGL does not need any plug-in, it just works! The reason why will be elaborated in subsection 2.1.2. The graphics capabilities are also a big improvement as WebGL enables access to hardware accelerated graphics. Marcus Krüger, founder and Executive Chairman of Goo Technologies, argues that WebGL is the largest leap in capability in the history of the web (Krüger, 2014).

2.1.1 Technical definition

WebGL is a browser version of OpenGL, based on OpenGL ES 2.0. The ES version is tailored for embedded systems like phones and tablets, and WebGL is built on this principle to more easily achieve a consistent, cross-platform, cross-browser 3D API for the web (Parisi, 2012). WebGL is accesses through JavaScript calls, and displayed in the browser by using the <canvas> element in HTML5. It is therefore easily combined with other web content, for example by placing it in a tag, using half page to display the canvas and the other half for other html elements.

Hardware based rendering is keyword when talking about WebGL. Previous 3D rendering plug-ins to browsers, e.g. Flash, have been using *software based rendering*, which means the rendering takes place solely in the CPU (Central Processing Unit). WebGL uses so called hardware based rendering. What we mean by this is that dedicated hardware, being the GPU (Graphical Processing Unit), takes responsibility for the rendering. This approach is in the vast majority of cases much more efficient than software based rendering (Cantor and Jones, 2012). *Client based rendering* is another characteristic of WebGL, meaning the rendering process takes place locally. The 3D model data is downloaded from server, but the processing required to obtain the image is done by the client's hardware. The opposite solution is *server side rendering* where the rendering process takes place remotely, on dedicated servers, and the resulting image or set of images are transmitted to the client. From these two properties of WebGL (hardware based and client based rendering) it becomes clear that the capabilities of the device running the application is decisive for the performance. This case will be further discussed later in the thesis (section 5.1).

2.1.2 Supported browsers

WebGL works without installing extra plug-ins or add-ons. This is because WebGL is regarded as the new standard for rendering 3D graphics on the web (Parisi, 2012), and

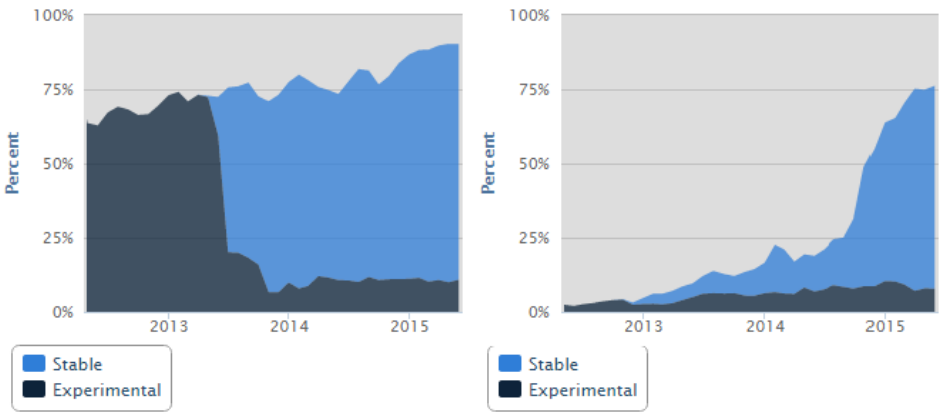
therefore the browser providers have committed significant resources to develop and support the technology. As of February 2015 all major web browsers fully or partially support WebGL. What is understood by partially support is that there might be other factors which will decide the availability of WebGL as well, for example the client's GPU or operating system. Following are two tables, Table 2.1 and Table 2.2, that provides an overview over WebGL availability on the most used web browsers for desktops and mobiles.

Desktop browsers	
Internet Explorer	Internet Explorer had no WebGL support until IE 11. Initially IE 11 did not even pass half of the tests in WebGL conformance test suite (McCoy, 2013), but several updates have made IE 11 compatible with WebGL. There also exists third-party plugins to provide WebGL support for older versions of Internet Explorer.
Google Chrome	WebGL has been partially supported in Chrome since Chrome 8, and enabled by default with the release of Chrome 9 in February 2013. WebGL has been fully supported since Chrome 18 was released in March 2012 (Rosenblatt, 2012).
Safari	Safari has partially supported WebGL since Safari 5.1 for OS X Snow Leopard, and Safari 6.0 for OS X Lion and OS X Mountain Lion. These versions were relased in respectively July 2011 and July 2012, but WebGL were disabled by default until the Safari 8 release October 2014. There is only WebGL support for Safari browsers running OS X operating systems.
Mozilla Firefox	Since version 4 of Firefox, launced in March 2011, WebGL has been enabled on all platforms with capable GPUs.
Opera	Opera has partially supported WebGL since release of version 12 in June 2011.

Table 2.1: Desktop web browsers and WebGL support

Mobile browsers	
Internet Explorer	Similar to the desktop version WebGL is supported from IE 11 on Windows Phone 8.1
Google Chrome	Chrome for Android has supported WebGL since version 25, released February 2013, and been enabled by default since version 30, October 2013. There are however mostly the newer phones that support it. As of February 2015 there is still no WebGL support for the iOS chrome browser.
Android Browser	The Android Browser partially supports WebGL, but only on some of the Samsung and Sony Ericsson devices. Google Chrome is also replacing the Android Browser in a long range of the Android phones.
iOS Safari	Safari's mobile browser for iOS has supported WebGL since the launch of iOS 8 in September 2014 (Pesce, 2014).
Firefox Mobile	The Firefox browser for mobile is only available for Android, and has had fully WebGL support since the release of version 4, in March 2011.
Opera Mobile	WebGL hva been supported for Android only since Opera Mobile 12 was released in February 2012.
Opera Mini	Opera's microbrowser does not support WebGL.
Blackberry browser	The blackberry browser has supported WebGL since the release of the Blackberry OS 10 in January 2013.

Table 2.2: Mobile web browsers and WebGL support



(a) WebGL for desktop

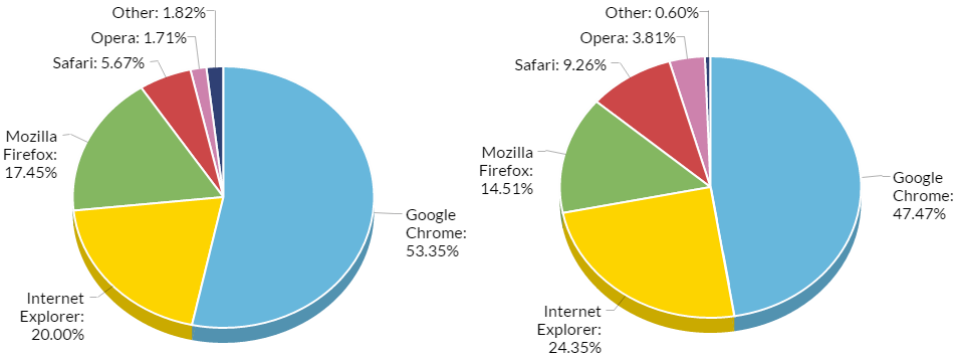
(b) WebGL for mobile devices

Figure 2.1: Proportion of visitors with WebGL support

The website webglstats.com collects data from web users through over 60 other websites, using a tracker frame embedded on these sites. As of May 2015 their data collection shows

that nearly 90 % of the visits to theses sites have the ability to run WebGL content. For desktops the percentage have been high for a long time and was as of May 2015 at 88.5 % (Figure 2.1a). The evolution for mobile devices is more interesting. In May 2012 only 2 % of the visits by mobile devices had the ability to run WebGL content. Three years later, as of May 2015, the share was 76.2 % (2.1b).

Looking at the user population Chrome is the most popular desktop browser by a wide margin, with over 50 % of the share. A complete overview of the worldwide market shares for desktop browser today (June 2015) is given in Figure 2.2a. Figure 2.2b shows the market share for Norway. The data are gathered by the web traffic analysis tool StatCounter.



(a) Worldwide **(b)** Norway

Figure 2.2: Desktop browser market share

2.2 Building information modeling

Building information modeling, better known through the acronym BIM, is naively explained a process resulting in a 3D representation of an existing building or a building project. Naively because it is more to it than just a 3D model. Aranda-Mena et al. (2009) found that BIM is an ambiguous term, that it means different things to different professionals. For some BIM is a software application, for others a process for designing and documenting building information. Another group regards it as a whole new approach to practice and advancing the profession which requires the implementation of new policies, contracts and relationship among project stakeholders.

The US National Building Information Model Standard Project Committee defines BIM in this way:

> *"Building Information Modeling (BIM) is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition.*

> *A basic premise of BIM is collaboration by different stakeholders at different phases of the life cycle of a facility to insert, extract, update or modify information in the BIM to support and reflect the roles of that stakeholder.*
>
> *The US National BIM Standard will promote the business requirements that BIM and BIM interchanges are based on:*
>
> - *a shared digital representation,*
> - *that the information contained in the model be interoperable (i.e.: allow computer to computer exchanges), and*
> - *the exchange be based on open standards,*
> - *the requirements for exchange must be capable of defining in contract language."*

Builing Information Modeling can be seen as the process of developing a complete Building Information Model (Hergunsel, 2011), that is a shared digital model covering all aspects of the construction. What this gives is a common understanding and improved visualization of the project. This can improve coordination and productivity, reduce the risk of mistakes and discrepancies, and minimize abortive costs. All the structural components in BIM creates a 3D scene, but BIM itself operates up to as much as 6 dimensions. The 4^{th} dimension projects time allocation and scheduling data, while the 5^{th} dimension adds cost estimation (infoComm, 2011). The 6^{th} dimension deals with the building data beyond completion, as for example maintenance cost and energy saving.

A BIM model is composed by data from multiply fields within the building planning and can be divided into different disciplines. The different disciplines, or roles, identifies what the sub-model deals with. For example you have a model that represent the architectural aspect, like walls, doors, windows etc., a model that deals with the plumbing, a model that represents interior design and so on. Together all these parts constitute the complete BIM model of the building project. Table 2.3 presents a selection of roles in the BIM process. The definitions are retrieved from Statsbygg's, the Norwegian Directorate of Public Construction and Property, manual for general guidelines for BIM (Statsbygg, 2009). The acronyms used to describe the disciplines are the Norwegian, not the international, standard and these are the ones that will be referred to in this thesis.

Role	**Description**
ARK	Architect
RIE	Electrical engineering (electricity, Telecom, automation etc.)
RIV	HVAC (heating, ventilation and air conditioning)
RIB	Building engineering
IARK	Interior designer
RIG	Geotechnics

Table 2.3: Examples of BIM disciplines

2.2.1 History and future of BIM

The history of BIM goes centuries back. The term Building Information Model first appeared in a paper in 1992 (van Nederveen and Tolman, 1992), but the concept was devised in the 60s, and the first BIM-like software to be released was Radar CH, in 1984, later known as ArchiCAD. However this product or other BIM software to be released were not in widely use through the 80s and the 90s. A problem encountered in the earlier days was the variety of programs and file formats used by different architects and engineers, which made collaboration difficult (Bergin, 2012). This resulted in the development of the Industry Foundation Class (IFC) file format in 1995, and this file format is the only truly open standard for BIM today, described by the International Standard ISO 16739:2013.

It was after the millennium that the use of BIM really accelerated, and especially in the period between 2007 to 2012. In North America the industry-wide adoption of BIM by architect, engineering and contractor companies increased from 28% in 2007 to 71% in 2012 (McGraw-Hill, 2012). If we restrict it to the bigger companies the adoption is even greater, as Figure 2.3 shows.

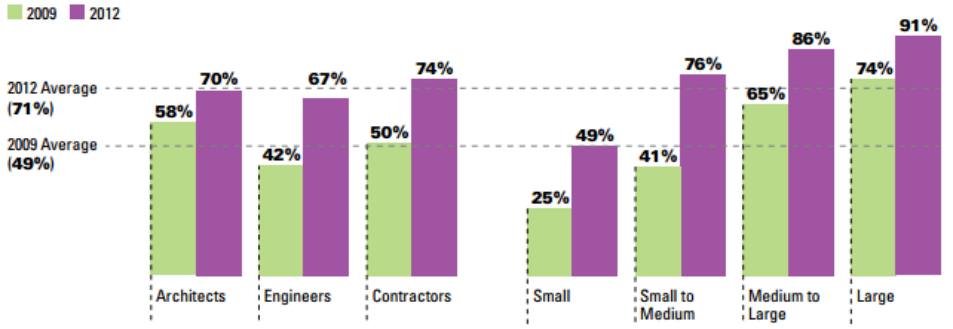


Figure 2.3: BIM adoption from 2009 - 2012 (McGraw-Hill, 2012)

In 2011 Paul Morrell, the United Kingdom government's chief construction adviser at the time, gave an interview to *Building*, the United Kingdom's largest magazine for the building industry. In this interview he claimed that BIM would be made mandatory on virtually all government projects within five years (Withers and Matthews, 2011).

2.2.2 Use of BIM in this project

St. Olav Eiendom has in the later years done a initiative in creating BIM models of their existing buildings. The level of detail in the models differs from facility to facility. The building that is represented in this project is the 1930-building, an administrative building over 5 levels. The modelling is still in progress, but the most part is completed. The 3D model for this building includes architecture (ARK), electrics (RIE) and HVAC (RIV), while interior (IARK) is in progress. HVAC will not be used as it will occupy more resources than it will give back in visible elements. The architectural model is naturally

the most essential part, while electrics (RIE) and interior will apply details to make the 3D world richer.

As discussed a BIM model can be as complex as a 6D models. The 4^{th}, 5^{th} and 6^{th} dimension are not of any particular interest in this project. It is the physical structures and objects of the building that are the interesting part. The time scheduling and cost of the building project have no relevance for the application.

2.3 Related work

This section will introduce work that is related to this thesis' focus on WebGL development and integration of BIM models

2.3.1 Existing WebGL applications

2.3.1.1 Zygote Body and The BioDigital Human

There are some examples of WebGL application within the medical field. Two very similar and well-developed solutions that presents a 3D view of the human body are called *Zygote Body* and *The BioDigital Human*. Both applications presents a human body, either male or female, with the possibility to display or hide the different systems of the body, being the integumentary, muscular, skeletal, respiratory, digestive, urinary, endocrine, reproductive, cardiovascular, lymphatic or nervous system. The BioDigital Human is a little more customizable, having a menu with checkboxes for every single element of the body to choose whether you want it displayed or not. There is also a whole lot information regarding all these elements in The BioDigital Human, and in addition a tab with illustrations and animations for different conditions and how they affect the involved organs.

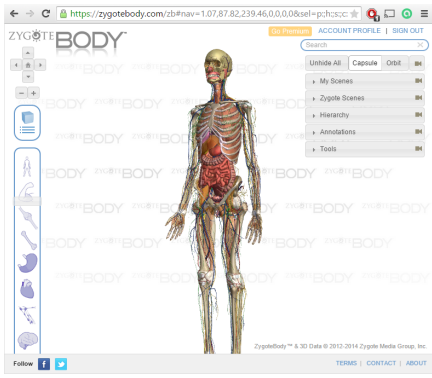


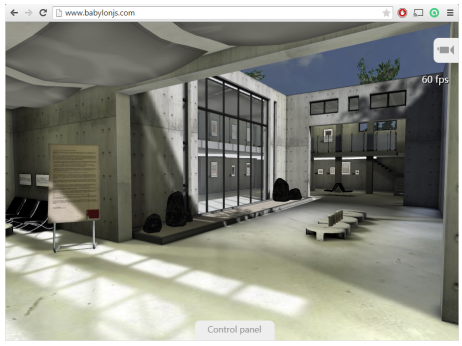
Figure 2.4: Zygote Body

Figure 2.5: The BioDigital Human

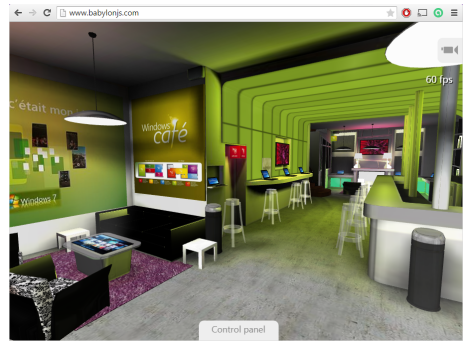
Both applications uses a arc rotate camera, a camera where there is a focus point the camera can rotate around. The focus point can be moved either up or down the body in Zygote Body, while you can move it more freely in The BioDigital Human. Zoom and highlighting of body parts by click interactions are supported in both applications.

2.3.1.2 Babylon.js example projects

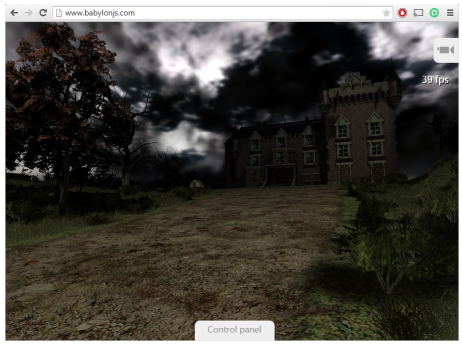
The higher-level WebGL framework Babylon.js, which will be discussed later, has a bunch of testable demos available on their website. Most of them are pretty similar feature-wise. You are spawn into a small, but detailed 3D scene, and can move around the scene in a first-person perspective. These demos have no support for any form of interaction with the environment, but a couple of the scenes presents animations of some objects. Texture-wise the scenes are very detailed.



(a) Espilit



(b) Windows Cafe



(c) Mansion

(d) Hill Valley

Figure 2.6: A selection of Babylon.js demos

2.3.2 Related work of integrating BIM and WebGL

One thing this literature study has uncovered is that there is not much published work that deals with use of BIM models for other purposes than building planning. Kumar et al. (2011) developed a game application for evaluating healthcare facility designs, and Lin et al. (2011) created a game for construction safety training. Both these games uses BIM models, however they are implemented as stand alone applications, i.e. they are neither browser based nor using a 3D Rendering API like WebGL. Shen et al. (2012) found that using BIM models to create interactive 3D computer games is still a relatively new approach, especially for web browser based solutions.

2.4 WebGL frameworks

Drawing primitives directly to the graphics card is a difficult task, and using WebGL could therefore prove to be a steep learning curve for developers new to the technology (Curran, 2012). Fortunately there exists intermediaries to make the development process easier. Ever since WebGL's very beginning high-level frameworks have been developed for the API. The alternatives range from one-man basement projects to commercial products from renowned companies. This section will give a brief introduction of some of the options out there. Only frameworks that support 3D will be taken into account. Some of the most popular WebGL game creators only offers a 2D engine actually, mentioning frameworks like Construct 2, pixi.js and Phaser.

2.4.1 Three.js

Three.js was first released to GitHub in April 2010, and ready to use as soon WebGL was introduced in 2011. It is a cross-browser JavaScript library to create animated 3D graphics. It runs in all browsers supporting WebGL. The mission of Three.js is to provide an abstraction layer to hide the detailed WebGL setup, and make it possible to create a 3D world without much knowledge of WebGL. The framework has a very broad approach, not focusing on a single niche. It is more a general purpose web animation tool. It is not a typical game engine.

Pros:

- A lot of features
- Good documentation and community support
- Widely used
- Free and open source under the MIT License
- Exporter for Blender

Cons:

- Still in alpha version and is changing frequently
- ”Jack of all trades, master of none” compared to some of the other frameworks
- No physics engine, nor collision detection

2.4.2 Babylon.js

One of the newer frameworks is called Babylon.js. It was launched in the summer of 2013, and is built by developers at Microsoft. This framework has a more targeted approach than Three.js, and focuses on game development with collision detection and anti-aliasing (Hewitson, 2013). Babylon.js is also open source. The framework supports a complete scene graph with lights, cameras, materials and meshes. It features engines for collisions, physics, animation and optimizations.

Pros:

- Free and open-source
- Good community support
- A lot of features
- Backed by a major company
- Exporter for Blender

Cons:

- Imported scenes or meshes needs to be converted to a Babylon file format

2.4.3 Goo Engine

The Goo Engine is a powerful WebGL game engine, and is coupled with Goo Create, a browser-based editor. The engine is build using the Entity-Component-System, a software design pattern that favours composition over inheritance. There are tools in the development environment, Goo Create, that offers easy “drag and drop” features for adding interactivity, importing 3D models etc. The engine is compatible with most of the modern mobile devices.

Pros:

- A well customized development environment
- Physics and collision engines

Cons:

- Not open source
- Only free to use in a limited period
- Locked to the Goo Create cloud platform

2.4.4 PlayCanvas

PlayCanvas is a 3D game engine written in JavaScript, and runs in all WebGL supported browsers. It is also a cloud-hosted creation platform, which allows for real-time editing of multiple developers simultaneously. PlayCanvas offers a powerful and at the same time simple API for scripting the game behaviour. Since June 2014 PlayCanvas has been open-source, and exists in both free and non-free versions.

Pros:

- Open source under the MIT Licence
- Physics and collisions engines
- Support for real-time editing

Cons:

- Development and publishing strictly tied to PlayCanvas' cloud platform
- No private projects in the free version
- Not as active community as some of the others

2.4.5 Turbulenz

Turbulenz is a polished 3D game engine in JavaScript that executes directly in the browser. The engine has support for rigid bodies, collision primitives and constraints. A large collection of built in animation controllers are offered as well.

Pros:

- Free and open source under the MIT license
- Easy to combine Turbulenz modules with external technology

Cons:

- Not the most active community

2.4.6 C3DL

Canvas 3D Library, known as C3DL, is a JavaScript library that was designed for the purpose of simplifying the use of WebGL. It provides a set of math, scene, and 3D object classes to make the canvas more accessible without dealing with 3D mathematics at a low level.

Pros:

- Free under the GNU GPL Licence
- Support collision detection

Cons:

- An old project, and not in development any longer
- No support at this time, and not a very active community
- Documentation removed
- No designated physics engine

2.4.7 CopperLicht

CopperLight is a WebGL library and JavaScript 3D engine developed by Ambiera. The main feature that differs CopperLight from the earlier mentioned frameworks is its binary compilation. The 3D meshes compiles to small, binary files to reduce the user's bandwidth usage. CopperLight also supports over 20 different 3D file formats. For CopperLight you can create scenes with a 3D editing tool called CopperCube, which was the first full 3D editor for creating interactive WebGL 3D scenes or games. The CopperLight library can be used outside CopperCube but is built in regard of this 3D editor.

Pros:

- Free and open source
- A solid 3D editing tool in CopperCube
- Physics engine and built-in collision detection
- Character animation support
- Good documentation

Cons:

- The CopperCube is not free
- Not the most active community

2.4.8 SceneJS

SceneJS is an extensible WebGL-based engine for high-detail 3D visualization using JavaScript. It is created by Lindsay Kay, who is also the 3D engine architect on The BioDigital Human team. This engine provides a JSON-based scene graph API on WebGL. SceneJS specialization is fast rendering of a vast number of different articulated objects. It is focused on CAD (Computer-aided design), like medical and engineering visualization, not on features as shadows, reflections, collisions etc. which are typical in game engines (Xeolabs, 2011)

Pros:

- Free and open source
- Used in the BioDigital Human project
- Fast rendering of complex objects.

Cons:

- Not designed as a game-engine
- No physics or collision engines
- Documentation not completed (as of January 2015)

2.4.9 GLGE

GLGE is a JavaScript library with the aim of masking the involvement with WebGL for the developer. It is a general purpose framework and support features as keyframe animations, reflections/refractions and parallax mapping.

Pros:

- Free and open source
- Some neat features
- Good documentation

Cons:

- Little community support at this stage
- General purpose framework
- No physics or collision engines

2.4.10 Unity

The Unity game engine is one of the most used in the marked today. It offers a vast number of platforms, including desktops, consoles, mobile devices, and from April 2014 support

for WebGL. The technology includes a game engine and an integrated development environment (IDE).

Pros:

- Easy to learn and use
- Large community support
- Physics engine and collision detection

Cons:

- Unity Pro is expensive
- Free version lack some neat features
- Not open-source
- Documentation lacking or out of date for some features

2.5 Framework comparison

Table 2.4 gives a comparison of the examined frameworks, given criteria that has a significance for the choice of framework in this project. The data basis for the table is gathered through the frameworks own websites and documentation, and from related sites like forums acting as community support for a specific framework. The assessment criteria and classifications of them are as following:

Open Source (Open): Whether the project has an open source licence or not. Classification:

- Yes/No

Free: Can the framework be used free of cost? Classification:

- Yes/No
- Partial - There exists both free and paid versions.

Community (Com): How good community support is there. This includes forums, blogs from developers etc. Classification:

- Active - Has a forum where questions are handled within a day, and there is activity weekly.
- Moderate - A forum with monthly activity, and questions are answered within days.
- Inactive - There are no or very occasional activity on forums, blogs etc.

Documentation (Doc): How well documented is the framework. Includes API documentation, tutorials etc. Classification:

- Rich - All classes are well documented and there exists tutorials or other examples of use.
- Fair - The vast majority of the classes are documented, but the documentation could be more detailed. There are tutorials of some degree.
- Lack - Documentation of classes are either lacking or minimal. Not many tutorials.

Physics (Phy): The framework has a designated physics engine. Classification:

- Yes/No

Collision (Col): The framework has a designated collision engine or collision handling integrated in the physics engine. Classification:

- Yes/No

Import (Imp): Support for importing external 3D models. Classification:

- Yes/No

Framework:	Open	Free	Com	Doc	Phy	Col	Imp
Three.js	Yes	Yes	Active	Rich	No	No	Yes
Babylon.js	Yes	Yes	Active	Fair	Yes	Yes	Yes
Goo Engine	No	Partial	Moderate	Rich	Yes	Yes	Yes
PlayCanvas	Yes	Partial	Moderate	Rich	Yes	Yes	Yes
Turbulenz	Yes	Yes	Moderate	Fair	Yes	Yes	Yes
C3DL	Yes	Yes	Inactive	Lack	No	Yes	Yes
CopperLicht	Yes	Partial	Moderate	Rich	Yes	Yes	Yes
SceneJS	Yes	Yes	Inactive	Lack	No	No	Yes
GLGE	Yes	Yes	Inactive	Rich	No	No	Yes
Unity	No	Partial	Active	Fair	Yes	Yes	Yes

Table 2.4: Comparison of high-level WebGL frameworks

Of the examined frameworks only Goo Engine and Unity are not open source. All frameworks can be used free of charge, but four of them (Goo Engine, PlayCanvas, CopperLicht and Unity) require a paid version to adopt all functionality/benefits. For Community support Three.js, Babylon.js and Unity stand out as the most active, while C3DL, SceneJS and GLGE are on the opposite side of the scale. Most of the framework are well documented, except C3DL and SceneJS. C3DL is more or less a dead project, while SceneJS has a large selection of tutorials and examples, but lacks a general API documentation.

When it comes to physics and collision engines it gets clear which of the frameworks are typical game engines and which are more of a general purpose animation or visualization tool. Three.js, SceneJS and GLGE do not have support for this, while C3DL only supports collision detection. All frameworks allows for import of external 3D models, but they vary in the number of accepted file formats. None of them can import IFC files directly.

From the criteria set for the comparison Babylon.js seems like the most adequate framework to apply for this project. The best alternatives to Babylon.js are PlayCanvas, Turbulenz and CopperLicht, while the frameworks without physics engine and/or support for collision detection are ruled out.

Chapter 3

Development

This chapter deals with the developing process of the WebGL prototype. A more accurate description of the prototype will be given. Elaboration of the development tools used will follow. Further the modelling and implementation process will be disclosed.

3.1 Prototype description

The prototype will present a scene with a 3D model of the 1930-building, an administrative building from St. Olav's Hospital, where the 3D model will originate from the actual BIM models of the building. This ensures that building measures, room divisions and interior of the virtual structure corresponds with the real building.

The idea was to create a navigation mode similar to typical first-person shooter games. What this implies is that the camera represents your eyes, and you alter your viewpoint by moving the mouse. Your leg movement is controlled with the keyboard. By adding collision handling and gravity to the scene this will create an illusion of a human walking around in the virtual world.

What is described above creates the foundation for this prototype, namely a way to walk around in an authentic building, in a virtual world, and be able to do it from your browser. The plan from here on was ambiguous at times, but the goal was to support some kind of interactions with the environment. Since St. Olav's has a large collection of art it was in St. Olav Eiendom interests that there was a possible to present the art in the virtual world, and display info of each item through interaction. Another feature that came to the table was to play videos of some sort directly on objects in the 3D world.

3.2 Development tools and environment

The following subsection will elaborate on the tools and development environments used in the process. All text editors, modelling tools, frameworks etc. that have been applied will be given a brief introduction, and a justification for the choice.

3.2.1 Sublime Text

The HTML, CSS and JavaScript source code is written in Sublime Text, a text editor written in C++ and Python. Sublime Text is a cross-platform editor, compatible with Linux, OS X and Windows, and natively supports over 40 programming languages. Additional languages might as well be installed via plug-ins. A hotkey focused inteface, rich functionality and customizability are characteristics that applies to Sublime Text.

Sublime Text is widely used text editor, especially for web development. A range of polls and discussions on the web also shows it is one of the most popular among software developers (Henry, 2014). The choice of text editor is not as essential for the development as the rest of the technology listed below. The choice was taken on the basis of personal preference and experience with the text editor.

3.2.2 Babylon.js

The high-level WebGL framework chosen for this project is already superficially described in section 2.4, and this subsection will go a little deeper into the framework.

Babylon.js has a rich library of supported features, and the most interesting for this project were:

- **Engines:** Collision, physics, animation
- **Optimization:** Selection octrees, incremental loading, binary compressed format, frustum clipping, hardware scaling
- **Cameras:** Free camera, touch camera, oculus rift camera
- **Light:** Point, directional, spot, hemispheric
- **Texture:** Dynamic texturing, video texture
- **Meshes:** Dynamic meshes, skybox
- **Fullscreen mode**

Section 2.4 shows there is a bunch of adequate higher-level WebGL libraries out there, and most of them are free and offer approximately the same features. The main factors for choosing framework in this project were the community support and documentation, a physics engine with collision detection and flexible import of external 3D models. Flexible in this context means support for a range of file formats, and the easiness of working with

the imported meshes. Babylon.js fulfils all these criteria. The community is one of the most active of the frameworks examined, and the documentation is, if not excellent, at least complementary. The only file format that is supported for imported 3D models is it own Babylon file format. It is a reason for this however as the framework aims to utilize Blender as intermediary when importing external models. More on this in the next subsection.

Another reason for going with Babylon.js is due to the competence and resources of the creators. The Babylon team consists of Microsoft employees, and although it is a stand-alone project Microsoft backs the initiative. The example work on Babylon.js' homepage displays a bunch of demo applications with a similar purpose as this project which also increased the desire to work with this framework.

3.2.3 Blender

Blender is a free and open-source 3D computer graphics software with a great amount of functionality. It can be used for a number purposes including 3D modelling, video editing, creating visual effects, animation films, 3D applications and video games. It even has an integrated game engine. Blender supports 3D model import and export for 14 different file formats by default, and several others can be added. The user interface mostly relies on hotkeys for the modelling process, and is a software that takes time to master due to its rich set of functionality and possibilities.

The choice of using Blender in this project is based on the use of BIM models and the Babylon.js framework. To import meshes in Babylon they need to be represented in the Babylon file format, as mentioned in the previous subsection. Babylon.js is dependant on a 3D modelling software that can export 3D models to this format. Blender, in addition to Cheetah3d and 3ds max, can export to the Babylon file format by installing an export plug-in to the software. An import plug-in that support the IFC file format can also be added to Blender, which make importing of BIM models possible. Blender therefore worked as an intermediary point for converting the IFC files, holding the BIM models, to the Babylon file format in this project. The software was also used for some extra 3D modelling as for example adding ground outside and other details to enrich the 3D scene.

3.2.4 Server

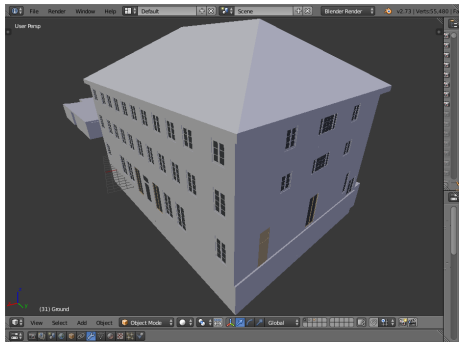
The project is hosted at NTNU's Apache servers, and can be accesses from the domain folk.ntnu.no/hestman

3.3 Modelling

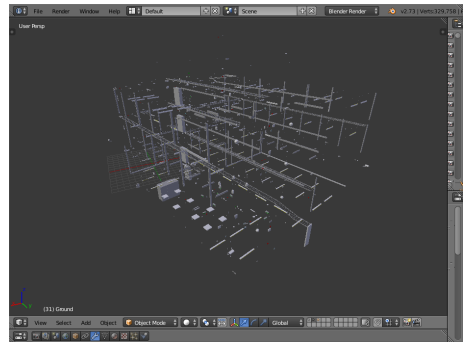
The basis for the 3D scene are the BIM models of the 1930-building at St. Olav's Hospital. The BIM models applied are the following:

- **ARK:** 1262 objects consisting of walls, windows, doors, ceiling, roof, beams, benches, lockers, shelves, kitchen benches and appliances.
- **RIE:** 1023 objects consisting of ceiling luminaires, lamps, downlights, emergency door openers, sockets, card readers, elbow switches, emergency exit signs, fire alarms, fire detectors, thermostats, surveillance cameras, cable trays and more.

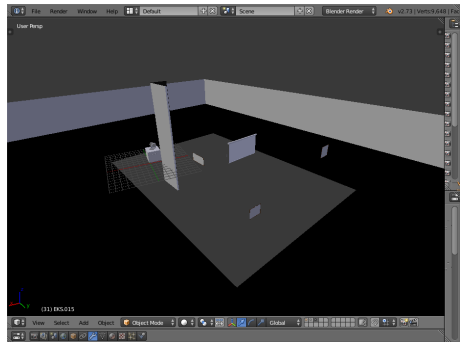
The Blender scene created consists of three layers. Layer 1 represents the ARK model and layer 2 the RIE model. Layer 3 consists of self-modelled meshes to complete the scene. There are multiple reason for using this layered approach in the development. First of all there was a wish from St. Olav Eiendom not to modify the BIM models themselves, but since this application will be presented in a game fashion, i.e. create an illusion of a person that can walk around, and not just a building viewer, there needs to be something that restricts and encloses the world. Layer 3 will handle these parts.



(a) Layer 1



(b) Layer 2



(c) Layer 3

Figure 3.1: Layers of the Blender scene

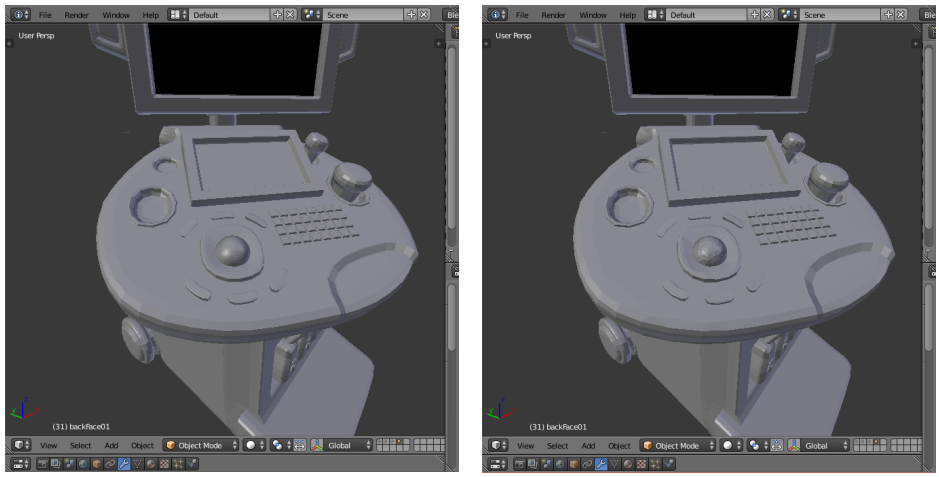
The idea of separating the BIM models into separate layers is because of two factors. The BIM models are already separated into different IFC files, and modified and improved versions of these can be released. By also keeping these separated in the Blender scene an updated ARK model for example can be imported and then only overwrite the existing architectural model without having to tamper with the electrical model. If other BIM

models were to be added to the scene, for instance an interior design model (IARK), these would also be placed in separate layers. The other factor for dividing the models are related to performance.

Rendering 3D graphics in web browsers can be a costly affair, especially if the machine is rather weak and the 3D scene is complex, which might result in high latency. By separating the BIM models we can more easily customize the model complexity by not necessarily import all the models. Since the users' machines might have completely different processing power making the model complexity customizable will help offering a working product even for the weaker machines and browsers by compromising with the 3D scene richness.

3.3.1 Supplementary modelling

Apart from the existing BIM models some objects had to be modelled to complete and enrich the scene. The supplementary modelling is done in layer 3 of the Blender scene. It consists of some simple structures as an outside ground, a brick wall that encloses the site and thus limits the area one can navigate. Some walls that fills the defects in BIM models are also present in this layer. In addition objects that were wanted to enrich the scene visually and interactively apart from the ones present in the BIM models are placed in this layer. It includes different screens, paintings and an ultrasound machine. The model of the ultrasound machine was downloaded from a free 3D model library and imported to Blender. The problem with this model was the detail level, consisting of more than 40 000 vertices and 80 000 faces. The edge-collapse algorithm were applied for mesh simplification where the number of faces were reduced with 80 %. The visual transformation can be seen in Figure 3.2.



(a) Before (41 346 vertices, 82 396 faces) **(b)** After (8 387 vertices, 16 478 faces)

Figure 3.2: Edge collapsing applied to a 3D model

3.4 Implementation

The following section will deal with the process of implementing the 3D ”game”. The complete source code can be found in Appendix C, while this section will in a step-by-step fashion present the different aspects that have been handled to end up with the working prototype.

Setting up a Babylon scene is fairly simple. The first step is to create a HTML5 <canvas> element and define the size of it with CSS, as illustrated in Listing 3.1. The rest is done in JavaScript. A BABYLON.Engine needs to be connected to the canvas, and a BABYLON.Scene attached to the engine, shown in Listing 3.2. From here on the 3D world can be built by adding objects, cameras, lights etc. to the BABYLON.Scene object.

```
<html>
  ...
  <style>
  #babylonCanvas {
    width: 100%;
    height: 100%;
    touch-action: none;
  }
  </style>
  ...
  <body>
    <canvas id="babylonCanvas"></canvas>
    ...
  </body>
<html>
```

Listing 3.1: Setting up a canvas in HTML

```
  var canvas = document.getElementById('babylonCanvas');
  var engine = new BABYLON.Engine(canvas, true);
  var scene = new BABYLON.Scene(engine);
```

Listing 3.2: Attaching a Babylon scene to the canvas

3.4.1 3D model import

Babylon has a straight forward way for importing external 3D models, and the library defines three different methods for this purpose. You can either load a complete scene which creates a Babylon scene, import a defined set of meshes to an existing Babylon scene, or append a complete scene to an existing Babylon scene. A code example and description is given in Listing 3.3 and Table 3.1.

```
//Loading a complete scene
BABYLON.SceneLoader.Load(rootUrl, sceneFilename, engine, onsuccess,
    progressCallBack, onerror);

//Importing a defined set of meshes to an existing Babylon scene
```

```
5 BABYLON.SceneLoader.ImportMesh(meshesNames, rootUrl, sceneFilename, scene,
      onsuccess, progressCallBack, onerror);
6
7 //Appending a scene to an existin Babylon scene
8 BABYLON.SceneLoader.Append(rootUrl, sceneFilename, scene, onsuccess,
    progressCallBack, onerror);
```

Listing 3.3: Import methods in Babylon.js

Parameters	Description
rootUrl	A string defining the root url of the Babylon file
sceneFilename	A string defining the name of the Babylon file
meshesNames	A list of names of meshes from the Babylon file to be imported. The empty string, "", imports all meshes from the file
engine	The BABYLON.Engine instance used to create the scene
scene	The Babylon.Scene instance the meshes will be imported to or scene be appended to
onsuccess	(Optional) A callback function triggered after a successful load /import
progressCallBack	(Optional) A function returning the import progress
onerror	(Optional) A callback function triggered if an error occurs during loading

Table 3.1: Paramenter description for the Babylon.SceneLoader

In this prototype two Babylon files are loaded separately. The first file made up of layer 1 og layer 3 in the Blender scene, while the other file consists of layer 2. As discussed in the previous section this is done to be able to easily opt out the RIE model if the complexity of the scene impairs the performance excessively.

3.4.2 Skybox

A 3D world needs to be delimited in some way, i.e. create a final visual background. The most common approach in video games is to create a *skybox*. A skybox is simply a cube that encloses the 3D world. The inside faces of this cube consists of images that overlaps at the edges, creating an illusion of distant 3D surroundings. Figure 3.3 illustrates a typical texture to be wrapped around the cube. Creating 3D models of mountains, buildings etc. and place them far outside the navigable part of the 3D world to create a background would just have been a waste of computations.

The only JavaScript-generated geometry in this project is actually the skybox, as seen in Listing 3.4. Notice line 9 where the property infiniteDistance is set to true, which makes the skybox move according to the movements of the camera. This amplifies the impression of the skybox being far away. With a static skybox it would be easier for the user to reveal that the background is actually a cube not that far away as it would approached the camera, similar to the other objects of the scene, when moving.

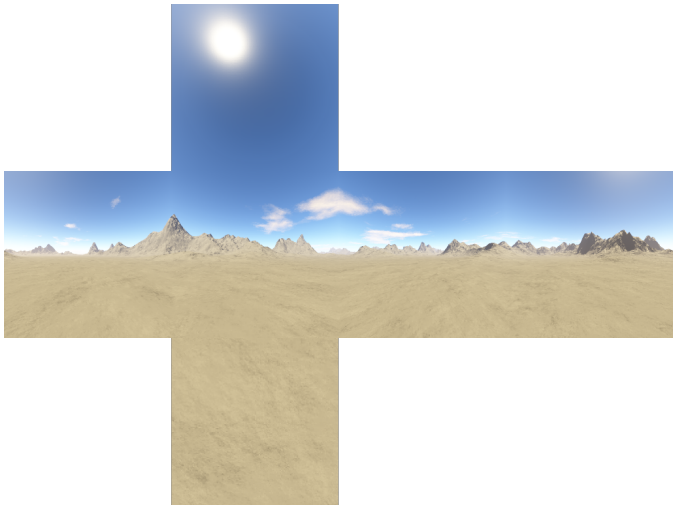


Figure 3.3: Skybox textures

```
1 var skybox = BABYLON.Mesh.CreateBox("skyBox", 150.0, scene);
2 var skyboxMaterial = new BABYLON.StandardMaterial("skyBox", scene);
3 skyboxMaterial.backFaceCulling = false;
4 skybox.material = skyboxMaterial;
5 skyboxMaterial.diffuseColor = new BABYLON.Color3(0, 0, 0);
6 skyboxMaterial.specularColor = new BABYLON.Color3(0, 0, 0);
7 skyboxMaterial.reflectionTexture = new BABYLON.CubeTexture("cubemap/skybox
      ", scene);
8 skyboxMaterial.reflectionTexture.coordinatesMode = BABYLON.Texture.
      SKYBOX_MODE;
9 skybox.infiniteDistance = true;
```

Listing 3.4: Creating a skybox in Babylon.js

3.4.3 Cameras and lightning

To be able to visualise the 3D objects in the scene a camera and a light source is necessary. The camera represents viewpoint you see the scene from, and the light settings will determine the diffuse and specular colour received by each pixel and thus decide what will be possible to see. In this scene a hemispheric light is used, which is a simple way to simulate a sun as a light source. This light does not actually have a concrete position, only a direction from where the light is coming from is defined. This means that the pixel colours are defined by the angle of the mesh surfaces according to the direction of the light. All rooms of the building are therefore equally illuminated as the light is not coming from an actual sun where walls blocks for the sunlight. The benefit with this approach is that you do not have to set up extra light sources all around the building. The downside is the indoor lighting environment becomes less realistic.

```
1 var light = new BABYLON.HemisphericLight("Hemi", new BABYLON.Vector3(0, 1,
      0), scene);
2 light.diffuse = new BABYLON.Color3(1, 1, 1);
3 light.specular = new BABYLON.Color3(1, 1, 1);
4 light.groundColor = new BABYLON.Color3(0, 0, 0);
```

Listing 3.5: Setting up hemispheric light in Babylon.js

Listing 3.5 shows how the hemispheric light is created with Babylon.js. A vector describes the direction of the light, which in this case comes straight from above. In addition the light intensity for diffuse, specular and ground colour. Diffuse represents faces pointing towards the light, while ground represents faces pointing downward from the light. The camera setup is further described in subsection 3.4.5.

3.4.4 Collisions and optimizations

Reducing the number of unnecessarily draw calls and other calculations is essential in a 3D scene because of the usually vast dataset. This subsection will describe how this is handled in the prototype. In addition 3.5.3 addresses some optimizations techniques that were more or less unsuccessfully.

3.4.4.1 Collision handling

Collision handling can easily become a bottleneck, and it is therefore vital to reduce the number of meshes to check collisions for. When we look at the two BIM models present in this application one thing is very apparent. The architectural model consists of walls, doors, floors etc., all of them objects that the moving camera needs to check collisions for. The RIE model covers object that mostly are attached to walls and ceilings in the ARK model, like sockets, door openers, lights etc. If there already is collision handling for the walls the electrical objects are attached to making collision checks for these objects will just aggravate the performance without giving you anything in return. In practice you will not notice that you floats through a door opener if you collide with the wall it is attached to. By separating the ARK and RIE model we can assign collision checks to the meshes in the ARK import, while we neglects it in the RIE import.

To be more accurate, not even all of the meshes in the ARK model are assigned collision checks, actually the minority are. The ARK model consists of over thousand objects, where the larger portion are objects that don't need collision checks as for example smaller windows, ceiling beams, roofing sheets etc. Luckily the naming conventions of the BIM model makes it possible to categorize the type of object by substrings of it's name. Listing 3.6 shows how collision checks are assigned in this prototype.

```
1 //Importing the ARK model and assigning collision detection
2 BABYLON.SceneLoader.ImportMesh("", "models/", "ARK.babylon", scene,
      function (newMeshes){
3   //...
```

```
4    var mName;
5    for(var i=0; i<newMeshes.length; i++){
6      mName=newMeshes[i].name;
7      if((mName.substring(0,4)==="EKS." || mName.substring(0,3)==="Dek" ||
           mName.substring(0,2)==="IV") || mName.substring(0,5)==="Trapp" ||
           mName==="Elevator wall" || mName==="WallBarrier" || mName==="
           Ground" || mName==="ultrasoundBoundingBox" || mName==="Ekran" ||
           mName==="Tak-.001" || mName.substring(0,5)==="DI040" || mName==="V
           -.006" || mName==="V-.007" || mName==="V-.013"){
8        newMeshes[i].checkCollisions=true;
9      }
10     else if(mName.substring(0,2)==="DB" || mName.substring(0,4)==="EKS_"
           || mName.substring(0,2)==="DI" || mName==="DU001"){
11       newMeshes[i].checkCollisions=true;
12       activateDoorActions(newMeshes[i]);
13     }
14   }
15   //...
16 });
17
18 //Importing the RIE model
19 BABYLON.SceneLoader.ImportMesh("", "models/", "RIE.babylon", scene,
       function(newMeshes){
20   //...
21 });
```

Listing 3.6: Assigning collision checks for the ARK model

Doing collisions handling for the RIE model would have aggravated the performance greatly. Not just because the number of meshes doing collision checks for would have been multiplied greatly, but also because of the complexity of the RIE meshes compared to the ARK meshes. The objects in the ARK model is mostly walls, floors, doors and roofs. These can be characterized as very simple 3D objects. A wall can in the simplest case be represented by as little as four vertices, and two triangular faces. The RIE model consists of much more detailed objects, for example lamps with pebbled surfaces represented by thousands of faces.

There is a work around for these cases, which is to create an invisible bounding box for the object and check collisions with this box instead. A bounding box is the smallest box possible that encloses the whole object. This primitive mesh consists of just 8 vertices and 12 faces, two triangles for each of the 6 sides of the box. When doing collision checks for the bounding box we can limit the calculations to the maximum and minimum x, y and z coordinates of the box, and not the enormous amount of coordinates representing the surface of the actual object. What is even better is to not do any collision checks at all if it is not necessary for the game's realism, which is the case for the objects in the RIE model.

A good example of how use of bounding boxes for collision detection increases performance can however be found in the prototype. The ultrasound machine is placed in a room on the second floor. This 3D object originally consisted of over 80 000 faces. Doing collision handling for this complex object is very costly, as illustrated in Figure 3.4. If an bounding box is created, and collision checks are done for this box instead the result is completely different, as figure Figure 3.5 shows. When approaching the ultrasound

machine in Figure 3.4 the frame rate dropped as low as 13 fps. In Figure 3.5, where the bounding box is used for collision checks instead, the frame rate is unaffected, and stays at 60 fps.

Figure 3.4: Collision check with a complex object

Figure 3.5: Collision check with a bounding box

3.4.4.2 Culling

In a 3D scene not all surfaces needs to be drawn at all time. Generally speaking only a small minority of surfaces in a scene is visible through the virtual camera. To reduce draw calls culling is used to eliminate non-visible surfaces. There are multiple types of culling:

- *Viewing frustum culling* is the elimination of objects outside the viewing frustum, i.e. objects placed outside the camera's field of view.
- *Backface culling* is elimination of the surfaces of an object that does not face the camera. For example removing the backside of a cube.
- *Occlusion culling* deals with eliminating objects that are entirely hidden behind other opaque objects seen from the camera's viewpoint

Viewing frustum culling and backface culling are supported in Babylon.js, though occlusion culling is not. This will result in a variable performance depending on the camera's viewpoint. For example looking at the building from the outside will require a whole lot more draw calls than looking out of the building from the inside. This is because a lot more meshes will be within the viewing frustum, even though they are hidden and not visible from the user's perspective.

For viewing frustum culling in the prototype *octrees* are used to optimize mesh selection for rendering. An octree is a tree data structure and is the three-dimensional analogue of a quadtree. Each internal node of the tree has exactly eight children. When representing a 3D scene with an octree, the world is divided into 8 blocks, each of these block may again be subdivided into 8 new blocks, and so on. When traversing the three and a node, representing a block, is considered to be invisible, which means outside the viewing frustum, then all of its child nodes are invisible too and does not need to be processed. This way all objects related to leaf nodes of a node found invisible can be eliminated at once.

Using octrees for this purpose is the recommended way of doing it in Babylon.js, and it only requires one call to the Babylon library to create or update the selection octree for the 3D scene.

```
1 var octree = scene.createOrUpdateSelectionOctree(capacity, maxDepth);
```

Listing 3.7: Create or update an octree in Babylon

3.4.5 Navigation

The navigation of the game is as mentioned based on the first-person shooter principle, that is altering the viewpoint with the mouse, and physically moving the camera with keyboard buttons. Apart from that artificial gravity is needed to prevent the user from moving through mid-air. By defining a *FreeCamera* in Babylon the altering of viewpoint by mouse interaction will be set up by default. For mobile devices a *TouchCamera* is being used instead. With this camera both viewpoint altering and movement is handled by touch events. In addition a moving speed for the camera is set to mimic walking speed.

Another vital thing is to make to the camera feel like it represents the view of a human being. This is done by creating an ellipsoid around the camera. Since Babylon uses a left-handed coordinate system the y-axis value determines the viewpoint's height above ground, while the values for the x-axis and z-axis defines the boundary ahead, rearward and to the sides. For the ellipsoid to have any function collision handling for the camera has to be activated. or else the camera will be able to move through objects. A gravitational force must be defined and applied to the camera to limit the cameras movement to solid ground. Listing 3.8 shows the whole process, using the Babylon library.

```
1  //setting up camera
2  var startingPos =  new BABYLON.Vector3(-2, 2, -35);
3  if(/Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.test(
       navigator.userAgent) ) {
4      camera = new BABYLON.TouchCamera("TouchCamera", startingPos, scene);
5  }
6  else{
7    camera = new BABYLON.FreeCamera("FreeCamera", startingPos, scene);
8  }
9  camera.setTarget(new BABYLON.Vector3(0, 0, 0));
10 scene.activeCamera.attachControl(container);
11
12 //Defining camera properties
13 camera.speed = 0.2;
14 camera.ellipsoid.y = 1.1;
15 camera.ellipsoid.z = 0.55;
16 camera.ellipsoid.x = 0.55;
17 camera.checkCollisions = true;
18
19 //Defining and applying gravitational force
20 scene.gravity = new BABYLON.Vector3(0, -0.01, 0);
21 camera.applyGravity = true;
```

Listing 3.8: Creating a first-person camera in Babylon.js

By default the FreeCamera in Babylon can be moved by the arrow keys. In addition W, A, S, D keys have been assigned to create a more user friendly option, as these keys are better

positioned for the left hand. The last modifications done regarding navigation is assigning some special keys for running and jumping. This is done by altering the camera moving speed if running button is activated, and creating a jumping animation when the button for jumping is pushed.

What distinguishes browser applications from stand alone application is the necessity of the mouse cursor to leave the browser application window if desired. In the typical first-person shooter the mouse cursor is locked, and Mouse movements alters the target of the viewpoint instead of moving the pointer across the screen. Even though this is possible to implement for browser applications there are some issues regarding this approach. First of all not all browsers allow that. Of the desktop browsers examined in this thesis only Google Chrome and Mozilla Firefox permits it (McCuthan, 2012). Secondly, it might not be in the user's interest usability-wise to loose the ability to leave the browser window with the pointer.

In this prototype the cursor is visible and altering viewpoint is done by mouse click and drag, like in Google Street View. In addition a fullscreen mode can be enabled if the user wants to alter viewpoint just by moving the mouse, like in first person shooters. However the nature of the fullscreen mode depends on the browser. Not all support pointer lock, thus the user still have to click and drag to alter viewpoint if he/she uses one of these browsers.

3.4.6 Interaction

A game with no sort of interaction, besides moving around, will quickly become uninteresting. Since the mouse cursor is visible, as discussed in previous subsection, the concept of click interaction is exploited. The mouse cursor will turn into a hand when moving over objects in the scene that can be interacted with by clicking. In the Babylon framework this is done by defining actions to be executed when triggers are fired. The triggers can be for example clicks on a mesh, moving pointer over a mesh or meshes who intersects. Actions can be changing of mesh properties, starting or stopping animations etc.

```
1 function activateDoorActions(mesh){
2   mesh.actionManager = new BABYLON.ActionManager(scene);
3   var child = [new BABYLON.SetValueAction(BABYLON.ActionManager.
        NothingTrigger, mesh, "checkCollisions", 0), new BABYLON.
        SetValueAction(BABYLON.ActionManager.OnPickTrigger, mesh, "
        visibility", 0)];
4   var child2 = [new BABYLON.SetValueAction(BABYLON.ActionManager.
        NothingTrigger, mesh, "checkCollisions", 1), new BABYLON.
        SetValueAction(BABYLON.ActionManager.OnPickTrigger, mesh, "
        visibility", 1)];
5   var action = new BABYLON.CombineAction(BABYLON.ActionManager.
        OnPickTrigger, child);
6   var action2 = new BABYLON.CombineAction(BABYLON.ActionManager.
        OnPickTrigger, child2);
7   mesh.actionManager.registerAction(action).then(action2);
8 }
```

Listing 3.9: Function assigning combined actions for a mesh

Listing 3.9 is a function from the source code used to assign actions to the door meshes in the scene. The function defines two combined action, one that sets collision checks and visibility to false, and one that sets these properties to true. These actions are assigned to the mesh in line 10, and will be called every other time when the mesh is clicked. The reason for these actions are to provide a way of opening and closing the doors in the building. The original plan was to make door opening animations, but due to the nature of the door meshes this idea was discarded. First of all double doors were modelled as a single mesh in the BIM models, so rotating the whole mesh would seem strange. Secondly there is no way to decide which way the doors should be opened in general, and this could make them become obstacles if not specifying a direction for every single door independently.

Other objects where click interactions are present includes paintings on the wall, screens and a ultrasound machine. Actions assigned to these objects use a custom code snippet defined in a callback function, which is run when a trigger is sparked under a condition that is true. The function ”activateVideoTexture()” in Listing C.2, in Appendix C, is an example of this. The outcome of these objects’ actions will be described in the next chapter.

3.5 Challenges

3.5.1 Stairs and elevation inequities

A typical challenge when building games with gravitational force and collision is to handle elevation inequities. In a 3D world there can be a lot of bumps etc. that a real person would have no trouble passing, but becomes a problem for the virtual person because of collision detection. Typical examples are stairs and door sills, which also exists in this prototype. There are two main approaches to this issue.

The first one attacks the bumps. What is meant by this is trying to physically remove the bumps, but preserve them visually at the same time. This is done by creating extra invisible meshes to smoothen the bumps. A good example of this is to place an invisible ramp on top of the stairs. This approach was first commenced in this project, but then discarded due to the large time consumption the modelling process entailed compared to the other approach.

The method that was used instead was to tamper with the gravitational force. By setting it artificially low the moving camera where able to walk stairs and pass other small hurdles without getting stuck. This method is very quick and easily implemented, but can not be exploited in every scenario. The drawback is situations where the user is able to jump off a hill etc. In these scenarios the unrealistic gravitational force will be revealed. However in this application there is to be no possibilities to end up in free fall, and therefore virtually impossible to uncover the nature of the gravity.

3.5.2 UV mapping

The video textures used in the prototype are applied to meshes at run-time, as they only will appear when requested/activated by the user. For meshes created with JavaScript, using the Babylon library, applying video textures is straight forward. It is just to apply a material containing a video texture. For imported meshes from a Blender scene one more step has to be conducted, namely *UV mapping*. UV mapping is the process of defining how a 2D image is to be mapped onto polygons. The UV mapping was done automatically for JavaScript generated meshes, this was not the case for the imported meshes from Blender.

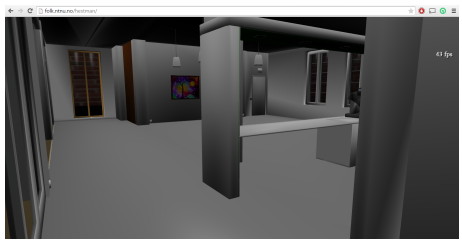
3.5.3 Failed optimizations

3.5.3.1 Incremental loading

Instead of loading the complete model at start up another approach is to load meshes and textures on the fly, when they first are needed, i.e. when they become visible. Babylon.js offers a service for converting the Babylon file into an incremental file, a folder where all meshes are seperated into individual files. Each individual file is then loaded on demand. When the incremental file was applied some objects were being misplaced, and it was therefore decided to discard the incremental loading method.

3.5.3.2 Babylon Scene Optimizer

The Babylon Scene Optimizer is a tool designed to ensure the application to maintain a specific frame rate. This is done be degrading rendering quality at runtime if the frame rate drops beyond the targeted frame rate. The operations the Scene Optimizer conducts includes downscaling textures, merging meshes, shadow optimizations among others. By default Babylon.js offers three predefined degradation sets; low, moderate and high. Independent of which of these sets that were applied problems with floors and walls becoming invisible occurred.



(a) Original

(b) After applying the Scene Optimizer

Figure 3.6: Problems with the predefined Babylon Scene Optimizer

As a consequence a customized optimizer were implemented. By comparing the scene before and after the custom scene optimizer were applied it was hard to tell whether the gain in performance were sufficient enough to defend the level of degradation. The customized scene optimizer was therefore not applied by default, but it can be activated, by pressing the ”O”-key, for testing purposes.

Chapter 4

Results

4.1 Prototype

The prototype developed in this thesis offers a virtual tour of the 1930-building, an administrative building at St. Olav's University Hospital, seen from a first person point of view. The camera is spawn on the inside of the main entrance, and the user can go wherever he/she wants in the five floor building, including the basement and the attic. All doors can be opened, even the outer doors if the user wants to move on the outside of the building. A collection of screenshots from the prototype can be found in Appendix A.

Figure 4.1: The scene seen from the spawn point

4.1.1 Controls

Figure 4.2: Controls

The game controllers, illustrated in Figure 4.2, are as following:

- **Arrow keys:** Moves the position of the camera
- **W, A, S, D:** An alternative to the arrow keys
- **Mouse + holding left mouse button:** Alters the viewpoint
- **Left mouse button click:** Interaction with objects
- **Shift:** Toggle running
- **Space:** Jump
- **F:** Toggle fullscreen mode

4.1.2 Features

One of the goals of this thesis was to present a proof-of-concept of how objects and interaction could be used in a virtual hospital context, either as an activity that simulate real-life task, or in an educational or building exploring context.

4.1.2.1 Art information

One of ideas from St. Olav Eiendom was to replicate the collection of art that is present in the building and let the user explore it and retrieve information about the artwork. Unfortunately St. Olav Eiendom were not able to obtain the data necessary to present the true items. Instead a bunch of example paintings have been placed around the building. Clicking on these painting will trigger a pane, containing information about the artwork, to appear. The pane can be closed by clicking the X in the top-right corner.

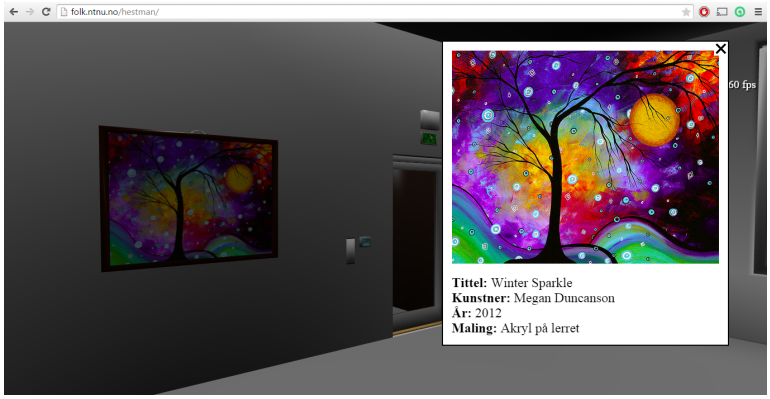


Figure 4.3: Presentation of art information in the prototype

4.1.2.2 Inhouse movies

In the first floor of the building there is projection screen placed in a larger meeting room. By clicking the screen a video starts playing on the screen. In addition there are two buttons to the left of the screen. With these button the user can scroll forward or backward in the list of videos. The videos are published by the National Competence Centre for Ultrasound and Image-Guided Therapy (USIGT) in Norway, and presents a series of cases showing how 3D ultrasound can be used in various neurosurgical procedures. A playing video can be paused by clicking on the screen, and then be continued again by a new click.

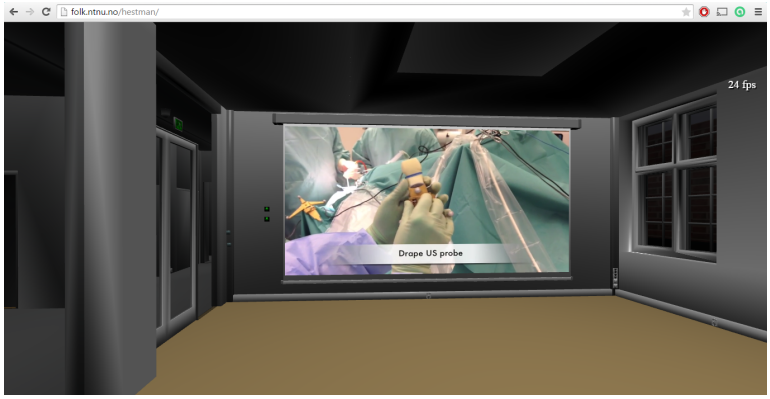


Figure 4.4: Projection screen in the prototype

4.1.2.3 Ultrasound machine

There is a ultrasound machine placed on the second floor of the building. By clicking at the screen of the machine a video stream of ultrasound images will be displayed. By

clicking on buttons on the machine the screen will display other ultrasound images. The ultrasound image collection derives from different cases, for example pregnancy and liver observations.

In a wider scope this feature can be used as a virtual training device. By creating a patient in the virtual environment, and by moving an ultrasonic transducer over the body the screen could projects images according to the position of the transducer. Here are two different approaches of how this can be implemented:

1. Dividing the patient's body into a two-dimensional grid, where for example the womb and the liver are represented in two separate cells. If the transducer is positioned over the cell representing the womb an ultrasound video of an infant in the womb is played on the screen. By moving the transducer over to the field representing the liver, a ultrasound video of an liver examination is played, and so on
2. Representing the whole body with a large image, created by connecting ultrasound images from the whole torso. Then let the transducer position represents the centre with a defined range for the x-axis and y-axis representing the size of the ultrasound screen. The whole image will be applied as an image texture to the screen polygon, and by moving the transducer the uv mapping of the image will be shifted accordingly to the position of the transducer.

Figure 4.5: Ultrasound machine from the prototype

4.1.3 Issues

An issue was noticed in Chrome for Windows concerning file transfer requests to the server. The problem was uncovered when trying to play video on the projecting screen described in subsubsection 4.1.2.2. When clicking the screen the request for the mp4 video would stay pending infinite if the file already was cached. This problem for chrome were actually reported through the Chromium project back in 2013, but have still not been fixed properly. There is a strange workaround for this issue in the prototype. By adjusting the camera viewpoint to be looking out of the building, and by that reducing the number

of draw calls, as there will be less meshes in the viewing frustum, the requests suddenly gets handled.

This issue is restricted to Chrome, and there have been no problems with file transfer requests in the other browsers.

4.2 Browser benchmark test

As this prototype is web-based there are several factors that influences how the application performs. Besides the hardware specifications, the web browser is the most deciding factor. A benchmark test of the major desktop browsers have been conducted to try to evaluated their WebGL 3D rendering performance. This section will give a description of the test procedure, present the results, and discuss the test results.

4.2.1 Test procedure

Three different machines were used in this test. Two of them were laptops, one with a Windows 8 operating system and one which runs Mac OS X. The last machine was a customized media centre running Ubuntu. The machine specifications can be seen in Table 4.1. The prototype developed in this project was used as the test application to measure the performance of the different browsers. Frames-per-second, or fps referred to from now on, represents the unit of measure in this benchmark test. All the desktop browser listed in Table 2.1 from subsection 2.1.2 were tested.

A test program was written to calculate the test results. It is a simple program that calculates the average frame rate while the test is in progress. By pressing the T-key on the keyboard the test program starts running. A counter variable, and a variable for summing the frame rate resets. For each render loop the counter increments and the current frame rate is added to the summing variable. When the counter reaches 1500 the average fps for the test is calculated and displayed, and the test is finished. The time each test run takes depends on the frame rate. With a high frame rate the test takes less time than if the frame rate is low. With a frame rate at 40 fps one test run would take around 30 seconds.

The test was conducted by starting the application, and pressing the T-key on the keyboard, which activates the test program. Then move towards the front door of the building, open it and move inside. Then walk down the corridor, click on the first painting, which makes the pop-up window with art information appear. After closing the pop-up window the trip continues down to the room with the projecting screen. After entering this room the test person clicks to start the video texture on the screen, then move across the room to the stairway and head to the second floor. From here on the test person will just move around the second floor until the test comes to an end, which is likely to happen very soon if the frame rate is not really low.

As mentioned the test was executed on three different machines, with three different operating systems. It was also conducted for two different versions of the prototype. We will

call the first version *full model*, where the complete 3D scene is rendered. In the other, called *light model*, the BIM model covering all the electric devices (RIE) is not present. The light model is therefore less complex. Every single test was conducted three times, and the fullscreen feature was activated in the last test run for each of the browsers. The reason for doing it three times was to make sure every single observation was reliable and similar results therefore could be replicated. For the observations to be considered reliable in this test the highest and lowest observation of the three had to be maximum 10 fps apart. A calculation of an average of the three test runs was considered as the final score.

Machine:	**ASUS S56CB**	**Macbook Pro**	**Custom**
OS:	Windows 8.1	OS X Yosemite	Ubuntu 14.04 LTS
Architecture:	64-bit	64-bit	64-bit
CPU:	Intel Core i5 1.8 GHz	Intel Core i7 2 GHz	Intel Core i5 3 GHz
RAM:	8 GB	8 GB	16 GB
GPU producer:	Nvidia	Intel	Intel
GPU model:	GeForce GT 740M	Iris Pro Graphics	HD Graphics 4600
GPU memory:	2 GB	1536 MB	1.7 GB

Table 4.1: Specifications of machines used in the benchmark test

4.2.2 Benchmark test results

The results from the benchmark tests i presented in Table 4.2. For each browser there are registered six observations for every machine/OS, three for each of the two application versions. The first three white columns represents these three observations. The fourth, blue column, is the concluding result, which is the calculated average of the three observations. All observations are rounded to the nearest integer since fractions of fps are insignificant in this context.

Windows								
Browser:	**Light model**				**Full model**			
Internet Explorer	40	43	39	**41**	25	27	27	**26**
Google Chrome	53	52	50	**52**	48	47	48	**48**
Safari	Unsupported							
Mozilla Firefox	42	40	39	**40**	31	33	29	**31**
Opera	52	53	52	**52**	49	46	49	**48**
OS X								
Browser:	**Light model**				**Full model**			
Internet Explorer	Unsupported							
Google Chrome	38	39	39	**39**	38	37	35	**37**
Safari	23	24	22	**23**	17	16	16	**16**
Mozilla Firefox	39	38	33	**37**	32	30	28	**30**
Opera	52	54	51	**52**	44	44	39	**42**
Ubuntu								
Browser:	**Light model**				**Full model**			
Internet Explorer	Unsupported							
Google Chrome	55	56	56	**56**	54	53	49	**52**
Safari	Unsupported							
Mozilla Firefox	39	38	34	**37**	30	32	29	**30**
Opera	56	55	56	**56**	50	54	51	**52**

Table 4.2: Browser benchmark test result

During the testing of Safari for OS X it became clear that the performance got considerable worse at the moment the video texture was activated. This observation was not as noticeable in other browsers. Anyhow an extra test was conducted where the video texture were not activated to compare this to the original results. Table 4.3 shows the original test results, in the blue column, compared to the result when the video texture was not activated, in the white column. These tests were conducted on the full model.

Browser:	**Windows**		**OS X**		**Ubuntu**	
Internet Explorer	26	26				
Google Chrome	48	48	37	47	52	52
Safari			16	21		
Mozilla Firefox	30	32	30	34	27	30
Opera	48	50	42	46	52	53

Table 4.3: Test results without video textures activated

4.2.3 Test remarks

Safari have discontinued their support for Windows. The latest release for Windows dates back to May 2012. Safari have therefore not been tested for Windows. The same goes for Internet Explorer for OS X, which discontinued the support as long back as in 2005. Internet Explorer and Safari does not exist natively for the Ubuntu operating system. These browser are therefore not included in the tests for Ubuntu.

Activating fullscreen mode was not possible in Internet Explorer for Windows. When activating fullscreen in Safari for OS X the keyboard keys would not work. The image was also stretched in the y-direction, given a left-handed coordinate system.

The observations registered seems reliable as all the three test runs for every unique browser, for each operating system, replicated similar values. At most 5 fps distinguished the highest and lowest observation within a test group.

4.2.4 Benchmark test discussion

First of all the results (Table 4.2) can not be used to determine the browsers performance across the platforms, i.e. conclude that Windows browsers in general performs better than OS X browsers. We have to keep in mind that WebGL, as described in section 2.1, utilizes hardware accelerated graphics. The machines specifications, and the GPU in particular, are crucial for the result. Since different machines, with different specifications, were used for each of the operating systems we can only compare results within the same operating system. The Macbook was the machine with the least powerful GPU so there should be no surprise that this machine also delivered slightly poorer results than the two machines used for the other operating systems.

For Windows Internet Explorer, Google Chrome, Mozilla Firefox and Opera were tested. The observations indicates that Google Chrome and Opera performs far better than IE and Firefox. These test results correlates with the subjective impression during testing. In Chrome and Opera it never felt particular laggy regardless of the model complexity. In Firefox and IE however there were moments of latency, especially for IE running the full model. As Table 4.3 indicates activating video textures did not lower the performance for this platform.

The browsers tested in OS X were Google Chrome, Safari, Mozilla Firefox and Opera. Opera measured the superiorly highest average frame rate for the light model, while the Chrome and Firefox scored more even. Running the full model however Chrome and Opera came closer, while Firefox waned. Safari delivered the far lowest results for both the model complexities. The subjective impression during testing does correlate quite well with the test results, except for Chrome which felt closer to Opera's performance than Firefox' with the light model applied. Safari's result reflects the impression during testing, very laggy compared to the others, and nearly unplayable running the full complexity model with video textures activated.

In contrast to Windows, Table 4.3 shows that performance fell when video textures were activated in the OS X tests. However this is more likely due to the variety of GPU in the testing machines than the browser software itself. A indication of this is that all the OS X browsers encountered a significant drop in frame rate, while this was not the case for Windows and Ubuntu.

In the tests for Ubuntu only 3 browser were tested, Chrome, Firefox and Opera. Opera and Chrome performed best here as well, while Firefox scored somewhat lower. The results match the impression while conducting the test. The playability was acceptable for all the browser, but Chrome and Opera gave a noticeable smoother experience. What stands out for these test results is the consistency the browsers delivers across the different platforms. Chrome and Opera performs very well in all contexts. Firefox delivers an acceptable result across the board, but performs considerably worse than Chrome and Opera. IE and Safari, which only are tested for one operating system each, are struggling. Especially when the complexity increases.

Finally, what does these test scores tell us? What is an acceptable frame rate? There is a quite clear consensus in the gaming community that 30 fps is the baseline for playability. Anything over 30 fps is acceptable, but the higher the better of course. Felix Turner, CEO of SimpleViewer Inc, puts it this way (Turner, 2015):

> *"The higher the frame rate, the smoother your content will be. Stutter and lag kills the brain's flow state. For a game it is especially important that motion is smooth and controls are responsive. Computer screens typically refresh at 60Hz, so this is the maximum bound we aim for. Note that 60FPS is the ideal target, but anything above 30FPS will still look pretty good."*

Looking at Table 4.2 again we see that Internet Explorer for Windows and Safari for OS X were the only browsers who did not reach the 30 fps bar, while Firefox landed just above the 30 fps for all the platforms.

4.2.4.1 Viewing frustum depended frame rate

Even though the average fps is pretty good for most of the browsers the test result do not show the whole picture. The frame rate actually dropped considerably at some points. Even though the decline was bearable, especially in Opera and Chrome, it was certainly noticeable. The points we are talking about are when the viewing frustum covers the larger part of the building, hence contains a great amount of meshes. Recall that Babylon does support view frustum culling, but not occlusion culling. An example where the fps drops significantly is when the camera is position outside and the whole building is covered in the viewing frustum. This problem will be further discussed in the next chapter.

4.3 Test on mobile devices

For mobile devices a less quantifiable test have been conducted, meaning that there was no test assignment with a specific route with tasks to be conducted and recording of average frame rates. While the Browser benchmark test from previous section tries to give an answer of how well the different desktop browsers performs, the purpose of this test is only to get an idea if today's smartphones are mature enough for WebGL.

The tests were conducted on two different devices, Samsung Galaxy S4 and iPhone 6 Plus. Specifications of the devices can be seen in Table 4.4. An important aspect to take note of is that the S4 is two years old and far from a top-notch phone today, while iPhone 6 Plus is Apple's latest and greatest. Google Chrome was the browser used for the Samsung phone, while Safari was used for the iPhone.

In addition two of the Babylon.js demos presented in 2.3.1.2, "Mansion" and "Espilit", were tested on the Samsung S4 for a comparison. Since a fps meter have been implemented both in the Babylon.js demos and for the prototype a general overview of the frame rates will be presented.

Device:	**Samsung Galaxy S4 I9505**	**iPhone 6 Plus**
Released:	April 2013	September 2014
OS:	Android 4.4.2	iOS 8.3
Architecture:	32-bit	64-bit
CPU:	Qualcomm Krait 300 1.9 GHz	ARMv8-A Cyclone 1.4 GHz
RAM:	2 GB	1 GB
GPU:	Qualcomm Adreno 320	PowerVR Series 6 GX6450

Table 4.4: Specifications of mobile devices tested

4.3.1 Mobile test results

With the full model applied the application was barely playable on the S4. The scene was loaded, but the response from touch events were a little slow, making manoeuvring difficult. With only the light model applied it was considerable easier to navigate, even though the latency was certainly visible. Trying to activate the video texture would force the application to crash. The frame rate only occasionally exceeded 10 fps with the full model applied, and was as low as 2 fps at its lowest. Most of the time the frame rate were just above 5 fps.

On the iPhone the application was definitely playable, even with the full model applied, and in contrast to the S4 there were no problems with video textures. Still the experience were not as smooth as in the tests conducted on laptop/desktop. At maximum a frame rate of 30 fps was recorded, and at it lowest it dropped to 9 fps.

For the Babylon.js demos tested on the S4 the results were quite shifting. The scene called "Espilit" worked pretty well. Despite a low frame rate the game was easily manageable

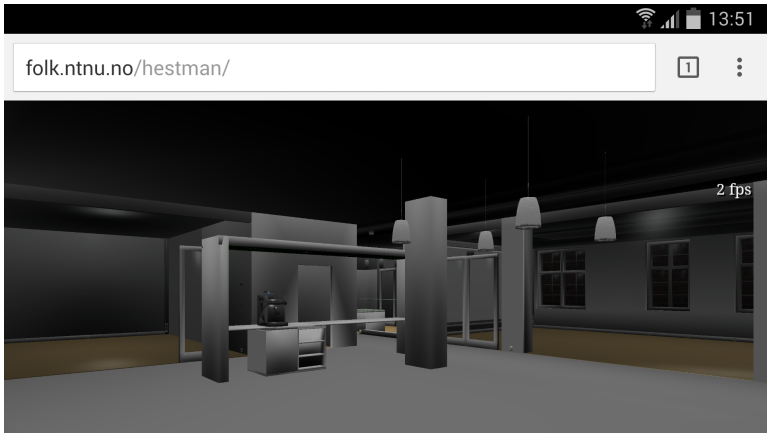


Figure 4.6: Screenshot from test of the prototype on Samsung S4

with quick responses to touch events. Compared to the this thesis' prototype the "Espilit" scene performed better. The "Mansion" scene however were barely playable, and performed slightly worse than the prototype. These two demo scenes are quite similar regarding the number of meshes. However the "Mansion" scene consists of a larger amount of textures, but also more advanced and costly texturing, e.g. moving clouds.

The frame rate for the "Espilit" demo was usually just above 10 fps, with a minimum and maximum frame rate of respectively 7 and 16 fps. In the "Mansion" demo the frame rate never exceeded 5 fps, and hit 2 fps on its lowest. Table 4.5 provides an overview of the test observations. The observations listed are the minimum (min) and maximum (max) frame rate observed, together with the expected (exp) frame rate, i.e. the range of the most common observed values in the tests.

Scene	Test phone	Browser	Min	Max	Exp
1930 building	iPhone 6 Plus	Safari	9	30	13 - 17
1930 building	Samsung S4	Chrome	2	14	5 - 7
Espilit	Samsung S4	Chrome	7	16	10 - 12
Mansion	Samsung S4	Chrome	2	5	3 - 4

Table 4.5: Mobile test results

4.4 Usability test

When testing self-developed applications one can quickly be blinded of which aspects are challenging and what needs improvement. Things can feel very easy and intuitive for the developers as they have designed it and knows how everything works, but this does not necessarily translate to the people that are going to use the application. Outside feedback is therefore very important to highlight the problems and difficulties. To get feedback on this

prototype a test assignment was written, accompanied by a questionnaire with questions related to usability and improvements of the prototype. The complete test assignment introduction and questionnaire can be found in Appendix B

4.4.1 Test assignment

The task of the test assignment was described to the test candidate in the introduction of the questionnaire. In addition the navigation system and controls were explained. The task was defined by a number of instructions to be carried out:

1. Open the first door and move down the hallway. Here are some pictures that you can click on.
2. Go to the room with the projector screen. Click on the screen to play a movie.
3. Click on the screen again to stop/pause the video.
4. Go to the second floor.
5. Find the ultrasound machine and click on its screen.

After conducting the test the test candidate was asked to answer the questionnaire. The questionnaire was short, only seven multiple choice questions, with a focus on how the user mastered the navigation system, and which aspects he/she found troubling. The questionnaire was distributed to a number of people working at St. Olav's to target an audience this type of application may be designed for. A total of 17 people answered the questionnaire, including one who only answered the first four questions. The questions in the survey were:

1. Which browser did you use?
2. How much experience do you have with playing video games?
3. How easy/intuitive did you find the navigation system?
4. Did you manage to accomplish all the tasks?
5. Which aspects did you find problematic?
6. Which changes/improvements do you find most important for improving the user experience?
7. For which purposes do you see a value in a web-based 3D applications?

4.4.2 Questionnaire results

Not surprisingly Google Chrome was the browser most people used with 65 % of the share. Mozilla Firefox and Opera were used by receptively 20 % and 7 %. None of the respondents used Internet Explorer, Safari, or any other browser.

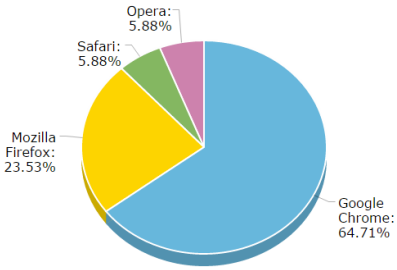


Figure 4.7: Q1: Which browser did you use?

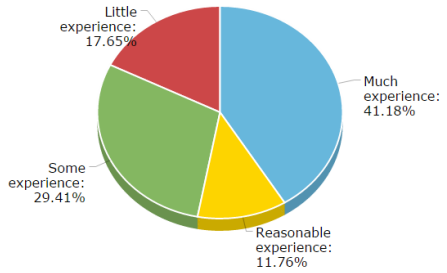


Figure 4.8: Q2: How much experience do you have with playing video games?

41 %, of the respondents answered they had much experience with video games. The rest where distributed over reasonable, some and litte experience, respectively 12 %, 29 % and 18 %. No one answered they had no experience with video gaming. Most people found it easy to navigate with 53 % answering it was easy and 29 % answering it was very easy. 28 % thought it was a bit difficult, while nobody found it very difficult.

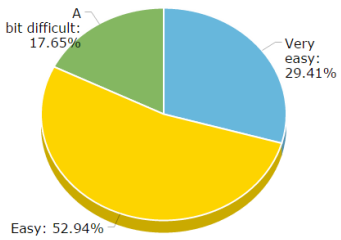


Figure 4.9: Q3: How easy/intuitive did you find the navigation system?

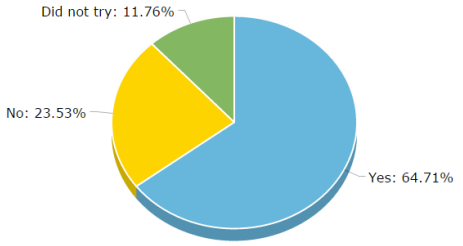


Figure 4.10: Q4: Did you manage to accomplish all the tasks?

By far the largest part, 65 %, managed to execute all the task described in the instruction. 24 % said they did not manage to accomplish all tasks, while 12 % did not try to do all.

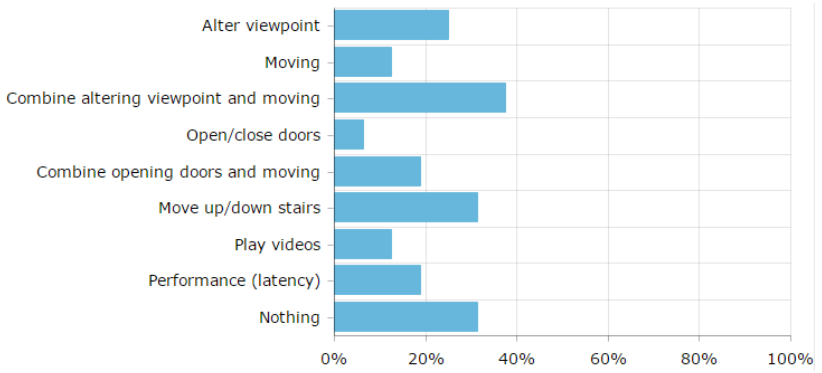


Figure 4.11: Q5: Which aspects did you find problematic?

For the last three questions, that deals with problems and improvements, multiple answers were accepted. The aspects people found problematic were quite evenly distributed. The problem most frequently selected was combining altering the viewpoint and moving, with 38 %. Opening and closing doors were the least problematic aspect according to the respondents, with only 6 %. 31 % did not find anything problematic, and was the second most chosen alternative, together with ”moving up/down stairs”.

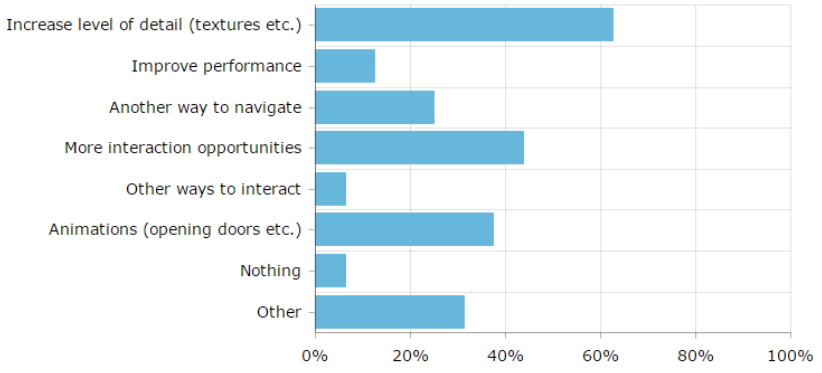


Figure 4.12: Q6: Which changes/improvements do you find most important for improving the user experience?

The most wanted improvement was to increase the visual detail level, with a majority of 63 % pointing this out. Use of animation, for instance when opening doors, were also frequently noted, by 38 %. Other ways to interact were the least wanted improvement with 6 % requesting this. The same amount of respondents answered that they did not see anything that needed improvement. 31 % submitted suggestions of other types of improvements. However all of these beside one could actually pass under the selectable alternatives, which they also had selected. The last suggestion pointed to improvements of lighting.

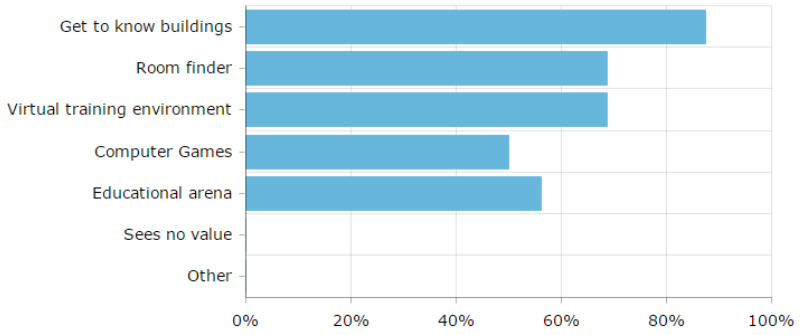


Figure 4.13: Q7: For which purposes do you see a value in a web-based 3D applications?

Of the questions accepting multiple answer the question related to utility value was the one getting the most response. Respectively 88 % and 69 % answered that they saw a value in these types of application for familiarize themselves in real buildings and as a room-finder. 69 % also thought web-based 3D applications have a potential in virtual training scenarios. Nobody answered that they did not see any utility value in web-based 3D applications, nor did any submit an area of use in addition to those listed.

4.4.3 Response patterns

The answers were examined individually to look for patterns based on the respondents choice of browser, and their experience with video games. The patterns mainly looked for were whether video game experience reflects how the way of navigating were perceived, and if choice of browser had an impact on which aspects were regarded problematic.

The Examination showed that there is a connection between experience with video games and how intuitive they found the navigation method. For those who answered they had much or reasonable experience with video games four found the navigation very easy, the same amount answered it was easy, and only one found it a bit difficult. For those with some or little experience only one found it very easy, five said it was easy, and two found it a bit difficult. Apart from this discovery there were no other patterns to be found. The choice of browser for instance did not show any trends in what was perceived as problematic.

Chapter 5

Discussion

In this chapter the test results discussion will be taken further by trying to drag the research questions into the mix.

5.1 WebGL discussion

Subsection 2.1.1 introduces the concepts of hardware based and client based rendering, which are properties of WebGL, as opposed to software based and server based rendering. This section will discuss the strength and weaknesses with this approach.

Where can WebGL's client based rendering approach prove challenging? You always have the machines with weak processing power. However all technology can't be based on supporting the whole spectre of machines regardless of their specification. Actually the best selling 3D games only targets high-end computers, i.e. you can not expect to run the most recent first-person shooter games on the average laptop. You have to invest in an expensive gaming PC. Some of the same can be transferred to WebGL. By utilizing hardware accelerated graphics and client based rendering WebGL makes it possible to render pretty complex 3D scenes in the browser, but the client's hardware has to be of a certain level to fully exploit this technology.

5.1.1 WebGL on mobile devices

There is however a clear difference between the targeted devices for the most advanced 3D games on the marked and the WebGL technology. The WebGL has a wider scope of course as it is an API for developing 3D applications with a variety of complexity. For web use there is especially one platform that has had an explosive growth, namely mobile devices. As Figure 5.1 illustrates the web in the US today is more often accessed

through mobile devices than through desktops (comScore, 2014). The data is gathered by comScore, a company who analyses what people do in the digital world. There are apps that stands for the greater part of web use from mobile devices, which still makes the desktop browser more used than the mobile browsers. However the mobile browsers account for a reasonable share.

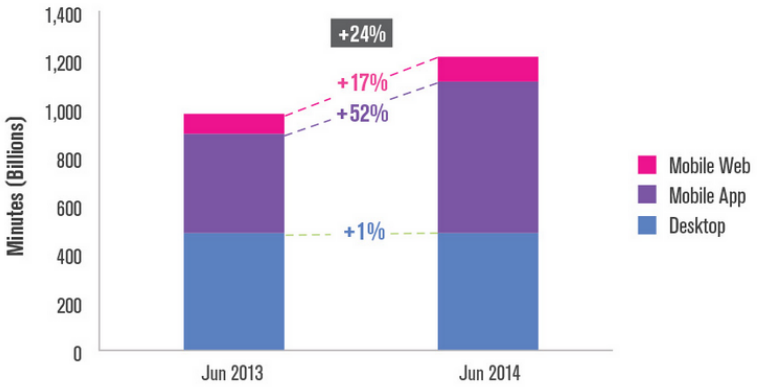


Figure 5.1: Growth in time spent on web in the US from June 2013 to June 2014, divided by mobile app, mobile browser and desktop browser

As with desktops, the specifications of smartphones and tablets can vary a lot. Nevertheless desktops will in general have higher performance than mobile devices. The results from the desktop browser and mobile device tests from previous chapter underpin this statement. What this means is if you want to make an WebGL application to run seamlessly on the average smartphone the complexity might have to be reduced for this platform although it runs seamless on the average desktop. One way this principle easily can be carried out in the prototype is by not importing the RIE model if the client is a mobile device. Since these models are imported separately it is easy to define which should be loaded to the scene, basing the decision on the type of device accessing the website.

Daniel Isaksson, technical director at North Kingdom and contributor at the HTML5 developer community HTML5 Rocks, delivers in an case study (Isaksson, 2014) a list of seven recommendations for WebGL application development to mobile devices and low-spec computers:

- Use low-poly models
- Use low-res textures
- Reduce the number of drawcalls as much as possible by merging geometry
- Simplify materials and lighting
- Remove post effects and turn off antialiasing
- Optimise Javascript performance

- Render the WebGL canvas at half size and scale up with CSS

It can seem like only the newest top-notch mobile devices are up the task of rendering large complex 3D scenes today. Well, that might be true for the WebGL technology alone. Here we come back to client based versus server based rendering. Döllner et al. (2012) published a paper where they presented a server based rendering solution for large 3D city models on mobile devices. They describe the solution this way:

> *Our approach uses a different strategy that relies on server-based 3D scene rendering and client-based 3D scene reconstruction based on virtual panoramas, technically represented by G-buffer cube maps. This way, we can avoid streaming 3D scene data to clients and, therefore, decouple 3D scene complexity from data transmission complexity. In addition, the server can use advanced 3D rendering technology, while only moderate 3D graphics capabilities on the clients are required.*

What this tells us is that by making use of server based rendering low-spec devices can render more complex 3D scenes than they would have been able to by only using a client based rendering technology, as for example WebGL. This does not mean that the WebGL technology cannot be a part of the solution since it is a client-side renderer. Actually the client implementation in Döllner et al.'s work uses WebGL-enabled browsers. The WebGL API is used as in other WebGL applications to display a 3D scene through the HTML5 <canvas> element, but the costly rendering process is "outsourced" to the server.

5.1.2 Client based versus server based rendering

What the example in the previous subsection, 5.1.1, proves is that server based rendering might offer a better service for low-spec machines than client based rendering. So is there any reason to use client based rendering instead? As just pointed out, Döllner et al.'s solution addresses a problem for low-spec devices. What about clients with better processing power, will they benefit from a server based rendering approach? This question is impossible to answer on a general basis. There are multiple factors involved. However two criteria must be met for server based rendering theoretically can perform better.

1. The server-side hardware must have greater processing power than the client-side hardware.
2. Server-side computational latency and network latency combined must be lower than the client-side computational latency.

In addition there are other factors involved, for example server demand and overhead. Martin (2000) argues that the main tradeoff with server-side rendering is loss of real-time interaction due to network latency and that the rendered images are delivered to clients at non-interactive rates. This is not wrong, but we have to keep in mind that Martin's paper is from 2000, and much have happened technology-wise since this time, including growth in bandwidth rates, which is very relevant for Martin's argument. Nilsen's Law of Internet Bandwith states that *"a high-end user's connection speed grows by 50 % per year"* (Nielsen, 1998). We have examples from recent times of server based 3D rendering

services where real-time interaction is essential. One example is OnLive, a cloud gaming service, where the client handles the input, transmits it to OnLive's servers which renders the game and provides a streaming video back to the client. The main purpose with this service is, as already pointed out in this section, allowing games to run on machines that have insufficient hardware to run it locally.

Another approach is to combine client based and server based rendering where the model is partitioned into different parts, assigning some parts to be rendered on the server, while other is rendered on the client. The main challenge with this approach is the task of deciding the partitioning, which is not trivial.

Server based rendering:

+ Might achieve higher performance than the client originally can achieve
+ A more unambiguous result by reducing the importance of client hardware
− Development complexity increases
− Introduces network latency
− The cost of server hardware
− Heavy server demand may decrease performance

Client based rendering:

+ Can utilize powerful client hardware
+ Excluding the network latency factor
− Performance very dependent on the client's hardware
− Harder to target a large range of devices

5.2 Applying BIM models in web-based games

The whole idea behind using BIM models in this project was to replicate an authentic environment, without having to do a huge modelling job. This section will discuss whether this goal was accomplished through this solution, and how well BIM models integrates for this purpose.

What cannot be argued is the fact that BIM models provide a model that is true to a structure's real measures. The building that is being rendered is identical to the real-life building when it comes to form, shape, room division and position of the components. section 2.2 has already addressed how the use of BIM has accelerated the last 15 years, and how common it is in larger building projects today. It therefore exists a great amount of 3D models over various real-life buildings. In addition the demand for virtual reality software is increasing, with a significant adoption in the training and infotainment sectors (MarketsandMarkets, 2014). For these types of systems BIM models have a potential as

a great share of them aims for replicating existing buildings and structures in the virtual environment.

The drawbacks of using BIM models in a game-like environment originates from the fact that the models are made for a another specific purpose, namely building planning. The model complexity is a central point. When modelling a 3D scene for a video game only visible, or in somehow intractable objects are of interests. The designer will not model the water pipes inside a wall etc. if these under no circumstances will come into sight. It is not just a waste of time for the modeller, it also increases the size and complexity of the model, without bringing any benefits. For a BIM model however all parts of the building, visible for the audience or not, is necessary to fulfil its purpose, providing a complete model that describes every single detail relevant to the building project.

5.2.1 BIM models applied in the prototype

As explained in section 2.2 BIM models are usually divided into different disciplines, and only the ARK and RIE models were applied in the prototype, which mean you can discard parts of the model you don't find necessary. Still the ARK model and the RIE model combined contains well above 2000 meshes. If we compare this to the Babylon.js demos presented in section 2.3, and tested in section 4.3, none of these scenes consist of more than 150 meshes. The scene from this prototype does admittedly require a greater amount of meshes as the building is considerable larger with a lot of smaller rooms etc. compared to the structures present in the Babylon demo scenes. However the amount of meshes could have been dramatically reduced, without damaging the visual appearance, if it was modelled exclusively for a game purpose.

Another important remark is the lack of texturing in the BIM models of the 1930 building. Most objects in the RIE model had at least been painted in their true colours, but only a few materials in the ARK model have a different colour than the default grey tone that is repeated throughout the 3D scene. Nor are any textures used besides those added to the self-modelled objects. The result of this is that the scene looks quite naked, which also were pointed out in the usability test questionnaire.

This problem could of course be solved manually by adding textures and colours to the objects in Blender, although this would be a major job, especially if the textures were to reflect the authentic appearance of the building. It was considered starting on this job, but it was put on hold, as it would be a time consuming operation and not vital regarding the focus area of the thesis. One factor that would add to the workload is the nature of how the faces of the meshes are structured, and the fact that some walls represents both the inside and the outside of the building. The whole wall cannot therefore be assigned a common texture, but each individual face making up the inside wall needs to be selected manually to not apply the same texture or color on the exterior wall. Figure 5.2 illustrated how this gets a little messy due to the structure of the face division.

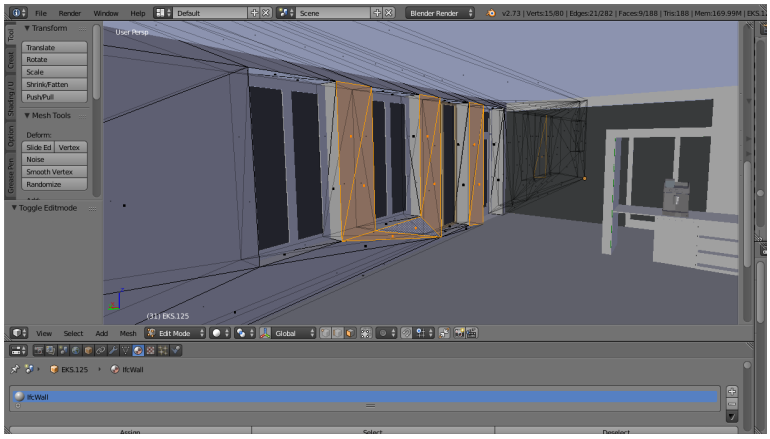


Figure 5.2: Selecting faces of a inner wall in Blender

5.3 Using higher-level WebGL frameworks

Use of higher-level framework to improve or simplify the work is something that applies to most software development projects. There are several good reason to use them. Mike Baker, cluster tool control practice manager at Cimetrix, advocates it this way (Baker, 2009):

> *"The purpose of a framework is to improve the efficiency of creating new software. Frameworks can improve developer productivity and improve the quality, reliability and robustness of new software. Developer productivity is improved by allowing developers to focus on the unique requirements of their application instead of spending time on application infrastructure ("plumbing")."*

Especially for a low-level API like WebGL a high-level frameworks can come in handy. The main advantages of using these are to abstract away the complex WebGL setup and graphics pipeline. WebGL is actually even harder to get started with than OpenGL. This is because WebGL inherits from OpenGL ES 2.0, which is a stripped down version of OpenGL, where the collection of beginner-friendly functions among others are removed to make the system smaller and simpler. What you get from these high-level frameworks or libraries is a layer of defined functions that will do the WebGL API calls for you. The developer will have a simpler interface to deal with, it will require a lot less code and conducting error handling will be easier, given that the error is not within the framework.

Brandon Jones, a Chrome WebGL implementer at Google, wrote a blog post back in August 2011 with the title "WebGL Frameworks are awesome, here's why I don't use them" (Jones, 2011). In this post he mentions one, and one reason only why he prefer to do without a higher-level framework, which is performance. By adding an extra layer you obviously get overhead, the question is to what extent. Jones argues that in the vast majority of cases the overhead is not decisive for the result, but he questions if a high-performance

WebGL app can afford it. It is uncertain what qualifies as high-performance WebGL app according to Jones, but the main point is that by discarding a higher-level framework you can exploit WebGL to the fullest performance-wise.

Another sensitive point is the limitations the framework creates. When applying a framework you are dependant on what it has to offer. How clamped you are to a single framework can differ. Some frameworks gives more freedom and are easier to combine with native code than others. What is important is to do a thoroughly study when deciding on a framework to use, making sure it actually will benefit your project and will not create limitations for your system's requirements.

5.3.1 Babylon.js evaluation

In the preliminary study an analysis of a bunch of WebGL frameworks was conducted, and the Babylon.js framework were chosen on this basis. Table 5.1 shows an excerpt of Babylon.js' notations from the comparison in Table 2.4. As the frameworks examined in the preliminary study were not tested in practice this subsection will evaluate how accurate the assessments were for Babylon.js.

Framework:	OS	Free	Com	Doc	Phy	Col	Imp
Babylon.js	Yes	Yes	Active	Fair	Yes	Yes	Yes

Table 5.1: Excerpt of the WebGL framework comparison in Table 2.4

The project is still free and open source and there is no plans of making changes here. The repository is available from GitHub, and is continuously under development. New contributions or improvements are pushed weekly. One of the main factors for choosing to work with Babylon was the activity level in the community. The main discussion forum for the framework is located at www.html5gamedevs.com. There is non-stop activity on the forum and major contributors to the framework sacrifice a lot of time to answer questions.

The only criteria that did not get top grade in the framework assessment in the preliminary study was the documentation. Even though there are a lot of good tutorials describing how to master the different aspects of creating a 3D world with Babylon.js the API documentation is a little incomplete. All classes and methods are accounted for, but the detail level could be better. There were times in the project were the application of methods were not completely understood due to the less detailed documentation, and the forums were used instead to clarify.

The physics and collision engines were simple to work with, and are already described in the development chapter. What stands out the most with Babylon.js is the richness of the library and how easy the setup is. Large snippets WebGL code is done with a simple method call to the Babylon library.

The biggest downside encountered with the framework were the lack of support for occlusion culling. This shortage became extra apparent due to the high complexity level in

the BIM models, i.e. the great amount of meshes. As addressed in the browser benchmark testing, section 4.2, the frame rate varied depending on the direction of the viewing frustum

In a scenario where the application for instance would be directly connected to a BIM model server to be able to automatically present models with the latest modifications, using the Babylon framework would not be ideal. As the framework only imports Babylon-files the IFC-files needs to be opened and converted to the Babylon file format with Blender or one of the other 3D modelling software supporting the Babylon export plug-in.

5.4 Potential use

The great thing about web-based applications is that they are available everywhere, at any time. All the user needs is a computer and a web browser. There is no need to install any additional software. The results of the questionnaire, presented in subsection 4.4.2, indicated that people see great utility value in web-based 3D applications. This section will discuss types of solutions where a more refined/modified version of this prototyped could be applied. The following example solutions were alternatives frequently checked by the respondents.

5.4.1 Room finder

For services that assists in navigating and locating things there have and there always will be a demand for. To emphasize this Google Maps usally ranges between the 1st and the 5th most popular smartphone app on the planet, depending on sources. According to the GlobalWebIndex the app was used by 54 % of the global smartphone population during July 2013 (Mari, 2013). However other services are needed for finding the way on the inside of buildings. MazeMap is a service that provides interactive indoor maps to help users to find their way in larger public facilities like universities, hospitals etc.

What can be a challenge with maps is that you are looking at them from another perspective than the first-person perspective a human inside a building has. If the building is very large, with many corridors and small rooms, you can easily lose track. Nor are there any recognizable visually objects that reminds you that you are on the right track. For instance a scenario most people, if not all, have experienced is walking around in an area they only have been in a couple of times before and are unsure if they are walking the right way. Suddenly they see this statue etc. that reminds them that they are on right track. With a 3D room finder, like an inside building version of Google Street View, people visiting a unknown building can find the way to their objective in advance in the virtual world and then easier find their way in real life.

By taking advantage of existing BIM models buildings can be authentically replicated without the development team having to do a massive modelling job, not to mention gathering all the building information laying the foundation for the modelling.

5.4.2 Educational arena for kids

In later years use of IT have been more and more common to integrate in the education in primary schools. Apart from exploiting IT for efficient learning, using it as motivation, by making the learning process more fun for the pupils, is another factor. Almost every kid in the western world play video games, and a study published in the Journal of Adolecent Health shows that kids like to play mature rated games, like for instance first-person shooters (Gitlin, 2007). The navigation form in this prototype is therefore something most kids recognizes from their gaming experience.

Creating a virtual classroom or school building where the user can move around and activate different tasks is an example of a solution. By using WebGL and have it available from a website means that pupils can access the application at home and might be something that tempts pupils to do extra work outside of school.

5.4.3 Virtual hospital

In the introduction of this thesis the idea of creating a role-playing virtual training environment for nurse anaesthetists was brought in. Virtual training methods are getting more and more integrated in the education in various fields, mentioning armed forces, emergency services and healthcare.

As the hospitals increase their effectiveness e.g. by lean patient pathways, there is dramatically less time with the patients for learning purposes. There are also clinical situations where students can not be present. There is thus a need for solutions that give students more time on the tasks, as well as possibilities to train and interact together across professions. One solution is to provide students with flexible online educational applications that must be embedded in a holistic system to be usable. Therefore the idea of an online virtual university hospital has emerged, to be used as a venue for learning, research and development.

Use of BIM models in the virtual reality training in the health sector can prove particularly suitable as the user will be familiar with the environment, given that the models used are from their own hospital. Everything can be evaluated in its right environment, from simple tasks like testing new equipment that is delivered with a virtual training program, to complex operations and patient pathways. The virtual hospital can be used as an arena for virtual education and simulation independent of time and space throughout the lifetime of the physical hospital and physical changes to the real environment can be assessed virtually first. Patients are explained where the examination will take place and how it is conducted using virtual models. A virtual patient can be used to explain medical students anatomy and physiology. A virtual representation of the knowledge portal or for instance an ultrasound museum can be visited at any time from anywhere in the world.

Chapter 6

Conclusion and future work

6.1 Conclusion

In the introduction these four research question were raised:

1. In what way is use of BIM models suitable with WebGL?
2. The major web browsers compatibleness with WebGL
   (a) Which support the technology?
   (b) How do they compare performance-wise rendering 3D using WebGL?
3. What exists of higher-level WebGL frameworks today, and how do they compare?
4. How does the WebGL technology adapt to the wide range of devices of different performance capabilities?

Topics regarding these questions have already been discussed throughout this thesis and this chapter will try to form a conclusion to the research questions based on these discussion and the results presented.

1. In what way is use of BIM models suitable with WebGL?

As demonstrated with the prototype and the results from the browser benchmark test fairly complex BIM models can be applied in a WebGL application and perform acceptable. There are however some sensitivity points related to the use. The complexity in the BIM models are for most game-like purposes unnecessarily large, and a similar structure could have been modelled much more efficient. The biggest problem encountered in this project were how the combination of a complex model and a WebGL framework without support for occlusion culling resulted in an variable frame rate.

The best reason for using BIM models in a game-like application would be in a setting where replication of real buildings are desired, without having the time or resources to do the modelling job yourself.

2. The major web browsers compatibleness with WebGL

All of the five most used desktop web browser supports WebGL, in addition to all of the most popular mobile browsers. Their performance are on the other hand very variable. Frame rate-wise the browser benchmark test uncovered that Opera and Chrome were superior regardless of platform. Safari and Internet Explorer performed rather poor, while Mozilla Firefox positioned itself between these groups. In Chrome there was a problem with file requests to the server in situations with a large amount of draw calls, which problem was not present in the other browsers. This Chrome issue is not exclusively a WebGL problem. Based on the results in this thesis a ranking of the desktop browser WebGL capabilities would look like this:

1. Opera
2. Google Chrome
3. Mozilla Firefox
4. Internet Explorer
5. Safari

3. What exists of higher-level WebGL frameworks today, and how do they compare?

This question were examined already in the preliminary study, to create a basis for deciding framework in the development process. There are numerous of higher-level WebGL framework, and only a handful of them were presented in this thesis. The study uncovers that most of the frameworks have the same approach and offer essentially the same features. Of the examined framework in this thesis Goo Enginge, PlayCanvas, Turbulenz, CopperLicht and Unity were the frameworks most similar to Babylon.js, the framework used in the development of the prototype. Three.js and SceneJS are framework that differs from these by not being designated game creator frameworks. Three.js is a general purpose 3D animation framework, and SceneJS focus on engineering visualization etc.

Doing a thorough examination of the various frameworks before deciding technology can prove valuable later in the development phase. In hindsight choosing framework with support for occlusion culling might have been a better solution in this project due to the nature of the BIM models.

4. How does the WebGL technology adapt to the wide range of devices of different performance capabilities?

Being a client side hardware based rendering technology the performance of WebGL applications are highly dependant on the specifications of the machine running it. The results show that the prototype developed, consisting of a relative complex scene, perform adequate on the average laptop. For mobile phones rendering complex 3D scenes with WebGL can be a challenge, especially for the phones that are not among the newest and most powerful. In a future perspective there are no reason to believe that this challenge will prevail, as the GPU in mobile only will get more powerful. Still, as mobile hardware gets better the same goes for the hardware in desktops, and developers need to keep that in mind the performance gap when developing resource demanding WebGL application targeting both platforms. A good approach is to create scalable applications where the complexity can be downgraded for mobile devices or low-spec computers.

6.2 Future work

6.2.1 Prototype development

The idea for the prototype was to replicate the 1930 building at St. Olav's Hospital, and be able to roam it freely. The most important works that remains to make the prototype more fully fledged is to increase the detail level. The results of the usability test also implied this. The main problem was the lack of textures in the BIM models, and that the model for the interior design (IARK) was not completed within the time this thesis was delivered.

When the IARK model becomes complete it can be imported in the same fashion as the ARK and RIE model. Textures for the ARK model will probably not be included in the nearest time. Here a manual texturing job in Blender can be conducted to give the environment more life and colour, and preferable gather information about the building first to recreating it as similar as possible. The same can be done for the paintings. I.e. gather information about the actual artwork and their positions in the building, and then replace the current sample paintings.

6.2.2 Further work utilizing the concept

Supervisor for this thesis, Frank Lindseth, and Tor Åsmund Evjan at St. Olav Eiendom plans to work further with the concept in this thesis in future projects/master's theses. The goal in a longer perspective is to combine the interests of Lindseth, to create a web-based virtual training environment for health professionals at St. Olav's (described in subsection 5.4.3), and St. Olav Eiendom's interests, which is to use the BIM model data directly from their model server.

The main challenge with this approach is that we are dependant on the detail level of the BIM models on the model server to be fulfilling, since they will be loaded directly, hence no opportunities of manually doing modifications in Blender etc. Secondly if the Babylon.js framework is to be used a server side script that converts the IFC files to the Babylon file format is needed, as this is the only file format the framework can import.

Bibliography

Aranda-Mena, G., Crawford, J., Chevez, A., Froese, T., 2009. Building information modelling demystified: does it make business sense to adopt bim? International Journal of Managing Projects in Business 2 (3), 419–434.

Baker, M., 2009. What is a software framework? and why should you like 'em? Available: http://info.cimetrix.com/blog/bid/22339/What-is-a-Software-Framework-And-why-should-you-like-em, (Accessed: 02.06.2015).

Bergin, M., 2012. History of bim. Available: http://www.architectureresearchlab.com/arl/2011/08/21/bim-history/, (Accessed: 09.04.2015).

Cantor, D., Jones, B., 2012. WebGL Beginner's Guide. PACKT publishing.

comScore, 2014. Media metrix multi-platform ands mobile metrix, u.s., june 2013 to june 2014.

Curran, K., 2012. The future of web and mobile game development. International Journal of Cloud Computing and Services Science 1, 25–34.

Döllner, J., Hagedorn, B., Klimke, J., 2012. Server-based rendering of large 3d scenes for mobile devices using g-buffer cube maps. In: Mouton, C., Posada, J., Jung, Y., Cabral, M. (Eds.), The 17th International Symposium on Web3D Technology, Web3D '12, Los Angeles, CA, USA, August 4-5, 2012. ACM, pp. 97–100.
URL http://dl.acm.org/citation.cfm?id=2338714

Gitlin, J. M., 2007. New survey shows that kids like games rated m for mature. Available: http://arstechnica.com/science/2007/07/new-survey-shows-that-kids-like-games-rated-m-for-mature/, (Accessed: 02.06.2015).

Henry, A., 2014. Most popular text editor: Notepad++. Available: http:

//lifehacker.com/five-best-text-editors-1564907215/1567287033, (Accessed: 23.04.2015).

Hergunsel, M. F., 2011. Benefits of building information modeling for construction managers and bim based scheduling. Master's thesis, Worcester Polytechnic Institute.

Hewitson, J., 2013. Three.js and babylon.js: a comparison of webgl frameworks. Available: http://www.sitepoint.com/three-js-babylon-js-comparison-webgl-frameworks/, (Accessed: 28.01.2015).

infoComm, 2011. Building information modeling (bim) guide. Tech. rep., infoComm International.

Isaksson, D., 2014. The hobbit experience, bringing middle-earth to life with mobile webgl. Available: http://www.html5rocks.com/en/tutorials/casestudies/hobbit/, (Accessed: 13.05.2015).

Jones, B., 2011. Webgl frameworks are awesome, here's why i don't use them. Available: http://blog.tojicode.com/2011/08/frameworks-are-awesome-heres-why-i-dont.html, (Accessed: 03.06.2015).

Kleven, N. F., 2014. Virtual university hospital as an arena for medical training and health education. Master's thesis, Norwegian University of Science and Technology.

Krüger, M., 2014. Flash is dead ... long live webgl. Available: http://insights.wired.com/profiles/blogs/flash-is-dead-long-live-webgl#axzz3S6S4gx6u, (Accessed: 18.02.2015).

Kumar, S., Hedrick, M., Wiacek, C., Messner, J. I., 2011. Developing an experienced-based design review application for healthcare facilities using a 3d game engine. Journal of Information Technology in Construction 16 (85).

Lin, K.-Y., Son, J., Rojas, E. M., 2011. A pilot study of a 3d game environment for construction safety education. Journal of Information Technology in Construction 16 (69).

Mari, M., 2013. Top global smartphone apps, who's in the top 10. Available: http://www.globalwebindex.net/blog/top-global-smartphone-apps, (Accessed: 02.06.2015).

MarketsandMarkets, 2014. Augmented reality & virtual reality market by technology types, sensors (accelerometer, gyroscope, haptics), components (camera, controller, gloves, hmd), applications (automotive, education, medical, gaming, military) & by geography - global forecast and analysis to 2013 - 2018. Tech. rep., MarketsandMarkets.

Martin, I. M., 2000. Adaptive rendering of 3d models over networks using multiple modalities. Tech. rep., IBM T.J. Watson Research Center.

McCoy, R., 2013. Ie11 fails more than half tests in official webgl conformance test suite.

Available: https://connect.microsoft.com/IE/feedback/details/795172, (Accessed: 17.02.2015).

McCuthan, J., 2012. Pointer lock and first person shooter controls. Available: http://www.html5rocks.com/en/tutorials/pointerlock/intro/, (Accessed: 22.05.2015).

McGraw-Hill, 2012. The business value of bim in north america: Multi-year trend analysis and user rating smartmarket report. Tech. rep., McGraw-Hill Construction.

Nielsen, J., 1998. Nielsen's law of internet bandwidth. Available: http://www.nngroup.com/articles/law-of-bandwidth/, (Accessed: 14.05.2015).

Ortiz, S., 2010. Is 3d finaly ready for the web? IEEE Computer 43, 14–16.

Parisi, T., 2012. WebGL Up and Running. O'Reilly Media, Inc.

Pesce, M., 2014. ios 8 release: Webgl now runs everywhere. hurrah for 3d graphics! Available: http://www.theregister.co.uk/2014/09/17/after_20_years_apple_finally_enters_the_third_dimension/, (Accessed: 17.02.2015).

Rosenblatt, S., 2012. Faster graphics for older pcs in chrome 18. Available: www.http://download.cnet.com/8301-2007_4-57405953-12/faster-graphics-for-older-pcs-in-chrome-18/, (Accessed: 17.02.2015).

Shen, Z., Jiang, L., Grosskopf, K., Berryman, C., 2012. Creating 3d web-based game environment using bim models for virtual on-site visiting of building hvac systems. In: Cai, H., Kandil, A., Hastak, M., Dunston, P. S. (Eds.), Construction Research Congress 2012: Construction Challenges in a Flat World. American Society of Civil Engineers, pp. 1212–1221.

Stangarone, J., 2013. 7 major web development trends of the next 5 years. Available: http://www.mrc-productivity.com/blog/2013/04/7-major-web-development-trends-of-the-next-5-years/, (Accessed: 03.06.2015).

Statsbygg, 2009. Statsbyggs generelle retningslinjer for bygningsinformasjonsmodellering (BIM). Norwegian Directorate of Public Construction and Property.

Turner, F., 2015. Building a 60fps webgl game on mobile. Available: http://www.airtightinteractive.com/2015/01/building-a-60fps-webgl-game-on-mobile/, (Accessed: 12.05.2015).

van Nederveen, G., Tolman, F., 1992. Modelling multiple views on buildings. In: Automation in Construction. Elsevier, pp. 215–224.

Withers, I., Matthews, D., 2011. All government projects to use bim within five years. Available: http://www.building.co.uk/all-government-projects-to-use-bim-within-five-years/5018349.article, (Accessed: 14.05.2015).

Xeolabs, 2011. Scenejs vs three.js vs others. Available: http://www.stackoverflow.com/questions/6762726/scenejs-vs-three-js-vs-others, (Accessed: 12.02.2015).

Appendix

Appendix A

Screenshots

Following is a collection of screenshots from various locations of the scene.

Figure A.1: Building seen from the outside

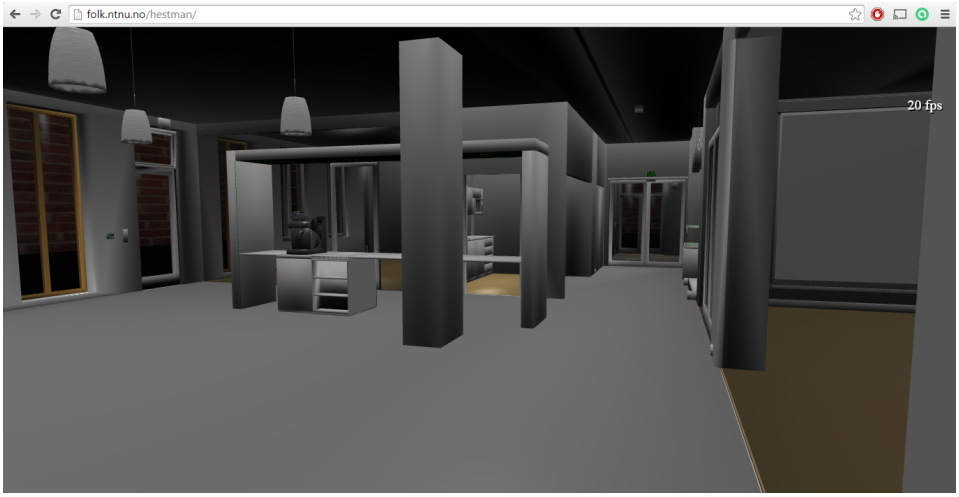


Figure A.2: First floor

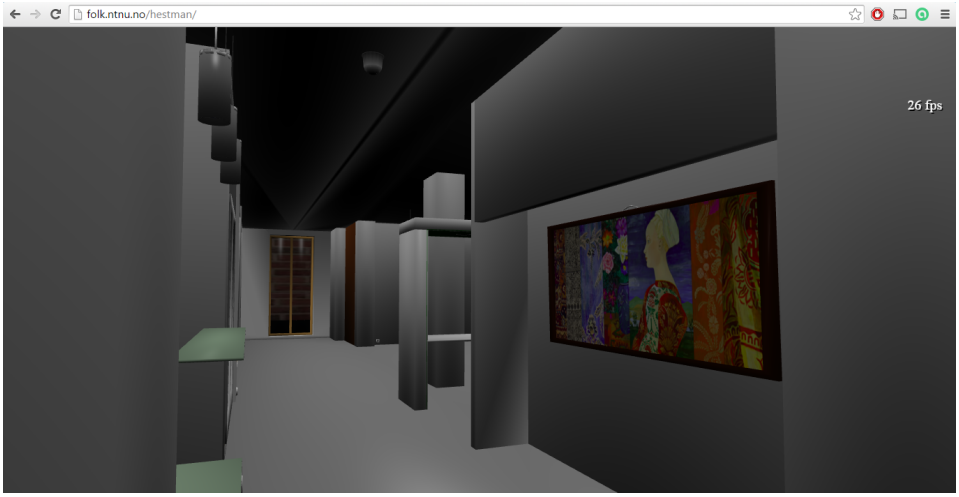


Figure A.3: First floor hallway

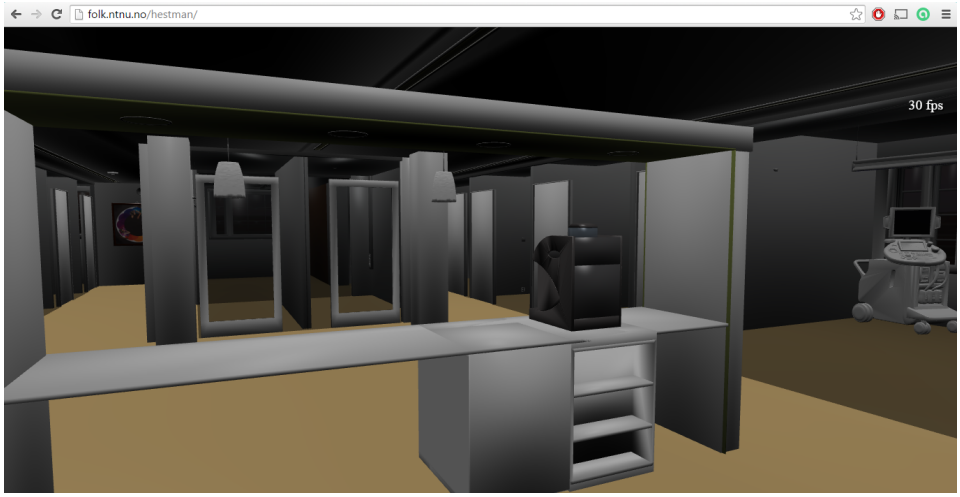


Figure A.4: Second floor office space

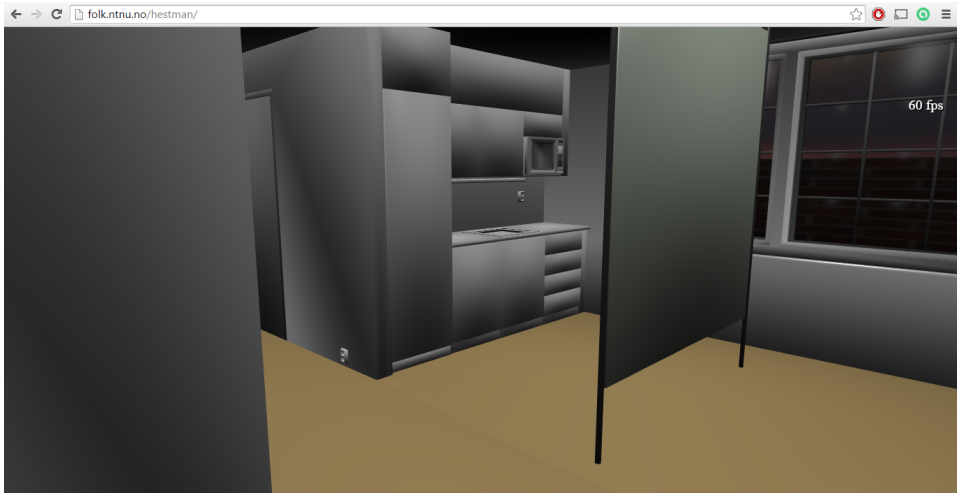


Figure A.5: Second floor kitchen

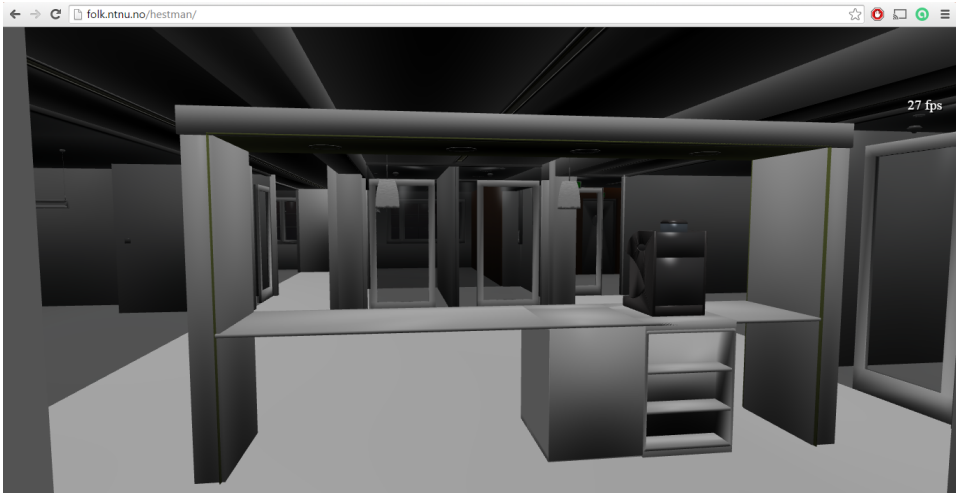


Figure A.6: Third floor office space

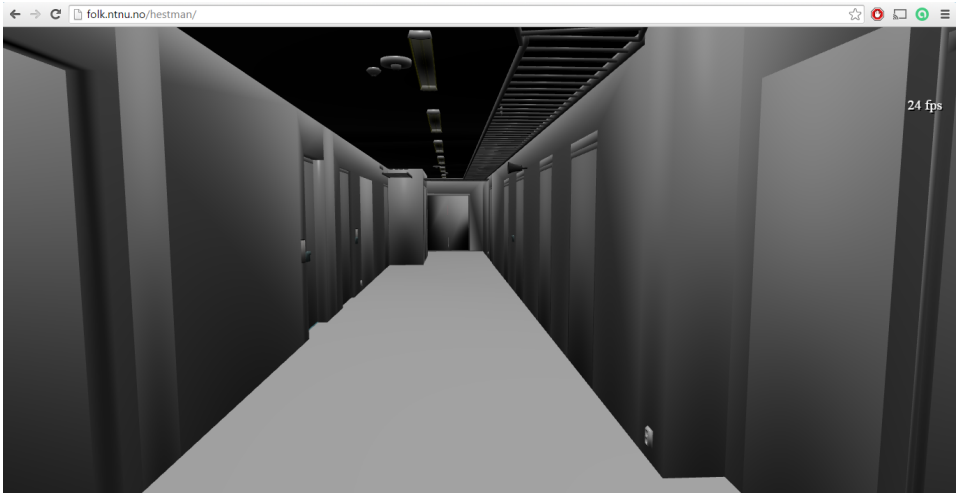


Figure A.7: Basement hallway

Figure A.8: Basement wardrobe

Figure A.9: Stairway

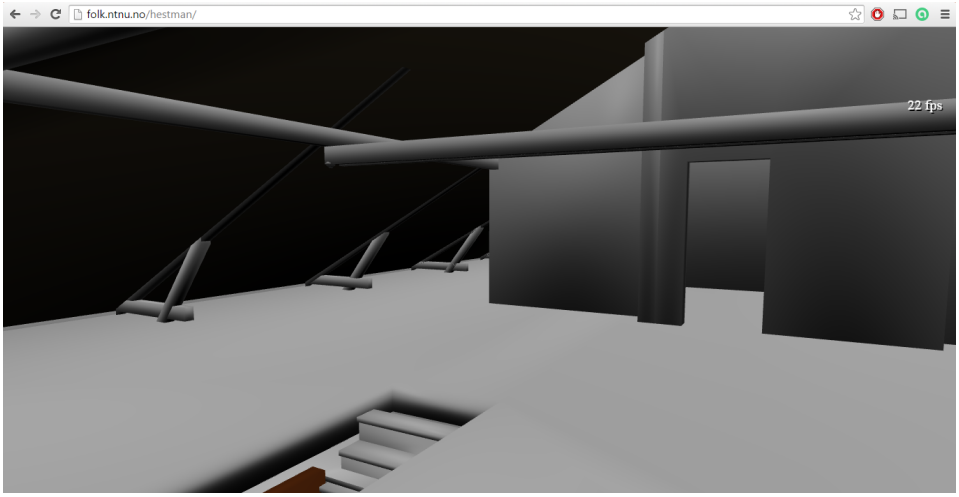
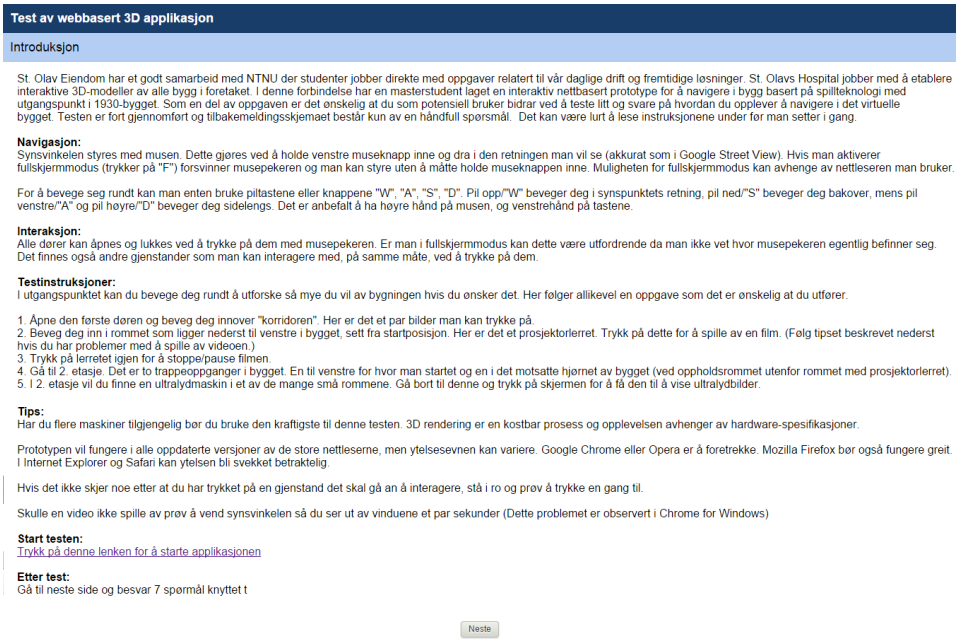


Figure A.10: Attic

Appendix B

Questionnaire

This appendix chapter shows the usability test questionnaire presented to the test group. Figure B.1 is the introduction page, while Figure B.2 contains the questions.



Test av webbasert 3D applikasjon

Introduksjon

St. Olav Eiendom har et godt samarbeid med NTNU der studenter jobber direkte med oppgaver relatert til vår daglige drift og fremtidige løsninger. St. Olavs Hospital jobber med å etablere interaktive 3D-modeller av alle bygg i foretaket. I denne forbindelse har en masterstudent laget en interaktiv nettbasert prototype for å navigere i bygg basert på spillteknologi med utgangspunkt i 1930-bygget. Som en del av oppgaven er det ønskelig at du som potensiell bruker bidrar ved å teste litt og svare på hvordan du opplever å navigere i det virtuelle bygget. Testen er fort gjennomført og tilbakemeldingsskjemaet består kun av en håndfull spørsmål. Det kan være lurt å lese instruksjonene under før man setter i gang.

Navigasjon:
Synsvinkelen styres med musen. Dette gjøres ved å holde venstre museknapp inne og dra i den retningen man vil se (akkurat som i Google Street View). Hvis man aktiverer fullskjermmodus (trykker på "F") forsvinner musepekeren og man kan styre uten å måtte holde museknappen inne. Muligheten for fullskjermmodus kan avhenge av nettleseren man bruker.

For å bevege seg rundt kan man enten bruke piltastene eller knappene "W", "A", "S", "D". Pil opp/"W" beveger deg i synspunktets retning, pil ned/"S" beveger deg bakover, mens pil venstre/"A" og pil høyre/"D" beveger deg sidelengs. Det er anbefalt å ha høyre hånd på musen, og venstrehånd på tastene.

Interaksjon:
Alle dører kan åpnes og lukkes ved å trykke på dem med musepekeren. Er man i fullskjermmodus kan dette være utfordrende da man ikke vet hvor musepekeren egentlig befinner seg. Det finnes også andre gjenstander som man kan interagere med, på samme måte, ved å trykke på dem.

Testinstruksjoner:
I utgangspunktet kan du bevege deg rundt å utforske så mye du vil av bygningen hvis du ønsker det. Her følger allikevel en oppgave som det er ønskelig at du utfører.

1. Åpne den første døren og beveg deg innover "korridoren". Her er det et par bilder man kan trykke på.
2. Beveg deg inn i rommet som ligger nederst til venstre i bygget, sett fra startposisjon. Her er det et prosjektorlerret. Trykk på dette for å spille av en film. (Følg tipset beskrevet nederst hvis du har problemer med å spille av videoen.)
3. Trykk på lerretet igjen for å stoppe/pause filmen.
4. Gå til 2. etasje. Det er to trappeoppganger i bygget. En til venstre for hvor man startet og en i det motsatte hjørnet av bygget (ved oppholdsrommet utenfor rommet med prosjektorlerret).
5. I 2. etasje vil du finne en ultralydmaskin i et av de mange små rommene. Gå bort til denne og trykk på skjermen for å få den til å vise ultralydbilder.

Tips:
Har du flere maskiner tilgjengelig bør du bruke den kraftigste til denne testen. 3D rendering er en kostbar prosess og opplevelsen avhenger av hardware-spesifikasjoner.

Prototypen vil fungere i alle oppdaterte versjoner av de store nettleserne, men ytelsesevnen kan variere. Google Chrome eller Opera er å foretrekke. Mozilla Firefox bør også fungere greit. I Internet Explorer og Safari kan ytelsen bli svekket betraktelig.

Hvis det ikke skjer noe etter at du har trykket på en gjenstand det skal gå an å interagere, stå i ro og prøv å trykke en gang til.

Skulle en video ikke spille av prøv å vend synsvinkelen så du ser ut av vinduene et par sekunder (Dette problemet er observert i Chrome for Windows)

Start testen:
Trykk på denne lenken for å starte applikasjonen

Etter test:
Gå til neste side og besvar 7 spørmål knyttet t

Neste

Figure B.1: Questionnaire page 1

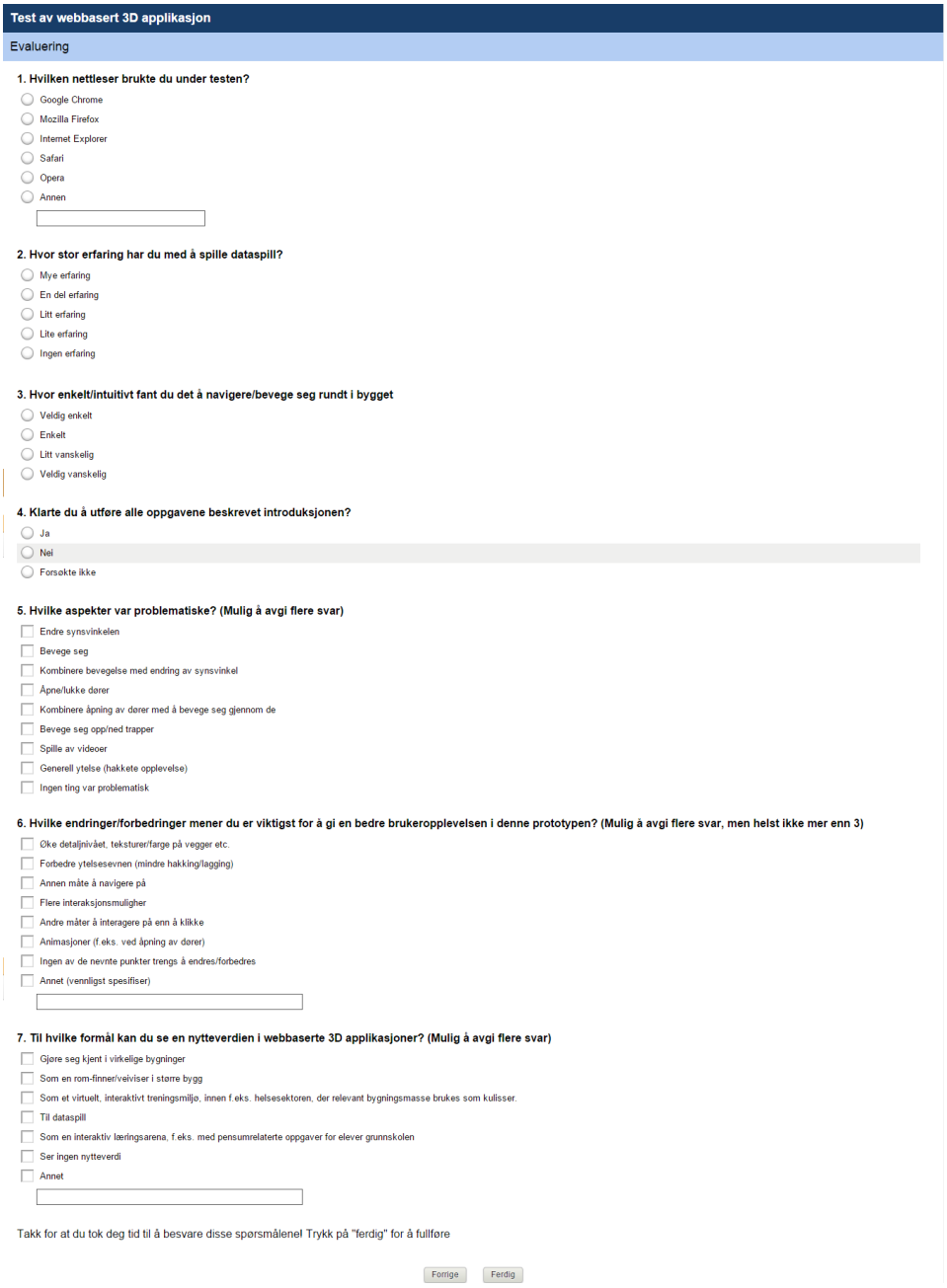


Figure B.2: Questionnaire page 2

Appendix C

Source code

The HTML and CSS source code is given in Listing C.1, while Listing C.2 shows the JavaScript code for the prototype

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>St. Olavs 1930</title>

    <style>
    html, body {
        overflow: hidden;
        width: 100%;
        height: 100%;
        margin: 0;
        padding: 0;
    }

    #babylonCanvas {
      width: 100%;
        height: 100%;
        touch-action: none;
    }

    #fps{
      position: absolute;
        right: 20px;
        top: 5em;
        font-size: 20px;
        color: white;
        text-shadow: 2px 2px 0 black;
    }
    #infoWindow{
      display: none;
          position: absolute;
```

```
33            top: 5%;
34            right: 5%;
35            width: 35%;
36            height: auto;
37            padding: 16px;
38            border: 2px solid black;
39            background-color: white;
40            z-index:1002;
41            overflow: auto;
42      }
43      #closebtn{
44        position:absolute;
45        right:0%;
46        top:0%;
47        width: 25px;
48        height: 25px;
49        padding: 0;
50        margin: 0;
51        border: 0;
52      }
53
54      </style>
55
56      <script src="http://www.babylonjs.com/babylon.js"></script>
57      <script src="http://www.babylonjs.com/cannon.js"></script>
58      <script src="http://www.babylonjs.com/oimo.js"></script>
59      <script src="http://www.babylonjs.com/hand.minified-1.2.js"></script>
60      <script src="index.js"></script>
61    </head>
62    <body>
63      <div id="babylon">
64        <canvas id="babylonCanvas"></canvas>
65        <div id="fps">50 fps</div>
66        <div id="infoWindow">
67          <div id="infoWindowImg"></div>
68          <div id="artText"></div>
69          <button type="submit" id="closebtn" onclick="document.
                getElementById('infoWindow').style.display='none';"><img src="
                closeicon.png"/></button>
70        </div>
71      </div>
72      <script>
73      var container = document.getElementById('babylonCanvas');
74      var engine, scene, camera, octree;
75      var artData, videoList, usVideoList, currentVideo;
76      var fpsTest, counter;
77      var testingMode=false;
78      var debug=false;
79
80      document.addEventListener("keydown", keydown, false);
81      createScene();
82      </script>
83    </body>
84  </html>
```

Listing C.1: HTML source code

```
1  //Jump animation
2  var cameraJump = function() {
3    var cam = scene.cameras[0];
4
5    cam.animations = [];
6
7    var a = new BABYLON.Animation(
8        "a",
9        "position.y", 20,
10       BABYLON.Animation.ANIMATIONTYPE_FLOAT,
11       BABYLON.Animation.ANIMATIONLOOPMODE_CYCLE);
12
13   // Animation keys
14   var keys = [];
15   keys.push({ frame: 0, value: cam.position.y });
16   keys.push({ frame: 10, value: cam.position.y + 1.2 });
17   keys.push({ frame: 20, value: cam.position.y});
18   a.setKeys(keys);
19
20   var easingFunction = new BABYLON.CircleEase();
21   easingFunction.setEasingMode(BABYLON.EasingFunction.EASINGMODE_EASEINOUT
         );
22   a.setEasingFunction(easingFunction);
23
24   cam.animations.push(a);
25
26   scene.beginAnimation(cam, 0, 20, false);
27 }
28
29 //Toggle running mode
30 function toggleRun(){
31   if(camera.speed==0.2)
32     camera.speed=0.4;
33   else
34     camera.speed=0.2;
35 }
36
37 //Assiging click actions that makes mesh invisible/visible
38 function activateDoorActions(mesh){
39   mesh.actionManager = new BABYLON.ActionManager(scene);
40   var child = [new BABYLON.SetValueAction(BABYLON.ActionManager.
         NothingTrigger, mesh, "checkCollisions", 0),
41   new BABYLON.SetValueAction(BABYLON.ActionManager.OnPickTrigger, mesh, "
         visibility", 0)];
42   var child2 = [new BABYLON.SetValueAction(BABYLON.ActionManager.
         NothingTrigger, mesh, "checkCollisions", 1),
43   new BABYLON.SetValueAction(BABYLON.ActionManager.OnPickTrigger, mesh, "
         visibility", 1)];
44   var action = new BABYLON.CombineAction(BABYLON.ActionManager.
         OnPickTrigger, child);
45   var action2 = new BABYLON.CombineAction(BABYLON.ActionManager.
         OnPickTrigger, child2);
46   mesh.actionManager.registerAction(action).then(action2);
47 }
48
49 //Assigning click actions that displays information about about a mesh
50 function activateInfoObject(mesh){
```

```
51    mesh.actionManager = new BABYLON.ActionManager(scene);
52    var action = new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.
          OnPickTrigger, function(){
53      var i = parseInt(mesh.name.substring(8,10));
54      document.getElementById("infoWindowImg").innerHTML='<img src="scenes/
            painting'+i+'.jpg" alt="fail" style="widtg:auto;max-width:100%";
            height:auto;>';
55      document.getElementById("artText").innerHTML='<br><font size="5"><b>
            Tittel:</b> ' + artData[i][0] + '<br><b>Kunstner:</b> ' + artData[
            i][1] +'<br><b>År:</b> '
56      + artData[i][2] + '<br><b>Maling:</b> ' + artData[i][3] + '</font>';
57      document.getElementById("infoWindow").style.display='block';
58    });
59    mesh.actionManager.registerAction(action);
60
61  }
62
63  //Applying VideoTexture to a mesh
64  function setUpVideo(mesh, videos, current){
65    mesh.material = new BABYLON.StandardMaterial("textVid", scene);
66    mesh.material.diffuseTexture = new BABYLON.VideoTexture("video", videos[
          current], scene);
67    mesh.material.emissiveColor = new BABYLON.Color3(1,1,1);
68  }
69
70  //Assigning click action to play/pause video
71  function activateVideoTexture(mesh, videos, type){
72    mesh.actionManager = new BABYLON.ActionManager(scene);
73    var action = new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.
          OnPickTrigger, function(){
74      if(currentVideo[type]==-1){
75        currentVideo[type]=0;
76        setUpVideo(mesh, videos, currentVideo[type]);
77      }
78      else{
79        mesh.material.diffuseTexture.video.play();
80      }
81    });
82    var action2 = new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.
          OnPickTrigger, function(){
83      mesh.material.diffuseTexture.video.pause();
84    });
85    mesh.actionManager.registerAction(action).then(action2);
86  }
87
88  //Assigning click actions for changing video
89  function previousVideo(mesh, screen, videos, type){
90    mesh.actionManager = new BABYLON.ActionManager(scene);
91    var action = new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.
          OnPickTrigger, function(){
92      if(currentVideo[type]===-1)
93        return;
94      if(currentVideo[type]<=0)
95        currentVideo[type]=videos.length-1;
96      else
97        currentVideo[type]--;
98      screen.material.diffuseTexture.video.pause();
```

```
99      setUpVideo(screen, videos, currentVideo[type]);
100   });
101   mesh.actionManager.registerAction(action);
102 }
103 function nextVideo(mesh, screen, videos, type){
104   mesh.actionManager = new BABYLON.ActionManager(scene);
105   var action = new BABYLON.ExecuteCodeAction(BABYLON.ActionManager.
          OnPickTrigger, function(){
106     if(currentVideo[type]===-1)
107       return;
108     if(currentVideo[type]>=videos.length-1)
109       currentVideo[type]=0;
110     else
111       currentVideo[type]++;
112     screen.material.diffuseTexture.video.pause();
113     setUpVideo(screen, videos, currentVideo[type]);
114   });
115   mesh.actionManager.registerAction(action);
116 }
117
118 //Setting up navigation puttons
119 function addNavigationButtons(){
120     scene.activeCamera.keysUp.push(87); // W
121     scene.activeCamera.keysDown.push(83); // S
122     scene.activeCamera.keysLeft.push(65); // A
123     scene.activeCamera.keysRight.push(68); // D
124 }
125
126 //Setting up test program
127 function runTestProgram(){
128   if(!testingMode){
129     fpsTest=0;
130     counter=0;
131     testingMode=true;
132     console.log("Test started")
133   }
134 }
135
136 function conductTest(){
137   fpsTest+=engine.getFps();
138   counter++;
139   if(counter===800){
140     testingMode=false;
141     alert(fpsTest/counter);
142   }
143 }
144
145 //Custom Scene Optimizer
146 function customOptimizer() {
147 var result = new BABYLON.SceneOptimizerOptions(40, 3000);
148 result.optimizations.push(new BABYLON.TextureOptimization(0, 256));
149 result.optimizations.push(new BABYLON.PostProcessesOptimization(1));
150 result.optimizations.push(new BABYLON.LensFlaresOptimization(2));
151 result.optimizations.push(new BABYLON.ShadowsOptimization(3));
152 result.optimizations.push(new BABYLON.RenderTargetsOptimization(4));
153 result.optimizations.push(new BABYLON.ParticlesOptimization(5));
154 result.optimizations.push(new BABYLON.RenderTargetsOptimization(6));
```

```
155 result.optimizations.push(new BABYLON.HardwareScalingOptimization(7, 4));
156 return result;
157 }
158
159 //Applying Scene Optimizer
160 function optimizeScene(){
161   BABYLON.SceneOptimizer.OptimizeAsync(scene, customOptimizer());
162   console.log("optimized");
163 }
164
165 //Show/hide debug layer
166 function toggleDebugLayer(){
167   if(!debug){
168     scene.debugLayer.show();
169     debug=true;
170   }
171   else{
172     scene.debugLayer.hide();
173     debug=false;
174   }
175 }
176
177 //Setting up scene
178 function createScene(){
179   engine = new BABYLON.Engine(container, true);
180   scene = new BABYLON.Scene(engine);
181
182   //Arrays holding art informations and video urls
183   artData = [["Die Antitheoretische Schutzmauer", "Hilde Vemren", "", ""],
        ["Winter Sparkle", "Megan Duncanson", "2012", "Akryl på lerret"], [
      "Rœd Gul Blå 3", "Martin Fasting", "2010", "Akryl på lerret"]];
184   videoList = [["media/01_Tumor-Glioblastoma.mp4"], ["media/02_Tumor-
      Glioblastoma_USOnly.mp4"], ["media/03_Tumor-Glioblastoma_Biopsy1.mp4
      "], ["media/04_Tumor_LGG_Biopsy2.mp4"]];
185   usVideoList = [["media/ultrasoundsample.mp4"], ["media/ultrasoundsample2
      .mp4"], ["media/ultrasoundsample3.mp4"]];
186   currentVideo = [-1, -1];
187
188   //Importing ARK model
189   BABYLON.SceneLoader.ImportMesh("", "scenes/", "ark2.babylon", scene,
        function (newMeshes, particleSystems, skeletons)
190   {
191     octree = scene.createOrUpdateSelectionOctree();
192     var mName;
193     for(var i=0; i<newMeshes.length; i++){
194       //newMeshes[i].createOrUpdateSubmeshesOctree();
195       //newMeshes[i].useOctreeForCollisions(octree);
196       mName=newMeshes[i].name;
197
198       //Assigning collision checks and actions to subgroups of meshes
199       if((mName.substring(0,4)==="EKS." || mName.substring(0,3)==="Dek" ||
            mName.substring(0,2)==="IV") || mName.substring(0,5)==="Trapp"
200         || mName==="Elevator wall" || mName==="WallBarrier" || mName==="
              Ground" || mName==="ultrasoundBoundingBox" || mName==="Ekran"
201         || mName==="Tak-.001" || mName.substring(0,5)==="DI040" || mName
              ==="V-.006" || mName==="V-.007" || mName==="V-.013"){
202         newMeshes[i].checkCollisions=true;
```

```
203       }
204       else if(mName.substring(0,2)==="DB" || mName.substring(0,4)==="EKS_"
              || mName.substring(0,2)==="DI" || mName==="DU001"){
205         newMeshes[i].checkCollisions=true;
206         activateDoorActions(newMeshes[i]);
207       }
208       else if(mName.substring(0,8)==="backFace")
209         activateInfoObject(newMeshes[i]);
210     }
211
212     //Assigning actions to screens and buttons
213     var screen1 = scene.getMeshByName("Lerret");
214     screen1.setVerticesData("uv", [1,0,1,1,0,1,0,0], true);
215     activateVideoTexture(screen1, videoList, 0);
216     var screen2 = scene.getMeshByName("Ultrasoundscreen");
217     screen2.setVerticesData("uv", [0,1,0,0,1,0,1,1], true);
218     activateVideoTexture(screen2, usVideoList, 1);
219     var button1 = scene.getMeshByName("screenButton1");
220     previousVideo(button1, screen1, videoList, 0);
221     var button2 = scene.getMeshByName("screenButton2");
222     nextVideo(button2, screen1, videoList, 0);
223     var button3 = scene.getMeshByName("usButton1");
224     previousVideo(button3, screen2, usVideoList, 1);
225     var button4 = scene.getMeshByName("usButton2");
226     nextVideo(button4, screen2, usVideoList, 1);
227
228   });
229
230   //Import RIE model
231   BABYLON.SceneLoader.ImportMesh("", "scenes/", "rie2.babylon", scene,
          function (newMeshes, particleSystems, skeletons)
232   {
233     //Update octree
234     octree = scene.createOrUpdateSelectionOctree();
235   });
236
237   //Creating skybox
238   var skybox = BABYLON.Mesh.CreateBox("skyBox", 150.0, scene);
239   var skyboxMaterial = new BABYLON.StandardMaterial("skyBox", scene);
240   skyboxMaterial.backFaceCulling = false;
241   skybox.material = skyboxMaterial;
242   skyboxMaterial.diffuseColor = new BABYLON.Color3(0, 0, 0);
243   skyboxMaterial.specularColor = new BABYLON.Color3(0, 0, 0);
244   skyboxMaterial.reflectionTexture = new BABYLON.CubeTexture("cubemap/
          skybox", scene);
245   skyboxMaterial.reflectionTexture.coordinatesMode = BABYLON.Texture.
          SKYBOX_MODE;
246   skybox.infiniteDistance = true;
247
248   //Setting up camera
249   var startingPos =  new BABYLON.Vector3(-6, 2.5, -6);
250   if( /Android|webOS|iPhone|iPad|iPod|BlackBerry|IEMobile|Opera Mini/i.
          test(navigator.userAgent) ) {
251     camera = new BABYLON.TouchCamera("TouchCamera", startingPos, scene);
252   }
253   else{
254     camera = new BABYLON.FreeCamera("FreeCamera", startingPos, scene);
```

```
  }

  camera.speed=0.2;
  camera.setTarget(new BABYLON.Vector3(-8, 2.5, 0));
  camera.ellipsoid.y=1.1;
  camera.ellipsoid.z=0.6;
  camera.ellipsoid.x=0.6;
  camera.checkCollisions = true;
  scene.activeCamera.attachControl(container);
  //scene.enablePhysics(null, new BABYLON.OimoJSPlugin());

  //Defining gravitational force
  scene.gravity = new BABYLON.Vector3(0, -0.01, 0);
  camera.applyGravity = true;


  //Setting up light
  var light = new BABYLON.HemisphericLight("Hemi0", new BABYLON.Vector3(0,
       1, 0), scene);
  light.diffuse = new BABYLON.Color3(1, 1, 1);
  light.specular = new BABYLON.Color3(1, 1, 1);
  light.groundColor = new BABYLON.Color3(0, 0, 0);

  addNavigationButtons();
  var fps = document.getElementById("fps");

  //Render loop
  engine.runRenderLoop(function() {
    fps.innerHTML=engine.getFps().toFixed() + " fps";
    /*if(testingMode){
      conductTest();
    }*/
    scene.render();
  });

  //Resize engine
  window.addEventListener("resize", function () { engine.resize(); });
}

//Handle input from keyboard
function keydown(e){
  if(e.keyCode==32)
    cameraJump();
  else if(e.shiftKey)
    toggleRun();
  else if(e.keyCode==70)
    engine.switchFullscreen(true);
  else if(e.keyCode==84)
    runTestProgram();
  else if(e.keyCode==79)
    optimizeScene();
  else if(e.keyCode==76)
    toggleDebugLayer();
}
```

Listing C.2: JavaScript source code