NTNU – Trondheim
Norwegian University of
Science and Technology

VR-Modelling of Data from "Norway Digital" in Combination with other 3D Models

Stig Henning Fredheim

Master of Science in Engineering and ICT
Submission date: June 2014
Supervisor: Terje Midtbø, BAT
Co-supervisor: Alexander S. Nossum, Norkart

Norwegian University of Science and Technology
Department of Civil and Transport Engineering

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY
DEPARTMENT OF CIVIL AND TRANSPORT ENGINEERING

Report Title: VR-Modelling of Data from "Norway Digital" in Combination with other 3D Models	Date: 10.06.14 Number of pages (incl. appendices): 66			
	Master Thesis	X	Project Work	
Name: Stig Henning Fredheim				
Professor in charge/supervisor: Terje Midtbø				
Other external professional contacts/supervisors: Alexander Salveson Nossum				

Abstract:

The purpose with this master thesis was to use new released data from the Norwegian Mapping Authority to make a virtual world that can be looked at with or without Oculus Rift. In addition, existing 3D models of buildings created in SketchUp were used. Oculus Rift is a new device within virtual reality that have received much attention. Gløshaugen, the campus for the technology students at NTNU, were used as a case.

The virtual world runs in a modern web browser, which makes it easily available. Three.js is the chosen graphics library. The program contains a mode with and without Oculus Rift. The report evaluates the performance and concludes that the frame per seconds is a bit low because of the textured buildings. The performance is worst in the mode with Oculus Rift.

The visual result should be good enough as a showcase, but not to show a very accurate model of the real world. The user can navigate through the model by using the keyboard or be guided by a flythrough.

The literature chapter contains some relevant information to this thesis, for instance virtual reality. It also includes an evaluation other frameworks for making virtual worlds.

Keywords:

1. Virtual Reality
2. Oculus Rift
3. Norway Digital
4. Gløshaugen

_____

Master thesis

(TBA4925 - Geomatikk, masteroppgave)

Spring 2014

for

Stig Henning Fredheim

Integration of national geographical information and 3D building models in Virtual Reality (VR)

- Case: making VR model over Gløshaugen

BACKGROUND

In 2012/2013 a new product for presenting virtual reality (VR) to the mass marked emerged. The product is named Oculus Rift, and is primarily intended for the gaming industry. However, the equipment which has an open programming interface, has showed potential for 3D visualization in different areas.

At their meeting with the university the first year students at Programme for Engineering and ICT participate in an introductory course named “Teknostart”. One main objective of the project part in the course is to make 3D models of building at the university campus and close by areas. Over the years a library of different buildings has been generated.

The Norwegian Mapping authorities has held leading roles in the work on establishing international standards for geographic information, and Norway is in front when it comes to organization and standardization of geographic information. This is realized through the geoportal “Norge digital” (Norway digital). In 2013 a considerable part of these data were made free available.

The main objective of this thesis is to investigate how 3D models made by students in Sketchup can be combined with data from “Norge digital” in a VR model for Oculus Rift.

TASK DESCRIPTION

Specific tasks:

- Study relevant literature related to VR and geographic information
- Look into other areas where Oculus Rift may be used in addition to gaming.
- Investigate and choose framework for construction of a VR model
- Make a VR model based on terrain model and ortophoto from “Norge digital” together with 3D building models made in Sketchup
- Evaluate the efficiency of the developed VR model

ADMINISTRATIVE/GUDIANCE

The work on the Master Thesis starts on January 11th, 2014

The thesis report as described above shall be submitted digitally in DAIM at the latest at June 11, 2014

Supervisors at NTNU and professor in charge:

Terje Midtbø

External supervisor:

Alexander Nossum, Norkart

Trondheim, January 10, 2014. (revised: 06.06.2014)

Preface

I would to thank my main supervisor Terje Midtbø for guidance through the semester. I would also like to thank Alexander Salveson Nossum, who as my external supervisor have been a great help. In addition, I have appreciated the fellowship with the other students from Geomatics. Without them, this experience would have been much lonelier.

Trondheim, June 10, 2014

Stig Henning Fredheim

Abstract

The purpose with this master thesis was to use new released data from the Norwegian Mapping Authority to make a virtual world that can be looked at with or without Oculus Rift. In addition, existing 3D models of buildings created in SketchUp were used. Oculus Rift is a new device within virtual reality that have received much attention. Gløshaugen, the campus for the technology students at NTNU, were used as a case.

The virtual world runs in a modern web browser, which makes it easily available. Three.js is the chosen graphics library. The program contains a mode with and without Oculus Rift. The report evaluates the performance and concludes that the frame per seconds is a bit low because of the textured buildings. The performance is worst in the mode with Oculus Rift.

The visual result should be good enough as a showcase, but not to show a very accurate model of the real world. The user can navigate through the model by using the keyboard or be guided by a flythrough.

The literature chapter contains some relevant information to this thesis, for instance virtual reality. It also includes an evaluation other frameworks for making virtual worlds.

Sammendrag

Formålet med denne masteroppgaven var å bruke nylig frisluppede data fra Kartverket til å lage en virtuell verden som kan bli sett på med og uten Oculus Rift. I tillegg ble eksisterende 3D modeller laget i SketchUp brukt. Oculus Rift er et nytt apparat innenfor virtuell virkelighet som har fått mye oppmerksomhet. Gløshaugen, universitetsområdet for teknologistudentene til NTNU, ble brukt som case.

Den virtuelle verdenen kjøres i en moderne nettleser, noe som gjør den lett tilgjengelig. Three.js er det valgte grafikkbiblioteket. Programmet består av modus med og uten Oculus Rift. Rapporten evaluerer ytelsen og kommer fram til at den er litt lav på grunn av alle teksturene til bygningene. Den er verst i modusen med Oculus Rift.

Det visuelle resultatet burde være godt nok for en «showcase», men ikke til å vise en veldig nøyaktig modell av den virkelige verden. Brukeren kan navigere i den virtuelle verdenen ved å bruke tastaturet eller ved å bli ført gjennom et tidligere opptak.

Litteraturkapittelet inneholder relevant informasjon, for eksempel litt om virtuell virkelighet. I tillegg evalueres forskjellige rammeverk som kan brukes til å lage en virtuell verden.

Contents

Preface iv
Abstract v
Sammendrag vi
List of figures ix
List of tables x
List of acronyms x
1 Introduction 1
2 Literature 2
2.1 Virtual reality 2
2.1.1 History 2
2.1.2 Applications 3
2.2 Oculus Rift 3
2.2.1 Specifications(First developer version)(20) 4
2.2.2 Sensors 4
2.2.3 Stereo vision 5
2.2.4 Applications 5
2.2.5 Additional devices 6
2.2.6 Issues 6
2.2.7 Competitors 6
2.3 Modelling or reconstruction of buildings 7
2.4 Norway digital 7
2.5 Relevant programs, technologies and formats 8
2.5.1 SketchUp 8
2.5.2 Blender 8
2.5.3 DEM (USGS DEM) 8
2.5.4 WebGL (Web Graphics Language) 8
2.5.5 WMS (Web Map Service) 9
2.5.6 WCS (Web Coverage Service) 9
2.6 Framework options within 3D graphics 10
2.6.1 OpenSceneGraph (OSG) 13
2.6.2 Three.js 14
2.6.3 JMonkeyEngine 17
2.6.4 Conclusion: 18
3 Implementation 19
3.1 Implementation description 19

3.1.1 Overview 19
3.1.2 Terrain 21
3.1.3 Buildings 22
3.1.4 Collision 23
3.1.5 Flythrough: 24
3.1.6 Performance measurements 24
3.1.7 Various details 24
3.1.8 External libraries: 25
3.2 Preprocessing of data 25
3.2.1 Terrain 25
3.2.2 Buildings 26
4 Results 29
4.1 Performance 29
4.1.1 Start-up times 30
4.1.2 Frame per seconds 32
4.1.3 Delay in the oculus rift: 38
4.2 Visual results 39
4.3 Navigation 41
5 Discussion 42
5.1 Performance 42
5.2 Visual results 42
5.3 Navigation 43
5.4 Questions 43
6 Conclusion 45
7 Further work 46
8 References 47
9 Appendices 54
9.1 Appendix A: Performance tables 54
9.2 Appendix B: Running locally 56

List of figures

Figure 1: Oculus Rift - Developer version (19) 4
Figure 2: Orientation Directions (20) 5
Figure 3: Stereo example from OculusWorldDemo (20) 5
Figure 4: Virtuix Omni (24) 6
Figure 5: Example on a WMS request and the corresponding result 9
Figure 6: Example on a WCS request and a result with color adjustments 10
Figure 7: A rendering of "Main administration building" in OSG 13
Figure 8: OSG examples with terrain 13
Figure 9: OpenSceneGraph with Oculus Rift integration 14
Figure 10: Rendering of a Collada file in three.js 15
Figure 11: Repeat vs clamping 15
Figure 12: Rendering of a OBJ file in THREE.js 15
Figure 13: Rendering of terrain in THREE.js(70) 16
Figure 14: Textured cube rendered in three.js in "Oculus mode" 16
Figure 15: Terrain example in jMonkeyEngine 17
Figure 16: Rendering in stereo with jMonkeyEngine 18
Figure 17: Google trends comparison between OSG, three.js and jMonkeyEngine(79) 18
Figure 18: System overview 19
Figure 19: Diagram describing flow and hierarchy 20
Figure 20: XY plane of terrain data from the XYZ file 22
Figure 21: Pseudo code - Find resolution of X 22
Figure 22: Load building 23
Figure 23: Icosahedron and axis in three.js 24
Figure 24: List of used external libraries 25
Figure 25: Preprocessing of terrain data in batch file with GDAL library 26
Figure 26: Preprocessing of buildings 27
Figure 27: Start-up times from configuration 1 31
Figure 28: Start-up times from configuration 2 31
Figure 29: Rendering of state 1 32
Figure 30: Results from state 1 without Oculus Rift 33
Figure 31: Rendering of state 2 34
Figure 32: Results from state 2 without Oculus Rift 35
Figure 33: Flythrough without Oculus Rift 36
Figure 34: States comparison 36
Figure 35: State 1 with or without Oculus Rift 38
Figure 36: State 2 with or without Oculus Rift 38
Figure 37: Flythrough with or without Oculus Rift 38
Figure 38: Gløshaugen seen from the air 39
Figure 39: Building seen from the ground 39
Figure 40: Unexpected wall texture 40
Figure 41: Gløshaugen seen from Oculus Rift 40
Figure 42: Comparison of texture resolutions. The original is at the left, while the lower resolution(1/4 in width and height) is at the right 40
Figure 43: Rendering without textures 41

List of tables

Table 1: Framework overview 12
Table 2: Computer specifications 29
Table 3: Startup times 30
Table 4: FPS in state 1 without Oculus Rift 33
Table 5: Extra measurements of FPS in state 1 33
Table 6: FPS in state 2 without Oculus Rift 34
Table 7: Extra measurements in state 2 34
Table 8: FPS for flythrough without Oculus Rift 35
Table 9: Extra measurements from flythrough 36
Table 10: FPS in state 1 with Oculus Rift 37
Table 11: FPS in state 2 with Oculus Rift 37
Table 12: FPS for flythrough with Oculus Rift 37

List of acronyms

DOM	Document Object Model
FKB	Felles Kartdatabase
FPS	Frame per seconds
GDAL	Geospatial Data Abstraction Library
HMD	Head mounted display
NMA	Norwegian Mapping Authority
OSG	OpenSceneGraph
SDK	Software development kit
UDK	Unreal Development Kit
URL	Uniform Resource Locator
UTM	Universal Transverse Mercator
WCS	Web Coverage Service
WMS	Web Map Service

1 Introduction

In 2012, a cheap tool for using virtual reality became available. The product is Oculus Rift and its main use is within the game industry. However, since it has an open Application Programming Interface (API) it is also possible to use it for 3D visualization in other fields, e.g. geomatics.

Through the introductory projects in the study Engineering and ICT at NTNU, students have made 3D-models of buildings at Gløshaugen, the campus for technology students at NTNU, and some important buildings in Trondheim. They created the models in the 3D editing program, SketchUp. In addition to this, Norwegian Mapping Authority (NMA) has recently released digitally terrain models with a resolution in both 10x10 and 50x50 meters over the whole country, which now are free to use.

The goal for this project is to look at visualization in 3D of terrain and buildings with Oculus Rift. This includes investigating different software alternatives, and using one of those to visualize an area. This project uses Gløshaugen as a case. The solution is evaluated based on performance, visual appeal, navigation. This report also contains background information about virtual reality, including Oculus Rift, and relevant technologies.

In addition, the report is looking into the following questions:

- How can a combination of terrain and 3D-models be visualized in a semi-automatic procedure by using open source software?

- Which interactions methods suits the navigation in this kind of virtual reality?

- How can other types of geographical data expand the proposed virtual world?

2 Literature

2.1 Virtual reality

Virtual reality have existed for several decades. Cruz-Neira has defined it as: “Virtual reality refers to immersive, interactive, multi-sensory, viewer-centered, three-dimensional computer generated environments and the combination of technologies required to build these environments(1)”. However, there are numerous of definitions, and different virtual reality systems will fit a definition in varying degree. Augmented reality is similar to virtual reality, but it adds information to the real world, e.g. it can add fly information to a fighter plane pilot’s view.

Head mounted displays (HMD) are on type of device used in VR. Other types include virtual reality rooms where projectors cover each wall, input devices like the controller in Nintendo Wii, and computer vision devices like the Microsoft Kinect(2).

Tracking of the head is a crucial part to achieve an immersive feeling. The interaction possibilities increases if other body parts like the hands also is tracked. The tracker can be magnetic, acoustic, optical or mechanical(3). There are several options for how to map user input to movement in the virtual world. Using the keyboard is perhaps the simplest solution. Bowman, Johnson and Hodges mentions moving by pointing, gazing in a direction, moving the torso, manipulating of an object and map dragging. They found gaze-directed steering and pointing to be among the most effective for travelling(4).

In addition to speed, other factors like naturalness should not be ignored. Usoh et al. compared walking, walking in place and flying. They concluded that walking was better than virtual walking and flying regarding simplicity, straightforwardness and naturalness. They also mentioned the gain in the feeling of presence by having a realistic avatar(5).

If interaction with objects is one of the goals with a virtual environment, then it could be advantageous to move without using the hands. Instead an operator can move by leaning in a direction(6).

Wayfinding in a big virtual world can be challenging, just like in the real world. This can be a problem depending on the usage. Darken observed that subjects improves their performance in wayfinding tasks when receiving environmental cues. The cues included map and organizational principles. Without these, the subjects used ineffective search strategies and experienced frequent disorientation(7).

2.1.1 History

Sensorama(8) is one of the earliest examples of virtual reality, and it takes the user on a journey through Brooklyn. Sensorama provides audio, haptic and olfactory stimuli in addition to the stereo vision, but the user have no interaction possibilities. Morton Heilig created the device in 1962. The device did not reach mass production since Heilig failed to get financial support(2).

“Sword of Damocles” created by Ivan Sutherland in 1968 was another development. It was a head mounted display hanging from the roof, which provided stereo vision and could track the position of the users head and eyes(2).

In addition to expensive prototypes, commercial actors have attempted to make products for the average user.

Nintendo released Virtual Boy in 1995. It was a stereoscopic game console, but lacked head tracking. Poor display, with only red images, and negative effects such as motion sickness and eyestrain, were some reason for the lack of success(9,10).

Sony released the head mounted display, Glasstron, in 1997. It has two small LCD screen in front of each eye that uses a video source, and ear buds for each ear. The device represented an alternative to watching at a television, so there was no 3D or head tracking(11,12).

2.1.2 Applications

Virtual reality has numerous of applications. It can present every visual thing imaginable, even things not existing on earth, in real 3D through for example a head mounted display.

A gaming experience can become more immersive and more fun. Playing games can also become healthier if body movements is transferred to the game.

When deciding if a building should be build, the immersive experience that the virtual reality may provide will make it easier to understand how it would look. The experience can also give ideas for changes in the design.

There are many possibilities with in education, e.g. flight simulators, hazardous tasks or practicing within medical procedures. Virtual reality can also be used to control robots located in hostile environments(3).

A completely different and perhaps unexpected application is within pain and fear reduction. Reduced pain in medical care while using VR is one example(13). Another study(14), found virtual reality to be effective in treatment of spider phobia.

2.2 Oculus Rift

Oculus Rift, also called the Rift, is a new product within virtual reality. This head mounted display gives the user the ability to see a virtual world in 3D. The device also gives sensor data, which means that the movement of the head is also affecting the “virtual head”. The Oculus Rift communicates with a computer through cables. As a result, there are some movement restrictions for the user.

Oculus VR, the company behind the Oculus Rift, used the crowd-founding site Kickstarter(15) in late 2012 for financial support. The response was huge and close to 10 000 people gave their support with a total of more than 2 million dollars (16). The affordable price is one of the main reasons for the big support. Later, in March this year, Facebook bought the company for $2 billion dollars(17).

The first version of the developer version, displayed in Figure 1, is used in this thesis. Oculus VR have scheduled a newer developer version for release in July 2014. Two important improvements are within resolution and latency. It is unknown when the consumer version will be ready. Oculus VR are also working with supporting the mobile operating system Android in an effort to make it more mobile(18)

Figure 1: Oculus Rift - Developer version (19)

2.2.1 Specifications(First developer version)(20)

Display:

Display area: 7 inches
Resolution: 1280x800 in total, 640x800 per eye.
Fixed lens distance: 64mm
LCD Panel: 60Hz

Tracker:

Up to 1000Hz in sampling rate
Gyroscope: three-axis
Magnetometer: three-axis
Accelerometer: three-axis

2.2.2 Sensors

As listed in the specifications, the oculus rift have a gyroscope, magnetometer and accelerometer. The orientation of the users head is determined by using sensor fusion. It is stored as quaternions internally, but the API can also present yaw-pitch-roll like illustrated in Figure 2. The orientation will drift when only using the gyroscope since it only measures the change in orientation. The magnetometer improves the yaw, and the accelerometer improves the pitch and roll. The SDK documentation(20) provides a more thorough explanation.

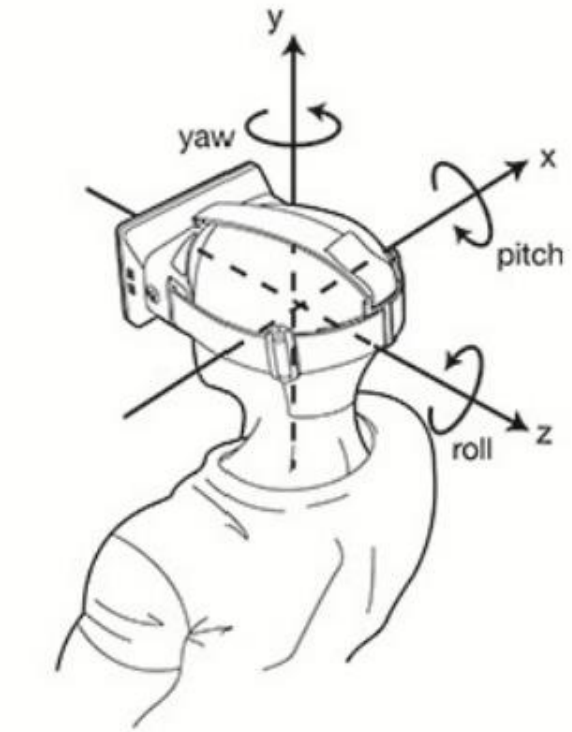


Figure 2: Orientation Directions (20)

2.2.3 Stereo vision

Oculus rift achieves stereo vision by displaying the left half of the screen to the left eye and vice versa. The lens in the Rift magnifies the image, in order to achieve a larger field of view. Unfortunately, this process also distorts the image. This can be counteracted by applying a barrel distortion on the image displayed on the screen(20). Figure 3 displays how the computer screen displays the distortion.

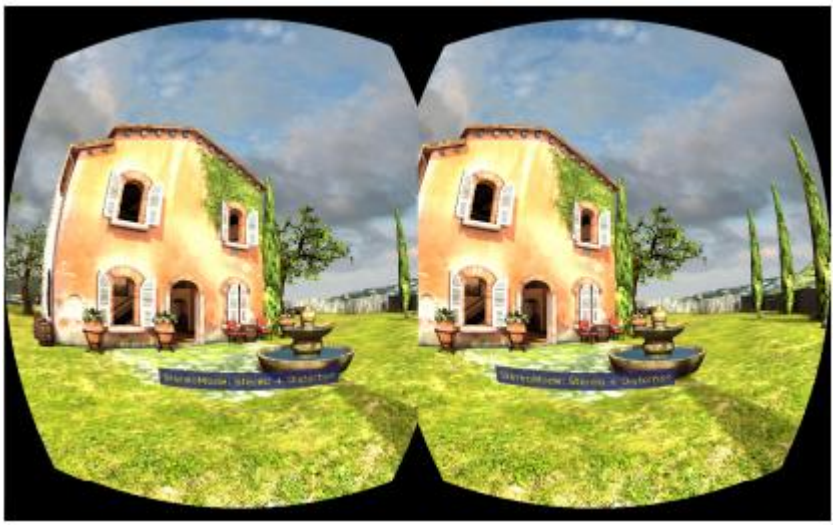


Figure 3: Stereo example from OculusWorldDemo (20)

2.2.4 Applications

The main application for the Rift is gaming. The goal is to make the gaming experience as real as possible. The Rift cannot make all the aspects feel realistic, like moving and hearing, but it helps with

the visual parts. The fact that many games already have implemented or is developing support for the device(21,22), says something about the confidence of what the Oculus Rift have and will achieve. The applications listed in Chapter 2.1.2 is also relevant for the Rift.

2.2.5 Additional devices

Even though Oculus Rift is a great addition for better immersion, it is not a complete tool. For instance, the Rift does not make you able to walk. A user can use the keyboard, but there are other products, that combined with Oculus Rift makes the experience even more immersive. Some are readily available today, while others are in the developing face. Two of those are mentioned below.

Virtuix Omni(23), displayed in Figure 4, is a treadmill that claims to make the walking in a virtual environment much more natural than using a keyboard or joystick. Like Oculus Rift, people have supported them through Kickstarter(24) and the price is in the same affordable area. Unfortunately, as of late April, there is only a pre-order choice for estimated delivery in September. Despite of this, Virtuix Omni sounds promising.

MVN BIOMECH Awinda(25) goes even further. It contains a set of sensors placed on several places on the body. As a result, the whole body can by simulated in a virtual world. The company behind this product, Xsens, also have videos showing their product together with Oculus Rift. The result is impressing, but the price seem to be in a much less affordable level.

Figure 4: Virtuix Omni (24)

2.2.6 Issues

One issue for long time Oculus Rift use is VR Sickness. Some of the causes are high latency, wrong camera calibration, and conflict between what the eyes senses and what the body feels. The intensity of the VR Sickness are highly dependent on each user(26). To make the experience look more realistic, the resolution have be increased.

2.2.7 Competitors

Even though Oculus Rift was first with a development version to a consumer friendly price, the commercial version is not available yet. Other companies have begun with their own version. One of

them Sony, have a system called Project Morpheus. Unlike Oculus Rift, the system is made for PlayStation 4 (27).

2.3 Modelling or reconstruction of buildings

3D models of buildings can be made manually in a 3D modelling program or automatically by using different sources of available data. The manually created models have the potential to become very good, but the process is quite time consuming. Automatically generated buildings can be made much more efficient, but it is difficult to match to best results from the manual made buildings.

It is possible to generate buildings from images, laser scans or both and the data can be acquired from the air or the ground. Ground based surveying achieves a better level of detail, while aerial surveys cover larger areas(28).

There have been much activity and many accomplishments within the automated process. P. Musialski et al. presents an extensive overview of the research within urban reconstruction(28).

Different solutions comes with different scales, where some cover whole cities while other covers one building. Qi Shan et al. (29) presents a solution for creating a model of a landmark by using images at Flickr(30) and aerial images from Google. They added images from Google Street view in areas with bad coverage. This approach appear to work well on popular buildings, but not on whole cities.

Poullis C. and You S. have created a system that generates 3D models over a city-sized area using LIDAR data to make the building geometries and imagery data to add textures to the building. The generation of building geometries is automated, while the adding textures requires some interactivity. An operator needs to manually set some correspondence points between the images and the 3D models(31).

Even though the research have gone far, more is needed because of the huge amount of data combined with the demand for more accurate and detailed models (28).

Google earth is one commercial example of the accomplishments with in automated building generations. In addition to the user created models, it also contains large sets of automatically generating buildings in many large cities(32). Google's support site(33) present a list of these. The 3D buildings are also available in the new version of Google Maps.

2.4 Norway digital

Norway digital ("Norge digitalt" in Norwegian) is a cooperation between many providers and users of geographical data in Norway. These include municipalities, counties and government agencies. They use standards for data and services, which makes it easier to share. The Norwegian Mapping Authority (NMA) is the coordinator of Norway digital. They have also established a portal, Geonorge, for the data sets and services(34,35).

NMA have recently begun to make some of their data freely available for everyone to use. In the fall 2013, they the released the datasets with borders, road data including addresses, terrain models and N50, the main map over Norway in the scale 1:50 000(36).

In addition to the mentioned releases, more is planned to be released in 2015-2017. These includes orthophotos, sea data, data from the land register, and detailed mapping data from "Felles kartdatabase (FKB)" (37).

The motivation behind this is to increase the innovation. There is also a goal that official maps should be available in the same way as other official information. An analyze, commissioned by the Ministry of Local Government and Modernisation, states that gain of releasing mapping and real estate data will be at least 70 million NOK annually(38,39).

NMA provides a downloading site (40), but the downloading process is manual. Users have to register and then manually choose what to download. In addition, the choices of formats on the download is limited, but the site informs that more choices might be added in the future depending experiences and feedback.

The terrain model and orthophotos are most relevant for the program made in the thesis, but many of the other data sets represents opportunities for expanding the model that the program is visualizing.

2.5 Relevant programs, technologies and formats

This section contains a brief explanation of various programs, technologies and formats that have some relevance in this project.

2.5.1 SketchUp

SketchUp(41) is a program where people can make different models in 3D. SketchUp has a free version, as in free lunch, and is relatively easy to learn compared to more advanced programs. The free version can export its objects to Collada and the Google Earth format KMZ. The latter is a zip file containing a Collada file, the images and a KML-file with the georeferenced coordinates. KMZ-files of the buildings were already made, so they were used in this project.

As mentioned in the introduction, first year students from Engineering and ICT have made the models of the buildings. The simplicity of SketchUp and the integration with Google Earth are probably why it became the program of choice.

2.5.2 Blender

Blender(42) is a free and open source 3D graphics programs with many features. The program is more advanced than SketchUp and is at the same time harder to learn. Making 3D models is outside the scope of this project, but Blender's export features is relevant. In addition to the included exporters, others can be installed as a plugin. It is possible to use Blender in the programming language Python(43) in order to automate processes.

2.5.3 DEM (USGS DEM)

USGS DEM(44) is file format developed by the United States Geological Survey(45) and stores a raster based digital elevation model. Others countries than USA, also uses this format, e.g. NMA in the distribution of their free terrain data.

A DEM file is stored as text with no line breaks. Metadata is located at the top followed by height values. Because these files tend to be quite large, it might be quite impractically to read in a text editor.

Free software like the command line library GDAL(46) can be used to clip the file with given boundaries and to convert it to other formats. QGIS(47), a free and open source GIS, can use GDAL as a plugin, if the lack of a GUI is a problem.

2.5.4 WebGL (Web Graphics Language)

WebGL(48) is a standard for low-level 3D graphics API in JavaScript, and is based on OpenGL ES 2.0. It is cross-platform, supported by the most of the major web browser (except Internet Explorer), and

gives the opportunity to render graphics in 3D without any plugin. WebGL is dependent on the canvas element in HTML5, the newest version of the HTML standard, to be able to work. The standard is created and maintained by Khronos Group(49) in cooperation with many of the major browser companies.

2.5.5 WMS (Web Map Service)

Web Map Service (50,51) is a standard created by the Open Geospatial Consortium (52). It defines a method for requesting georeferenced maps from a server. A request is an URL consisting of the address to the WMS server and different parameters. The received map will usually be in image formats like PNG, GIF or JPEG. The server must support two operations, *GetCapabilities* and *GetMap*. The result from the first operation is metadata about the service, like the layers that is available and which formats that is supported. GetMap returns a map based on the parameters that is given. Figure 5 gives an example of a request URL and the response from Norwegian Mapping Authority's WMS server. The image displays a layer over the administrative border in the southern half of Norway. The coordinate reference system (CRS) and the bounding box (bbox) defines which area to request. The CRS value EPSG:32632 is another name for UTM32.

GetFeatureInfo is an example of an optional operation that the WMS servers may support. When using this operation, the server will give more information about a feature on specified pixel coordinates.

The WMS standard is makes it possible for clients to request images with overlapping areas from different sources, and then combine them to one image.

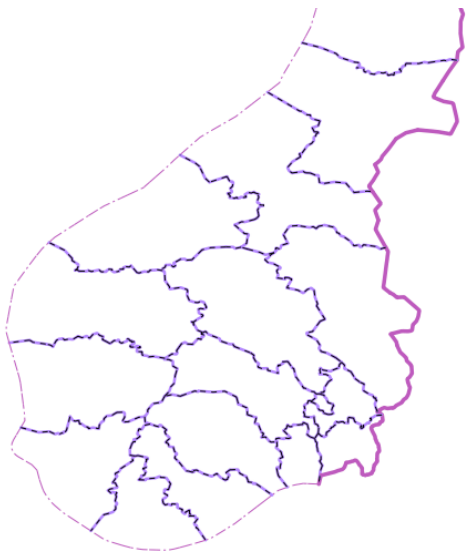
URL request: *http://openwms.statkart.no/skwms1/wms.topo2?request=GetMap&Service=WMS&version=1.3.0&format=image/png&crs=epsg:32632&layers=AdministrativeGrenser&bbox=186738,6382662,819642,7320797&width=500&height=500&styles=*	

Figure 5: Example on a WMS request and the corresponding result

2.5.6 WCS (Web Coverage Service)

Web Coverage Service(53) is another standard created by the Open Geospatial Consortium. It defines a method for accessing multi-dimensional data from a server. As with WMS, requests can be made from URLs and information about the service is given by using the operation GetCapabilites. The operation GetCoverage is used for requesting the needed the data. Depending on the implementation, it can be multiple output formats to choose from, e.g. jpg, png, xyz. Since the usage within this project is limited to a digital terrain model from the Norwegian Mapping Authority, more

information is excluded. A terrain result from Gløshaugen is displayed in Figure 6 as a picture. The color levels has been adjusted to make the differences more noticeable. Each pixel value represent a height in the terrain, and the darkest color is on the highest point.

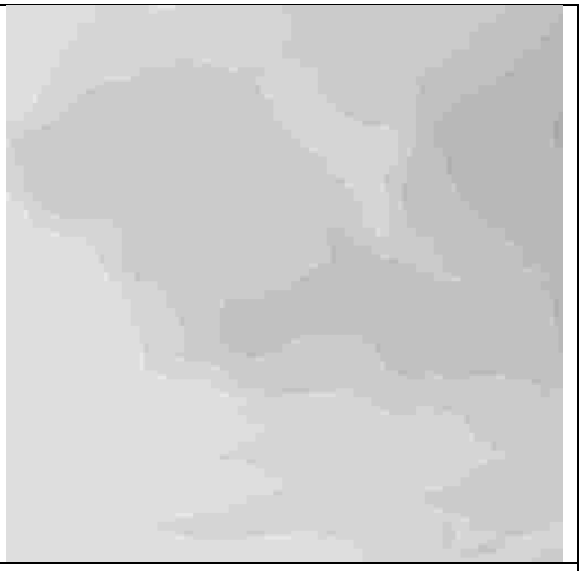
URL request: http://openwms.statkart.no/skwms1/wcs.dtm?service=WCS&version=1.0.0&request=GetCoverage&format=jpeg&crs=EPSG:32632&srs=EPSG:32632&coverage=land_utm33_10m&bbox=569800,7032300,570500,7033300&width=500&height=500	

Figure 6: Example on a WCS request and a result with color adjustments

2.6 Framework options within 3D graphics

A framework needs to be chosen, in order to make a virtual world with Oculus Rift. Here the word framework is a meant as a general term, which includes other categories, e.g. libraries. This chapter presents the process of choosing a fitting framework for this project.

The documentation from Oculus Rift(20) gives three official alternatives, i.e. Oculus SDK, Unreal Engine 3/UDK(54) and Unity game engine(55). However, there are several unofficial alternatives width varying quality.

The forum at the official Oculus Rift site has a category “Other Engine Integrations”(56), where people mentions frameworks that have different degrees of support for the Oculus Rift. The range of options is here since an open source framework makes it possible for dedicated users to make improvements, in this case by supporting the Rift. A Wikipedia article with a list over 3D game engines (57) was also a helpful source, but many of those do not support Oculus Rift.

It is worth mentioning that since the Oculus Rift still is bleeding edge, there have been a lack time for developers to adapt. The number of frameworks with support and the quality of the support will likely increase, assuming the big interest in Oculus Rift’s continues.

To being able to compare different options, some criterions needed to be set. The importing of terrain and buildings are crucial. Regarding terrain, it is important that the terrain can be imported and rendered in a practically manner. Support of the 3D format Collada is beneficial since the available buildings are from the free version of SketchUp, which support exporting to this format in the free version. Collada is also a common interchange file format.

Support for Oculus Rift is also important, and its completeness and quality varies. Open source alternatives are preferred since they give more control and the community can fix bugs or make

improvements. In addition, it was a choice within this project, to focus on open source solutions. Nevertheless, the table also include some proprietary alternatives. The cost of the program is important, since the budget is limited. When comparing programming languages, one is not strictly better than another is, but it is still an important criterion. The program language may affect performance and usability. Low-level languages is usually more difficult to use, while at the same time they may give a better performance. At the same time, earlier experience with a language or other similar languages will influence the productivity. The game engine criterion has no wrong answer, but at the same time, it may have a minor contribution. A game engine might be too oriented with a game setting, but it does not have to be. In addition, performance is quite important within games. As a result, being a game engine gives reason to believe that performance has a high priority. Recent activity is important for more than one reason. Low activity may indicate that a framework is not useful anymore, and that nobody will fix existing problems. In addition, when using new technology, i.e. Oculus Rift, an old framework with little activity gives little reassurance. Documentation is important for being able to do anything without using too much time. Even though available source code makes the need for documentation less crucial, it can be too time consuming to look in the code for help every time. An active community may also serve as a type documentation since different people usually have some common problems.

Table 1 presents potential choices. The ten first in the list include a rank, while the rest have an approximate order. The data gathering happened in early February 2014. The label “No”, means that no evidence was found for the current feature, and the label “Unclear” means either that quality of a feature is questionable or that it is unclear whether a feature exists or not. There are many options with the potential to work well. Even though Oculus Rift is such a new device, the need for Oculus Rift still gives many options. The three most promising alternatives is looked at more closely. The following sections describe some of the experiences with each of them.

Frameworks	Support of Oculus Rift	Support of Collada	Terrain	Free source code	Cost (dollar)	Programming language	Game engine	Recent activity	Documentation	Other	Rank
OpenSceneGraph (OSG)	Yes	Yes (plugin)	Yes	Yes	Free	C++	No	50+ commits in 2014	- Much documentation, but parts of it look outdated - Books	-Some effort is needed with the setup. +SketchUp-plugin	**1**
Three.js	Yes	Yes	Yes	Yes	Free	JavaScript	No	50+ in commits 2014	It apears to be many tutorials		**2**
JMonkeyEngine	Yes	Yes	Yes	Yes	Free	Java	Yes	50+ commits in 2014	Good documentation, many tutorials		**3**
UDK	Yes (official)	Yes (in the editor)	Yes	No	Free	C++, UnrealScript	Yes			Unclear how much that can be done with commands, instead of in their editor.	4
Oculus SDK	Yes (official)	No	Unclear	Yes	Free	C++					5
Minko	Yes	Yes	Yes	Yes	Free	ActionScript		Last commit Sep.13			6
JavaScript med CubicVR	Yes	Yes	Yes	Yes	Free	JavaScript	No	A little. Last commit 12.jan 14			7
Ogre3D	Unclear	Yes(plugin)	Yes	Yes	Free	C++	No		Some tutorials		8
Away3D	Unclear	Yes	Yes	Yes	Free	ActionScript		Some activity this year			9
Torque3D	Yes	Yes	Yes	Yes	Free*	C++, Lua		Last commit Nov.13			10
JavaScript med cesium.js	Yes	Unclear	Unclear	Yes	Free	JavaScript		New version every month			
Horde3D GameEngine	Yes	Unclear	Unclear	Yes	Free	C++				Maintained by a university. Based on Horde3D	
Outerra	Yes	Yes	No	Yes	15	Non?	No			Unfinished, lacking API	
Unity Pro	Yes(official)	Yes(editor)	Yes	No	75/month	C++, C#	Yes			Much use of the editor	
Irrlicht	Unclear	Yes	Yes	Yes	Free	C++		Last update: 2014-01-02			
OpenWebGlobe	No	Unclear	No	Yes	Free	JavaScript	No	Last commit: 8.des.13	Some tutorials and some documentation at GitHub		
Panda3D	Unclear			Yes							
Esenthel Engine	Yes	Yes	Yes	No	200+						
Visual3D Engine	Unclear			No	150+						
Unigine	Yes			No	Not free						
Grit	No			Yes		C++, Lua					
Ardor3D	No	Yes		Yes	Free	Java	Yes				
Cafu Engine	No			Yes	Free		Yes	10+ commits 2014			
Crystal Space	No			Yes	Free	C++					
GamePlay3D	No		Yes	Yes	Free	C++	Yes	50+ commits 2014			

*Some functionalty is not free

Table 1: Framework overview

2.6.1 OpenSceneGraph (OSG)

OpenSceneGraph (58) is a quite extensive open source 3D graphics toolkit, which dates back to 1998. Even though originally hypothesized to be the most promising, it became less attractive since the documentation in some areas were either lacking or outdated. For example, the documentation for adding the Collada plugin is several years old, and it is not clear which versions of the prerequisite library that is compatible(59,60). As a result, the attempt to import a Collada file was not successful.

An alternative to using the Collada plugin is to use the plugin (61) for SketchUp that exports files to the internal OpenSceneGraph format. It was quite easy to install in Windows 7, and it has a version for both SketchUp 8 and SketchUp 2013. Figure 7 displays the rendering of a building at Gløshaugen. It is the clear that the plugin would be helpful in this project, but it is too specific to be a good solution in general.

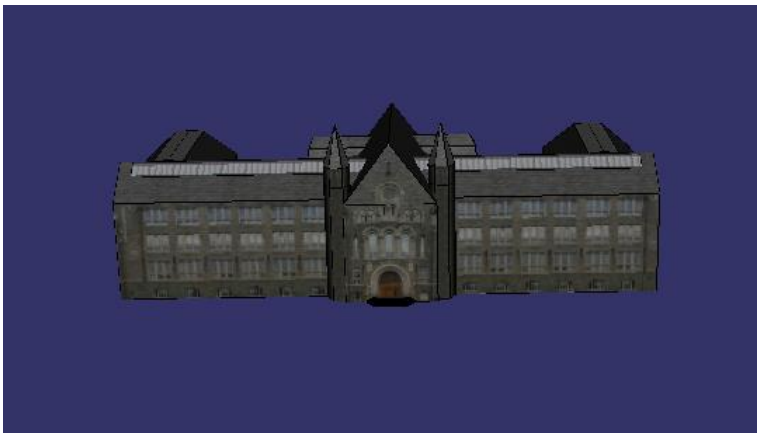


Figure 7: A rendering of "Main administration building" in OSG

The website to OpenSceneGraph provides some runnable examples with source code. Two examples (osgshaderterrain and osgforest) gives an idea of how the importing and rendering procedure for terrain works. The former renders a terrain, while the latter also places trees on this terrain. Figure 8 displays the rendered result, and it looks satisfactory. Based on the source code, terrain appears feasible to implement, but a bit cumbersome.

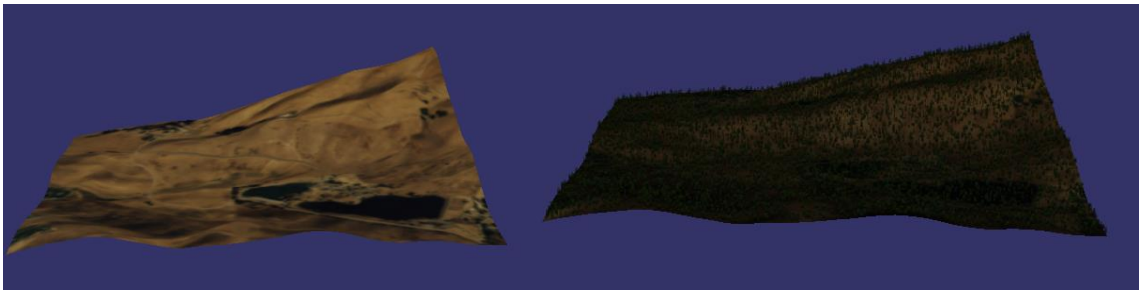


Figure 8: OSG examples with terrain

A Oculus Rift plugin is located at Github(62) and is mostly developed by one person. After some preliminary testing, the plugin seem to work mostly as it should. A couple of warnings and error messages in the console is concerning, but the program is rendering in stereo. The characteristic barrel shaped black background is missing though. Figure 9 displays how out looks on the computer screen. Based on the number of people at GitHub that is watching or have labeled the project as

interesting, the interest appears to be quite low. This gives reason to wonder if the plugin will keep up with the updates from Oculus SDK and bugs that shows up.

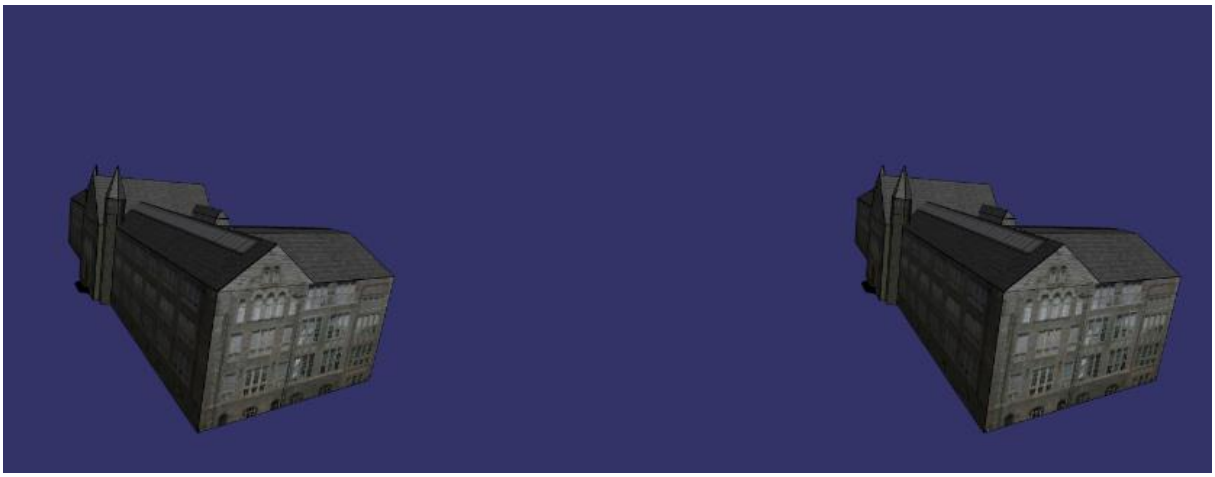


Figure 9: OpenSceneGraph with Oculus Rift integration

In addition to the plugin issue with Collada, there are no binaries for the newest stable version. This contributes even more to make the entry barrier higher since the frameworks need to be compiled. Even in the earlier version, with binaries, many things could go wrong like all the environmental variables that need to be set.

OpenSceneGraph does not have support for collision detection. Even though it might not be need in all cases, it is a feature that many might miss. If the feature is important, osgBullet(63) can be used instead. It is a integration that combines OpenSceneGraph with the collision detection library Bullet(64).

The fact that the OpenSceneGraph is written in C++ is good for performance. At the same time, many problems may occur, that a high-level language will abstract away.

2.6.2 Three.js

Three.js (65) is a relatively new 3D graphics library in JavaScript and dates back to April 2010. It has a renderer that uses the powerful WebGL. To support older web browsers, the library also provides an alternative render method as a fallback. Three.js has managed to become quite popular. More than 15000 users at GitHub have given it a star, which indicates that they find the library interesting. Many people are also contributing to the library. Even though two people is behind most of the code, GitHub(66) lists over 300 people as contributors.

In addition to the core three.js, extra libraries made for three.js, or general libraries for JavaScript or WebGL can add extra functionality. Luckily, combining compatible libraries in JavaScript is quite easy.

ColladaLoader.js(67) provides the possibility for importing the buildings in the Collada format. Even though the loading is quite easy, the result was disappointing. As seen in Figure 10: Rendering of a Collada file in three.js, several of the textures on the building looks like someone have greased painting on the wall. In addition, the console outputs an error message about WebGL.

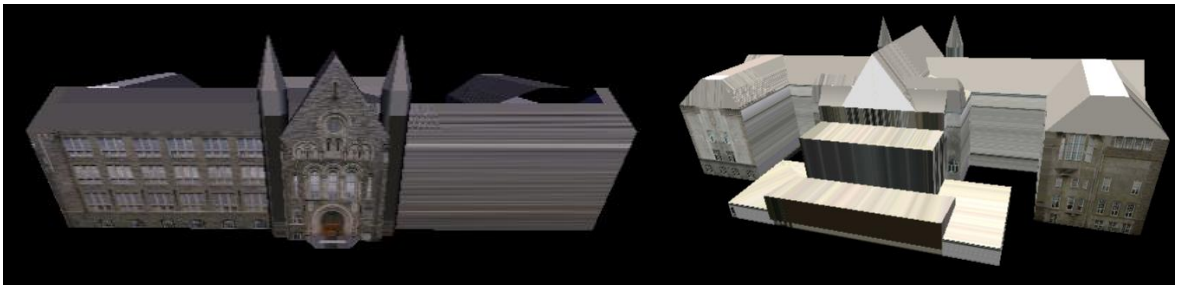


Figure 10: Rendering of a Collada file in three.js

An explanation for the unfortunate texture problem was discovered later in the project. When the renderer wants to map a texture to a surface, it can choose between different techniques. The wanted behavior in Figure 10 is to use “repeat”, but the renderer used “clamping” instead. “Repeat” means that the texture repeats itself multiple times on an object. With “clamping” one copy of the texture is placed on the object, while on the remaining area the color closest to the first copy is used. Figure 11 illustrates the difference. The problem is that the Collada loader use clamping even though instructed to use repeat when the resolution of the textures is not a power of two(e.g. 512x128).

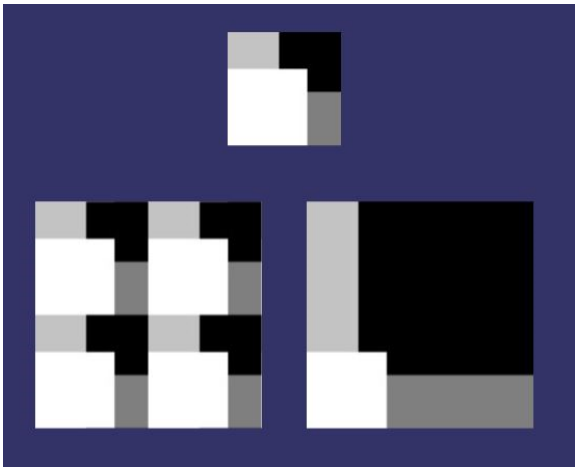


Figure 11: Repeat vs clamping

Three.js has plugins for others loaders, like the common OBJ format, which makes the problems with the Collada files less severe. Blender (as described in “Relevant programs”) can convert the Collada files, but since those do not specify the ambient light values, these becomes zero in the files. When not fixing this, the building will look black if no directional light is present. Figure 12 displays the result after the ambient values is fixed.

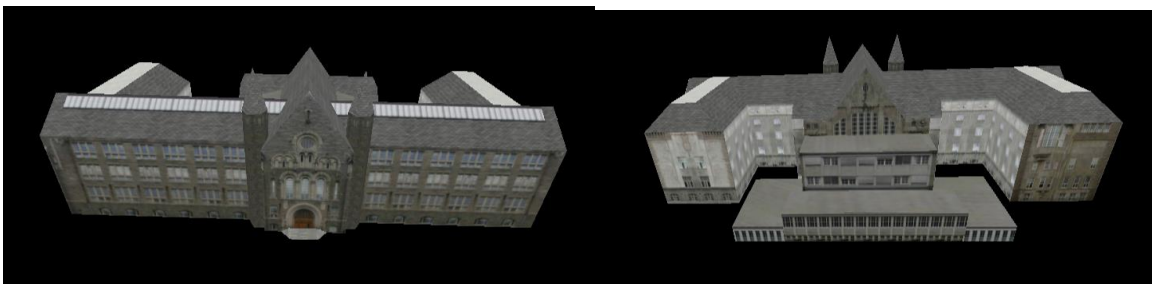


Figure 12: Rendering of a OBJ file in THREE.js

Bjørn Sandvik, the author of the thematic mapping blog (68), presents some possibilities for terrain importing and rendering with three.js, very well. He uses Jotunheimen, a mountainous area in Norway, as an example. Combined with terrain examples at the official three.js website(69), it is clear that the terrain functionality is sufficiently supported.

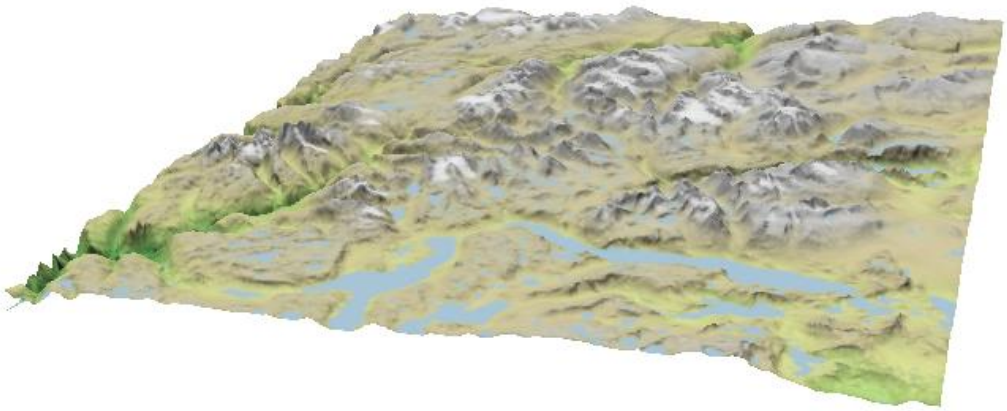


Figure 13: Rendering of terrain in THREE.js(70)

With THREE.js, there exists more than one alternative for supporting Oculus Rift. The two libraries, OculusBridge(71) and THREE.OculusRiftEffect(72) combined with a native program, Oculus Bridge.exe, ended up being used. The latter is necessary for the head tracking data.

Another alternative is vr.js, combined with a plugin in Chrome or Firefox (73). The installing process of the plugin takes more effort than usual, which makes it less user friendly for the average user. A third alternative consists of the libraries, THREE.OculusControls(74) and THREE.OculusRiftEffect combined with oculus-rest(71) for the head tracking data. It is reassuring with several alternatives, in case the chosen does not work as expected.

Figure 14 displays the rendering of a textured cube in Oculus Rift mode.

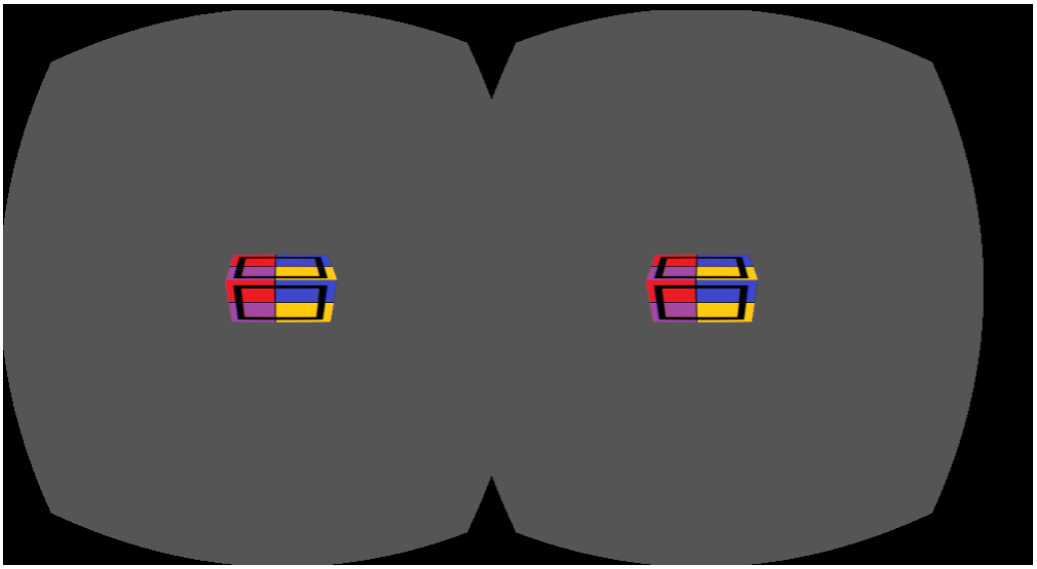


Figure 14: Textured cube rendered in three.js in "Oculus mode"

2.6.3 JMonkeyEngine

JMonkeyEngine(75) is an open source game engine made in Java and dates back to 2003. Their website claims to have a good documentation and it appears to be right. However, outdated information in forums, related to an older version, might be confusing. The game engine have its own software development kit, jMonkeyEngine SDK, which is based on the integrated development environment NetBeans(76).

Regarding the importing of buildings, the attempt to load Collada files with textures was not successful. Other options were attempted, like converting Collada files to another format with better support. An acceptable result was achieved for the test building, when using a plugin for SketchUp and doing some extra steps. An operation, explode, needed to be performed on the building, and the file names on the textures with "ÆØÅ" had to be replaced. An even bigger set back, was that the plugin did not work in the newer version of SketchUp, which is needed for the newer building models.

Figure 15 displays an example of rendered terrain made by following a tutorial in the documentation (77). The program makes the terrain by using an image, where each pixel represents a height value. A texture is also mapped on the terrain.

Figure 15: Terrain example in jMonkeyEngine

Most of the information about the oculus rift library is located in a forum post(78) at JMonkeyEngine's web site. As seen from Figure 16, the rendering of an example dataset looks promising. Unfortunately, the sensor integration to the Rift did not work. The process of adding the library was cumbersome and error-prone, which might be the cause of the problem. It is worth mentioning that the library is a work in a progress, and it may improve in the future.

Because of the mentioned issues, jMonkeyEngine was no longer a potential choice. As a result, it was not necessary to investigate further.

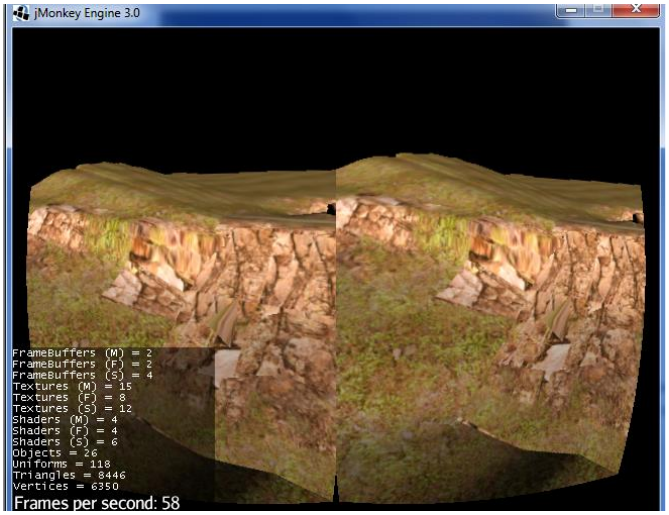


Figure 16: Rendering in stereo with jMonkeyEngine

2.6.4 Conclusion:

JMonkeyEngine is as already mentioned not a candidate anymore. Both OSG and three.js appears to be viable alternatives. OSG is feature rich, but have at the same time a high entry barrier. Sharing a resulting program could be unnecessary troublesome because of the install process. An application in three.js is easier to share. Everyone with a modern web browser should be able to run the application, although, the head tracking requires a native file or a plugin. The high number of users in the three.js is reassuring. Figure 17, which displays search trends, suggests that three.js is quite popular compared to OSG. Three.js also looks safer regarding the Oculus Rift support. 3D Graphics in the web browser relatively new, which also makes it a more exciting choice. To conclude, three.js is the choice in this project.

Figure 17: Google trends comparison between OSG, three.js and jMonkeyEngine(79)

In retrospect regarding OSG, importing files in the OBJ format should have been considered. OSG supports this format without a plugin that causes problems. It would not have changed the decision, though.

3 Implementation

This chapter describes the created program from an implementation perspective. Figure 18 illustrates the different components that interact together. A website hosts the main application, which is available with the internet. The computer is connected to the Oculus Rift, and it runs a native program, “Oculus Bridge”, which handles the sensor data from the Rift. The web site can retrieve terrain data from a stored file or from a WMS and WCS server. The code run on the web site, has the focus in this chapter. In addition, it also describes the preprocessing of data.

As mentioned in the section about various frameworks, the JavaScript library three.js became the choice in the project. One of the advantages with JavaScript is that adding different libraries are quite easy as long as they are compatible. Because of this, three.js is not the only library used in the implementation. Some of the extra libraries are specific plugins for three.js, while others are useful for 3D graphics or JavaScript in general.

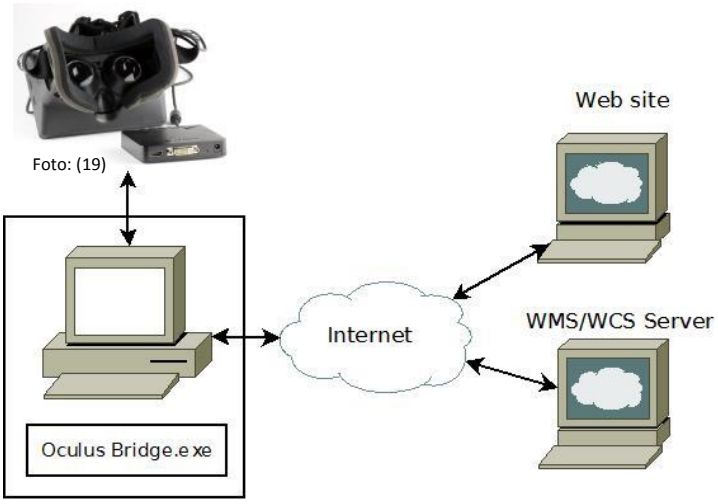


Figure 18: System overview

3.1 Implementation description

3.1.1 Overview

The files index.html and index.js are the main files. Since the GUI on the website only consists of the rendered 3D model over Gløshaugen, the index.html is quite simple. It imports all the JavaScript libraries and source files that the program needs. The file do also have a div tag, which contains the rendering, and a style element that sets the margin to zero. Since the website does not have any more content, the whole page is showing the rendering.

Figure 19 attempts to give a general idea about the flow and the structure of the program. The diagram use containment in order to show hierarchy and sequential execution. When a function (except constructors) like init() is called, the actions contained inside the init() element, is called by the function. Arrows is another way to say that a function calls another function.

The main file, index.js, sets up all the common elements needed for three.js to a render a 3D world. It also takes parameters given in the web address (URL), and delegates tasks to other files. The two main functions in this file, are *init()* and *render()*.

Init() does everything that is necessary to do in the beginning. Among other things, it adds the mandatory three.js elements. A scene is necessary to store all the 3D objects, and a camera defines

where in the scene to look. A renderer is also required so that we can see what the camera is looking at.

Figure 19: Diagram describing flow and hierarchy

To have some interactivity, the init() function also add a controller, in this case by using the FlyControls library. With the WASD keys the camera moves forward, left, backwards and right. Pushing the R or F key, makes the camera go upwards or downwards (from the cameras point of view). Pressing Q, E or the arrow keys will change the rotation.

As described in the section about Oculus Rift, when using the Rift, the screen is divided in a left and a right part with some distortion. To be able to enjoy the web site without Oculus Rift, there are also non-Rift mode. If the Rift mode is on, the Init() function also starts a initializing on the communication with the sensors and making a oculus renderer. The oculus libraries takes care of the heavy lifting.

For a more robust experience, an event listener listens to window size changes so that the camera and renderer can adapt to these. The stats.js library helps with measuring performance by adding a little info box in the corner that displays frames per second.

With the above mentioned functionally, we have a nice setup, but nothing to display. Other JavaScript files takes responsibility for importing the terrain and buildings, and detecting collisions. The terrain is loaded before the buildings, since the buildings should be on top of the terrain.

The render function is a recursive function that renders the scene as the camera sees it repeatedly. *Render()* contains everything that is needed to be done between each frame. This includes measuring the time to get frame per seconds, blocking move directions to avoid collision, updating the cameras position and rotations. The controller handles the latter action.

Since the web site does not have a fancy graphic user interface, the user can give some parameter with the query string. This is the last part of the URL (web address) and starts with a question mark (?) and contains a sequence of elements, <variable name>=<value>, separated by an ampersand.

3.1.2 Terrain

TerrainImporter.js is responsible for creating a terrain mesh. The mesh consists of a geometry and a material. The geometry contains all the height values, and the material is the texture applied on the geometry.

The height values is loaded from a file or from WCS (WCS is described in 2.5.6). Both cases use the simple .XYZ format where each line consists of the x, y and z-values separated by a space.

When importing with WCS, the caller function chooses a WCS service from a dictionary. The dictionary currently only contain the Norwegian Mapping Authority's service. Since the format chosen for the WCS is the same, as when loading from file, the rest of the importing procedure is the same.

To be able to place buildings on top of the terrain at a later stage, some data about the terrain is retrieved by iterating through all the x, y and z values. To find a resolution in on the x and y values, we assume that it is equally over the whole data set. Figure 20 visualizes an example of the XY-plane of a small terrain sample.

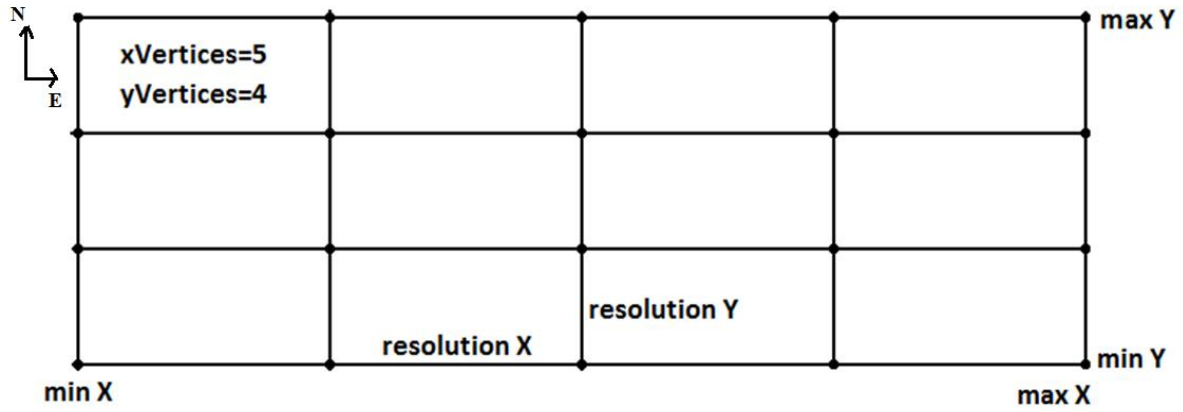


Figure 20: XY plane of terrain data from the XYZ file

The resolution x and y is found by taking the minimum difference between e.g. x in the first point and x in every other point excluding the cases when the two values are equal. Figure 21 explains the calculation with pseudo code.

```
for every point:
        if (point.x not equal firstPoint.x) then:
                differenceX= abs(point.x - firstPoint.x)
                resolutionX = min(differenceX, resolutionX)
```

Figure 21: Pseudo code - Find resolution of X

The resolution is used to calculate the number of x and y vertices by following Equation 1.

$$xVertices = \frac{maxX - minX}{resolutionX} + 1$$

Equation 1: Number of x vertices

The number of vertices is needed when making a flat terrain object. Then, every vertex in the terrain object can correspond to a point in the terrain data. In addition, the width and height in virtual world size is given. The function getScale() calculates the scale between the real world and the virtual world before adding the height values to the flat terrain object.

After the geometry of the terrain is ready, the material is made by using a either a map image or ortho photo corresponding to the terrain. The image is loaded either from a stored file or by using WMS. The stored file is quicker to load, but is at the same time static. As with WCS, all the necessary information except the bounding box and width and height is stored.

3.1.3 Buildings

The program adds buildings to the scene depending on a URL-parameter. The program supports Collada files, OBJ and the internal three.js format (js), but the latter is most successful and therefore focused on here. When choosing the js-format, the loader adds all the buildings from the js-folder. The following procedure is performed for each building. The embedded loader inputs the file paths for the building file and the texture folder. A vertex normal have to be computed in order for the collision detection to work. A corresponding KML file has the geographical coordinates in WGS84 for the building. The Holsen library converts those to UTM32 since this coordinate system is easier to work with. The function georeferenceBuilding() places the building on the terrain by using the coordinates and information about the terrain. Since the virtual coordinate system is not equal to the real world, this requires some transformations. The height value of the buildings is calculated by

finding the height value to the terrain vertex, which is closest to the mid position of the building. With interpolation, the result could have could have slightly more accurate. In addition, sometimes the midpoint of the building is a bad place to use, e.g. when the midpoint is outside the building. Figure 22 summarizes the process.

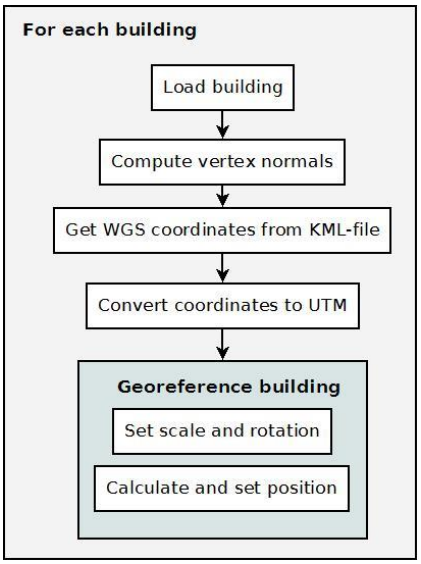


Figure 22: Load building

3.1.4 Collision

CollisionDetection.js is responsible for finding obstacles in the movement directions and blocking the movement in these directions if the obstacle is close enough. In addition, there are some unused code, which serve as a starting point for interacting with the objects in the scene. The following description does not describe the latter.

BlockCollidingDirections() is the only exposed function. It needs the movement controller as parameter to be able to block the movement in directions with obstacles.

It is necessary to know the global movement directions, in order to detect obstacles for these directions. The directions change when the camera rotates, e.g. the forward direction is not the same after turning 180 degrees in the horizontal plane. An invisible help geometry keeps track of the changes in the directions. The geometry is an octahedron, illustrated in Figure 23, where each vertex represent one of the six possible movement directions(up, down, forward, back, left, right). The octahedron synchronizes its position and rotations with the camera.

Three.js' raycaster can make a ray based on the position and direction vector, an afterwards find all the intersections between the ray and each object in the scene. If the closest intersection is close enough, then there is a potential collision in the analyzed direction. The movement is disabled in all the directions that will cause a collision.

The render function in the main file calls *BlockCollidingDirections()* before the controller updates the position and rotation. Otherwise, the blocking would not have any effect.

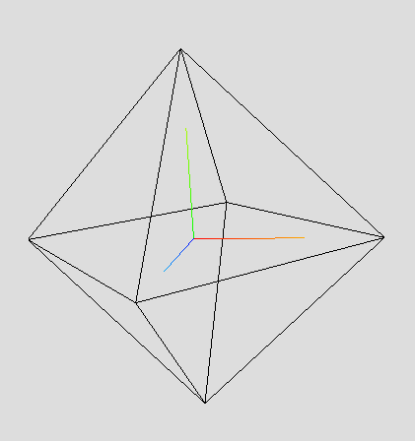


Figure 23: Icosahedron and axis in three.js

3.1.5 Flythrough:

The fly through feature was added late in the project and is not considered one of the core elements. This results in a short description.

The feature supports recording and playing of the cameras view. The recording process stores a timestamp, position and rotation (approximately) equal intervals. Depending on a URL parameter, the recording is downloaded as a text file or saved in the local storage in the browser. An operator can run the recording by pushing a key, if the feature is turned on.

3.1.6 Performance measurements

The performance of the website is measured with a combination of timers within the code and the Google Chromes developer tools. First, we will look at the startup measurements, then the frame per seconds. The program outputs timestamps relative to the start before starting the terrain process and after it is finished. An end time after the scene adds each building, is also used. In the latter case, the textures have not been added yet. The developer tools tells the DOMContentLoad time, when all html and JavaScript content is loaded, and when every resource is loaded. It also tells how long time each resource takes to load. This feature is used to get the load time to the terrain texture.

A function, getAverageFPS, measures the average frames per second by using a counter that increases by one for each rendering. The average is equal to the difference in render count divided by difference in time.

3.1.7 Various details

Three.js, and many other graphics libraries, considers the y-axis to point upwards instead of the z-axis. Since this program uses the z-axis to point upwards, it needs to make some rotation adaptions. Automated tests were not a high priority, but some tests were made using the test framework Jasmine(80).

Github(66) hosts the code in a repository(81), and the Github Pages(82) hosts the resulting web page(83). There is a good integration between the Github and Github Pages. The web host site have some limitations though. It does not execute server languages, e.g. PHP. In addition, it ignores files starting with underscore, which a small portion of the textures have. This have the unfortunate effect that some walls becomes black.

3.1.8 External libraries:

Figure 24 presents a list of all the external libraries from this project. The main library is three.js with release version number 66 (r66). Based on the list over all the releases, a new version is usually released every month(84).

The three oculus rift libraries work together on different elements. “Oculus Bridge.exe” is responsible for receiving the sensor data from the Rift, and OculusBridge.js use this sensor data. OculusRiftEffect.js is responsible for adapting the rendering to the Rift.

The loaders is included to support loading of Collada and OBJ files. Both MTLLoader.js and OBJMTLLoader.js is necessary to load the OBJ files. As already mentioned, the focus have mostly been on the internal three.js format.

FlyControls.js is responsible for the navigation in the virtual world. It moves the user (camera) around in the virtual world based on keyboard and mouse input.

Jquery(85) is a popular library that eases some JavaScript processes. In this case, it have helped with the loading of different files.

Holsen.js(86) is a ported library that, among other things, can transform coordinates WGS84 to UTM32. It is based on an old FORTRAN program that was made by a Jon Holsen, a professor of geomatics at NTH (the earlier name of NTNU).

Detector.js(87) checks whether the web browser supports WebGL or not, and stats.js(88) provides a small info box that display either the frames per seconds or the milliseconds need to the render a frame.

- Main library:
  - Three.min.js(R66)
- Oculus:
  - Oculus Bridge.exe
  - OculusBridge.js
  - OculusRiftEffect.js (Is called RiftCamera.js in this repository)
- Loaders:
  - ColladaLoader.js
  - MTLLoader.js
  - OBJMTLLoader.js
- Controls:
  - FlyControls.js
- Jquery-2.1.0.js
- Holsen.js
- Detector.js
- Stats.js

Figure 24: List of used external libraries

3.2 Preprocessing of data

3.2.1 Terrain

This section explains the acquiring of the terrain data from NMA and the necessary processing. The 50x50 kilometer tile covering Gløshaugen in UTM32 with 10x10 meter resolution was chosen, ordered and downloaded. The GDAL library(46) processed the downloaded DEM-file in two steps. *Gdalwarp*(89) clips the dataset with a bounding box that covers the Gløshaugen. The process also

changes the format to GeoTIFF, the default output format. Gdal_translate (90) then converts the format to user friendly XYZ format.

```
gdalwarp -te 569800 7032300 570500 7033300
7005_2_10m_z32.dem gloshaugen.tif
gdal_translate -of XYZ gloshaugen.tif gloshaugen.xyz
```

Figure 25: Preprocessing of terrain data in batch file with GDAL library

3.2.2 Buildings

The 3D buildings at Gløshaugen are in the Google Earth format .KMZ and SketchUp format .skp, none of which that can be used directly. Three.js' internal format is the target, but Collada and OBJ files were tested to some extent.

When converting the exporting the SketchUp file to e.g. Collada, the georeferenced coordinates is not included. That is one reason the .KMZ files were used as a starting point. They contain a Collada file, a KML file with the coordinates and an image folder. However, all the files have unfortunate names that do not reflect the KMZ-file name. For example, the Collada file is named untitled.dae most of the times.

There is two approaches for preparing the building files. The necessary actions can be done manually or script can be implemented to automate the process. The former method were mainly used in the beginning, while the latter was attempted later with mixed results.

The manually method consists of renaming the KMZ-files to zip before unzipping them. The next step is to give the image folder, KML-file and the Collada file to more fitting names, i.e. the name of the original KMZ-file. Since the Collada file references the image files, the folder name in each reference also have to be updated. After this, Blender (described in the literature chapter) can import a Collada file, and then export it to OBJ or the three.js format. The latter requires an plugin(91) to be installed first. When exporting to OBJ, the ambient light levels need to be changed from zero to a bigger value in order to be seen without a direct light in the scene.

Due to all the mentioned steps, it would be much better if a more automated method could be accomplished. The implemented solution is described first before some suggested changes to improve the result. Figure 26 visualizes the procedure.

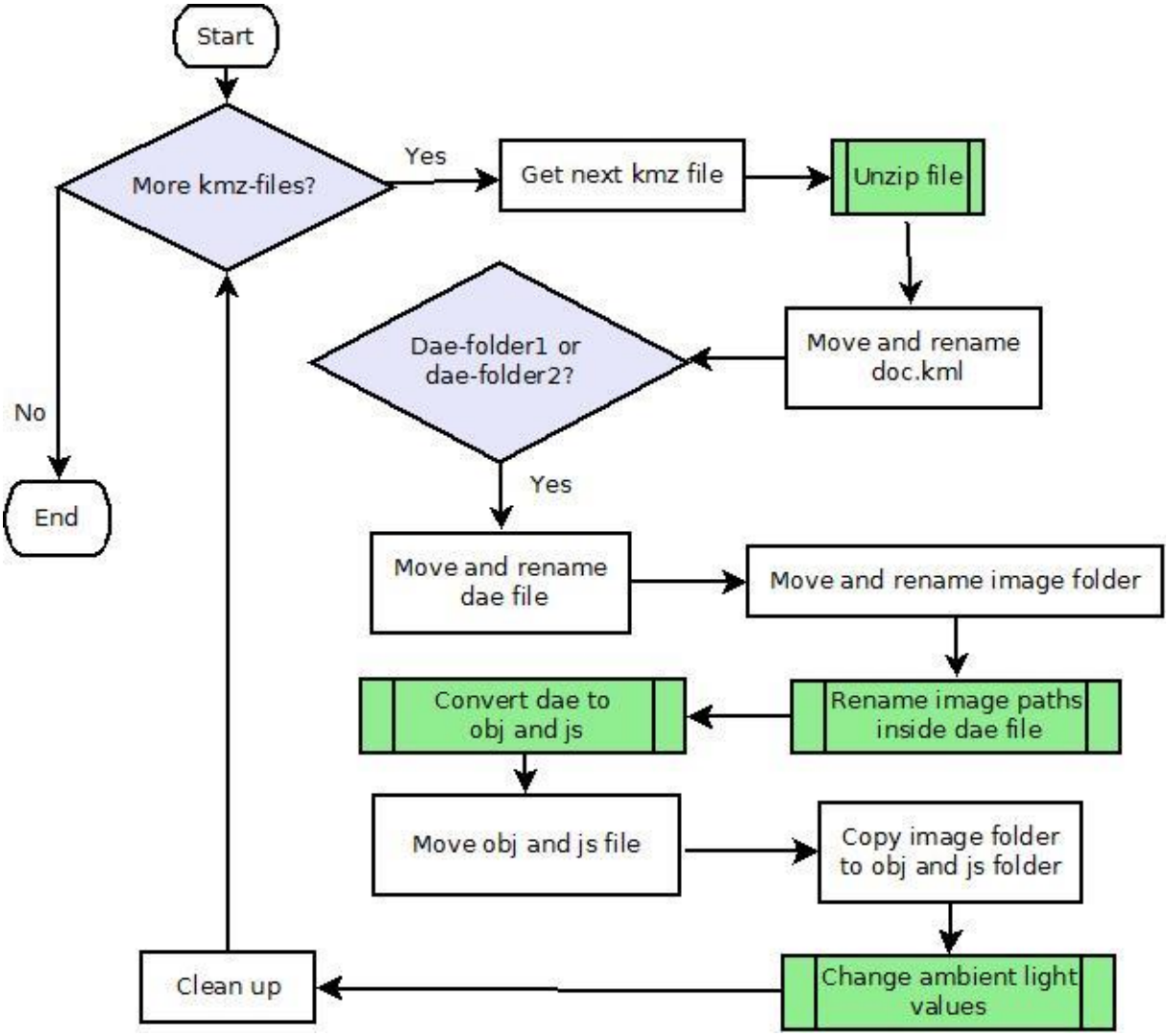


Figure 26: Preprocessing of buildings

The processing is done with a combination of batch files(using the command line language from Windows) and the programming language Python(43).

ProcessKmzFiles.bat is the main file. It iterates through all the KMZ-files in the current directory. The following process in done for each KMZ-file. First the command line library for 7z-zip(92) unzips the file. Then the KML file is renamed and moved to a KML folder. Since the names and the other files in the KMZ-file varies, two different situations is handled separately. For each the dae-file and image folder is renamed and moved to a dae folder. Then all the texture referances in the dae file is updated by calling the python script replaceInFile.py. The python script takes a list of files and a list of find and replace pairs. It reads every line the inputted files, and replaces the given word line by line before overwriting the existing files.

The next step is the convert the dae files to OBJ and the internal format to three.js. This is possible by calling blender with some parameters, including another python script which tells blender what to do. The created files are moved over to an OBJ and JS folder. The material file for the OBJ format need its ambient light parameters replaces. It is done with the same method as the last replace. The script finalizes by removing the rests of the unzipped KMZ-file.

One problem with the mention solution is that it does not catch all the cases of the content of the KMZ-files. The KML-files contains information that should make the method more robust. By parsing

the KML this could have been accomplished. More, if not all of the code, could also have been written in Python since it is a more user-friendly language.

The resulting buildings have to be found by the website. The website host, GitHub Pages, does not support directory listing. This means that the JavaScript program cannot find all the files that is stored in a folder. The following solution is a workaround for this problem. A batch file is responsible for generating and updating an html-file containing links to all the content. This means that the batch file should be run after adding or deleting building files.

3.2.2.1 Image shrinking

Image shrinking was an attempt to improve performance. A manual approach would be very slow, since there is about 500 pictures. FastStone Photo Resizer(93) were used instead. Among other things, it can create resized images with relative sizes to the original. It made a copy of all the images that were one fourth in both width and height from one folder at a time. Unfortunately, the program outputs all the files to one image format (instead of using the input format). Most of the input files is in jpeg, but not all. This means that the reference to these images is wrong. The building walls with those will become black.

4 Results

The web site should work on multiple modern web browsers. However, Google Chrome in Windows 7 have been testing the web site most of the time. When using Oculus Rift, the operation system is limited to Windows and probably Mac (there exists a Mac version of the Oculus bridge.exe). Even smartphones with a WebGL enabled web browser can be used to a very limited extent. The focus of this chapter is on performance, but it also contains a description of the visual results and the navigation.

4.1 Performance

This section presents performance measurements from the website. This includes the startup time and frames per seconds after startup. Two different computers, computer 1 and computer 2, performed most of the tests. A third computer tested one configuration. Table 2 contains specifications from the test computers. Computer 1 is a portable computer, while the others are stationary.

Computer 2 used an older version of the web browser on the most of the tests. To have a more fair comparison, this computer also performed a few extra tests with the updated version. Table 2 also provides benchmark values for the processor and the graphics card. They were collected from two different benchmark web sites(94,95). The benchmarks is based on multiple tests in PassMark PerformanceTest(96) done by different users. The processor, memory and graphics card indicates that computer 2 should give the best performance.

	Computer1:	**Computer 2**	**Computer 3**
Operation System	Windows 7 Home Premium Service	Windows 7 Enterprise Service	Windows 7 Enterprise
Processor:	Intel® Core™ i5-3210M CPU @ 2.50GHz, 2 Cores, 4 Logical Processors (Q1 2012)	Intel® Xeon® CPU X5650 @ 2.67GHz, 6 Cores, 12 logic processors(Q1 2010)	Intel® Core™ i7-2600 CPU @ 3.40GHz, 4 Cores, 8 Logical Processors (Q4 2010)
Benchmark*	3809	7662	8299
Memory:	6GB	24GB	8GB
System type:	64-bit Operation System	64-bit Operation System	64-bit Operation System
Graphics Card	Intel® HD Graphics 4000	NVIDIA Quadro FX4800	AMD RADEON HD 6350
Benchmark**	456 / 413	990	213
Extra graphic card	NVIDIA GeForce 610M		
Screen-resolution	1366x768	1920x1080	1920x1080
Screen-refresh rate	60 Hz	60 Hz	60 Hz
Web browser (Google Chrome):	35.0.1916.114	34.0.1847.137/(35.0.1916.114)	35.0.1916.114

Table 2: Computer specifications

The website has many configuration options that influences the performance. It can load the digital terrain model from a file or from WCS. WMS or a local file gives the terrain texture. The texture is either a map or an orthophoto. Lastly, the buildings have a shrunk version of the texture. With more

than one test computer and measurements per configuration, the number of possible combinations becomes quite large. Some of those are measured.

4.1.1 Start-up times

Computer 1 and computer 2 measures the startup time in two different configurations. Configuration 1 consists of the terrain and buildings, where the terrain consists of height data and an orthophoto. Configuration 2 uses height data from WCS and an orthophoto from WMS before adding the buildings. The cache was disabled during the tests to avoid a decrease in load time after the first measurement. All the computers performed the tests at NTNU Gløshaugen, but computer 1 used a wireless connection. This results in a lower download speed, but in both cases, it is quite high. Measurements at a web site(97) resulted in 21 Mbps for computer 1 and around 90 Mbps for computer 2.

Table 3 includes average load time and standard deviation measurements. Appendix A: Performance tables contains the complete datasets, not just the averages and standard deviations. Since many of the processes on the web page is asynchronous, the total time is not equal the sum of all the part times. The loading time for the DOM-Content, represent the time it takes all the html and JavaScript files to load. Terrain heights describes how long time it takes to create and add the terrain to the scene without the texture, while the terrain texture time says how long time the terrain texture takes. The buildings have the same principle. One of the measurements on configuration 2 with computer 2 had to be disregarded since it was much lower than the rest. The time and standard deviations (SD) is in seconds. The column graph in Figure 27 and Figure 28 displays the same data, except for the standard deviations.

	Configuration 1				Configuration 2			
	Computer 1		Computer 2		Computer 1		Computer 2	
	Time	SD	Time	SD	Time	SD	Time	SD
DOM-Content loaded	1.62	0.871	1.42	0.061	1.14	0.210	1.38	0.134
Terrain heights	0.77	0.416	0.42	0.041	1.14	0.127	1.27	0.132
Terrain texture	2.24	1.428	0.62	0.119	7.17	0.935	3.66	0.955
Buildings without textures	2.09	0.119	1.50	0.125	1.70	0.112	1.37	0.120
Buildings with textures	12.98	3.243	12.91	0.398	14.64	3.950	12.56	0.521
Total time	15.36	3.318	14.74	0.369	16.89	4.067	15.18	0.523

Table 3: Startup times

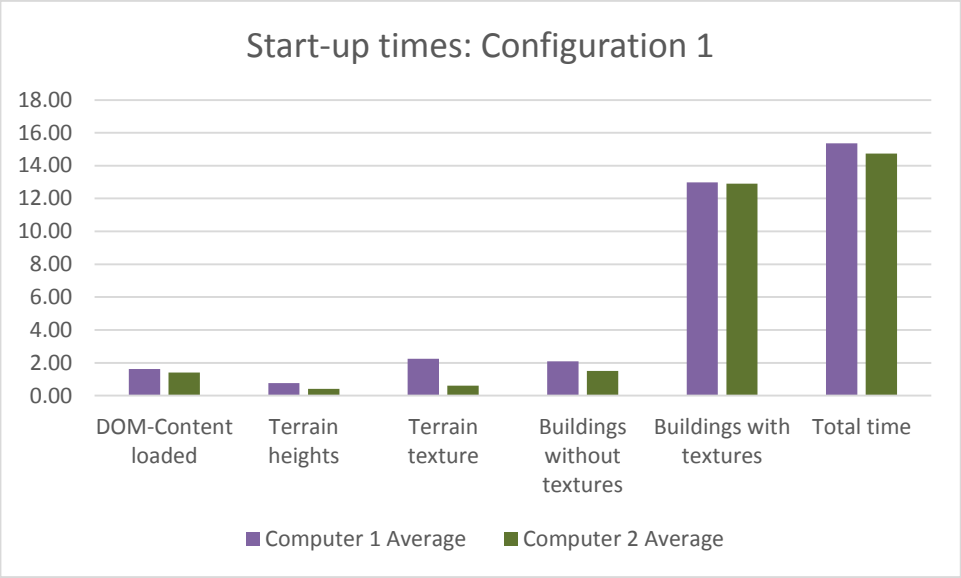


Figure 27: Start-up times from configuration 1

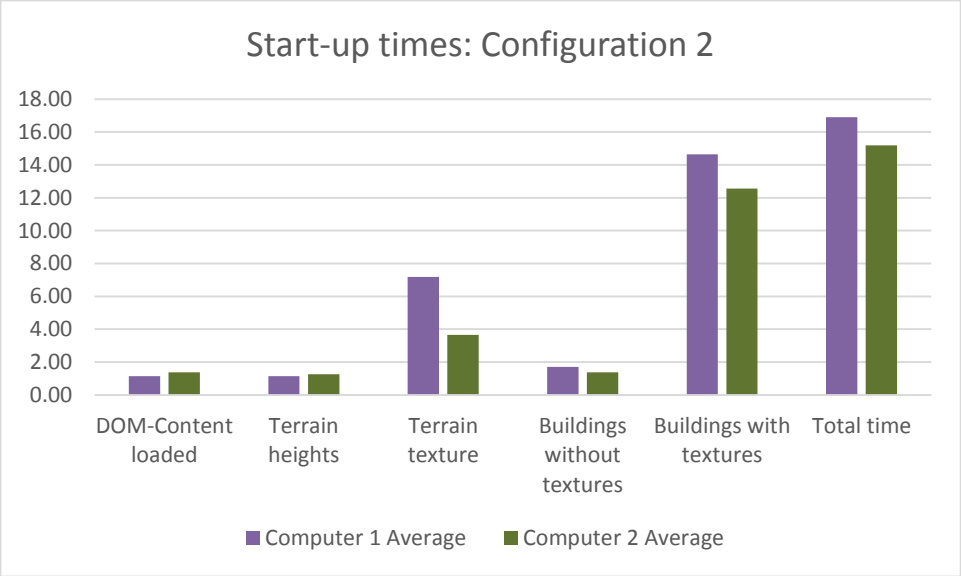


Figure 28: Start-up times from configuration 2

The start-up time in both configurations are around 15 seconds. Computer 2 has a slightly better average time compared to computer1. The average in configuration 2 is slightly better than configuration 1. The measurements from computer 1 have a considerable standard deviation, which makes it reduces the credibility on the mentioned average times. The loading of the textures has most variation. The loading of building textures is the biggest contributor on the total load time, but the different download speeds on the computers have no apparent effect. This suggests that the hosting server is the bottleneck. In configuration 2, which uses WMS, the loading time the terrain texture is also noticeable. Although, it does not appear to have a big effect on the total time. It is reasonable, since the program is not stopping until the server provides the image.

4.1.2 Frame per seconds

The FPS were tested in two states and a flythrough. A state represent a position and rotation in the virtual world. In state 1 and 2, the measured FPS equals the average over 5 seconds. The flythrough, a tour of 15 seconds, guides the camera around the virtual world. The reason behind not only using static states is that performance could potentially be affected by moving camera. In addition, moving around in a world is more representative for the use, compared to just looking at one specific view.

Four different configurations are tested. The first only renders the terrain, with the ortophoto as texture and with terrain data from the stored file. The latter is relevant since the terrain model from WCS has a higher density, which could influence the measurements. The second configuration represent the most relevant regarding normal usage. It consists of the terrain and the buildings. The two latter are looking at buildings with possible performance improvements, either by reducing the texture size or by using a color instead of a texture.

The FPS measurements are done both with and without the Oculus Rift, and the web site was refreshed between each set of measurements. We start by looking at the measurements without Oculus Rift.

4.1.2.1 Without Oculus Rift

State 1:

State 1 is at the start position and rotation, where the camera looks at the virtual world from the ground. Figure 29 displays the view, which contains several buildings.

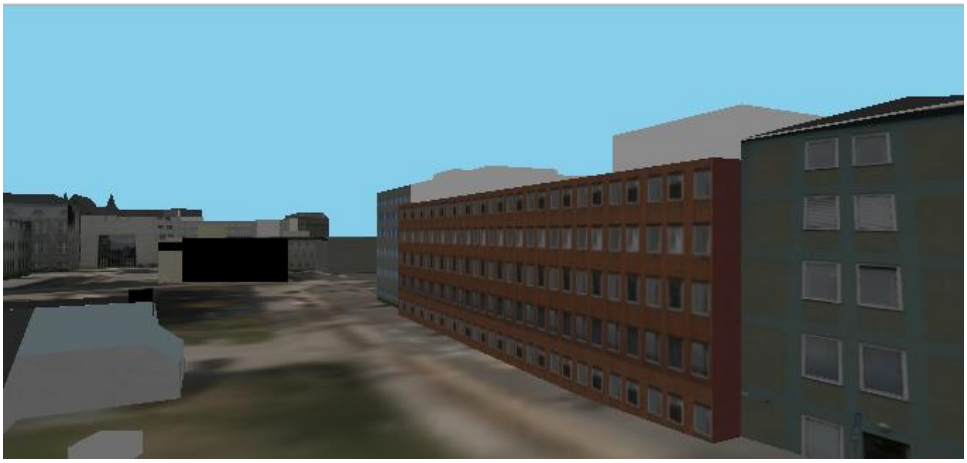


Figure 29: Rendering of state 1

	State1			
	Computer1		Computer2	
	Average	SD	Average	SD
Terrain(Ortophoto + local heights)	59.9	0.2	60.0	0.2
Terrain + buildings	24.3	0.7	25.8	1.2

Terrain + buildings with shrunken textures	24.7	0.2	24.1	0.6
Terrain + buildings with grey walls	59.9	0.1	60.0	0.0

Table 4: FPS in state 1 without Oculus Rift

	State1			
	Computer 2 with updated Chrome		Computer3	
	Average	SD	Average	SD
Terrain + buildings	28.8	0.5	36.4	1.1

Table 5: Extra measurements of FPS in state 1

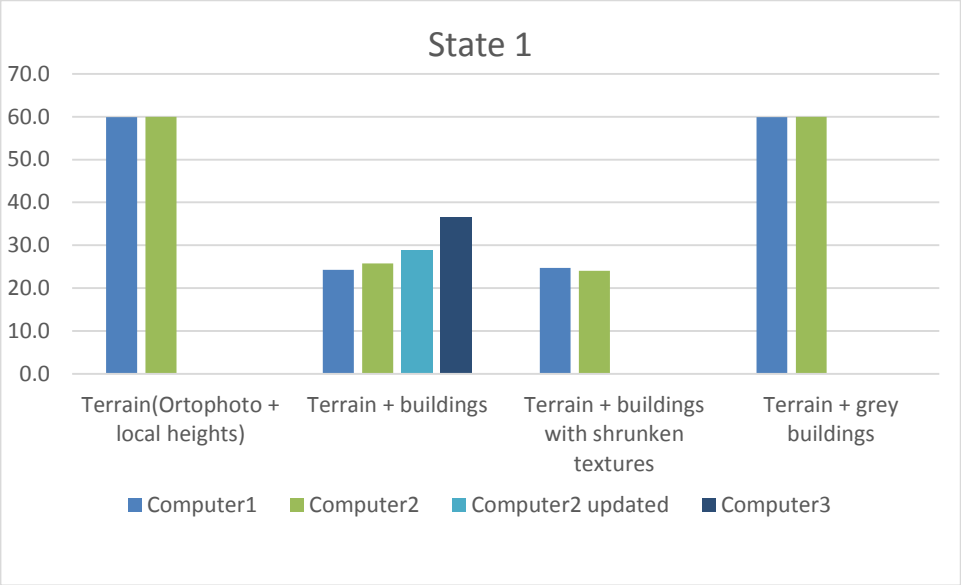


Figure 30: Results from state 1 without Oculus Rift

The configuration with only terrain resulted in 60 FPS, the maximum frame rate. This is also the case when rendering terrain and grey buildings. Not surprisingly, buildings with or without textures have a big impact on the performance. Here, the frame per seconds ranged from 24 to 36 depending on the computer. However, textures with reduced size does not appear to have a noticeable effect. The few results from computer 3, suggest that it gives the best performance. The computer 1 and 2 have similar results. After an update of the web browser in computer 2, the results became a little better.

State 2:

State 2 is a perspective from the air (Figure 31), where the camera looks down on the buildings. As a result, state 2 displays more buildings than state 1. This gives reason to believe that the frame per seconds will be lower.

Figure 31: Rendering of state 2

	State2			
	Computer1		Computer2	
	Average	SD	Average	SD
Terrain(Ortophoto + local heights)	60.0	0.0	59.9	0.1
Terrain + buildings	17.3	0.8	14.6	0.5
Terrain + buildings with shrunken textures	17.6	0.3	16.1	1.1
Terrain + buildings with grey walls	59.6	0.2	60.0	0.0

Table 6: FPS in state 2 without Oculus Rift

	State2			
	Computer 2 with updated Chrome		Computer3	
	Average	SD	Average	SD
Terrain + buildings	19.7	0.1	26.6	1.5

Table 7: Extra measurements in state 2

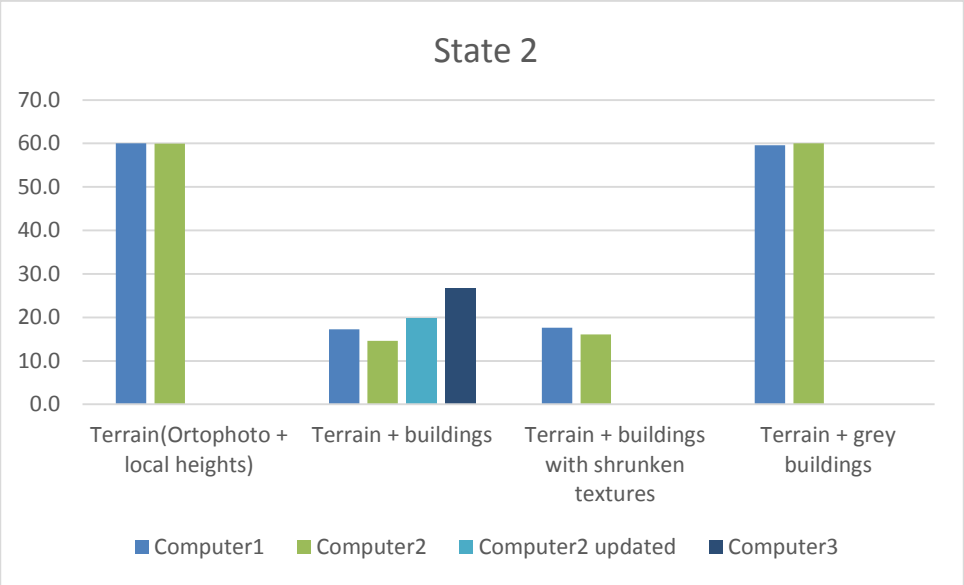


Figure 32: Results from state 2 without Oculus Rift

The two configurations that performs best did not suffer any cut in the FPS. This is reasonable since the maximum frame per seconds is not set by the operations in the program, but by the refresh rate setting on the computers. As suspected, the other configurations performs worse. As with state 1, computer 3 continues to outperform computer 1 and computer 2. Here computer 2 performs slightly worse than computer 1. This is not the case after the web browser update.

Flythrough

The flythrough starts close to the start position, at the ground, and moves through parts of the virtual world. The camera is looking from the ground perspective most of the time, but ends of in the air. On parts of the journey, fewer buildings are visible compared to state 1. This suggest that the performance should be similar to state 1, possibly better.

	Flythrough			
	Computer1		Computer2	
	Average	SD	Average	SD
Terrain(Ortophoto + local heights)	60.0	0.1	59.9	0.1
Terrain + buildings	28.3	1.1	25.9	1.4
Terrain + buildings with shrunken textures	28.8	0.3	27.8	0.6
Terrain + buildings with grey walls	60.0	0.0	59.9	0.1

Table 8: FPS for flythrough without Oculus Rift

	Flythrough			
	Computer 2 with updated Chrome		Computer3	
	Average	SD	Average	SD
Terrain + buildings	29.4	0.7	40.2	2.0

Table 9: Extra measurements from flythrough

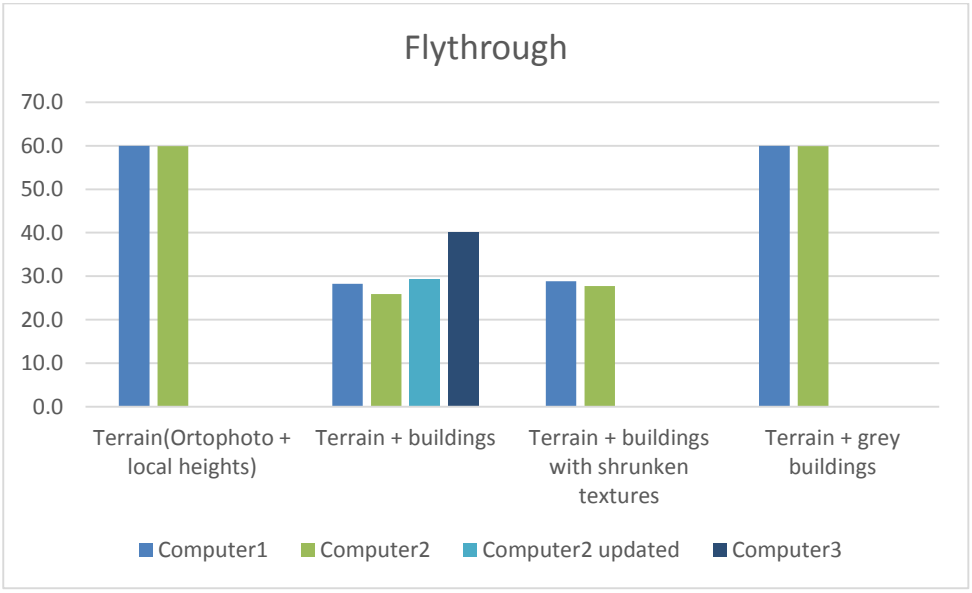


Figure 33: Flythrough without Oculus Rift

The values from Figure 33 displays similar results compared to state 2 regarding the relative performance between the computers. The performance in general appear to be better than both state 1 and state 2. Figure 34, which compares the different states, makes the different clearer. It contains an average of the all the computers for each configuration. The figure also indicates that the flythrough has better performance than state 1.

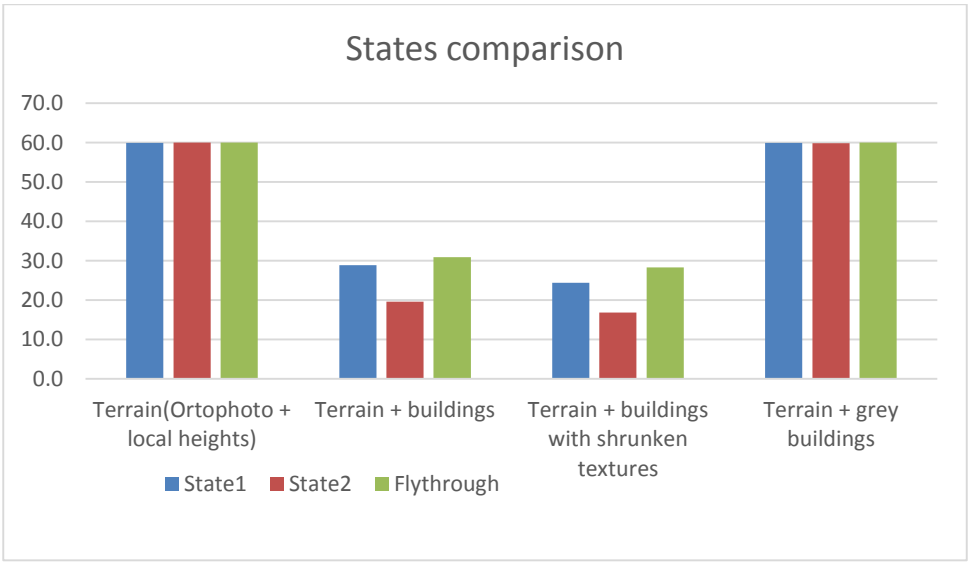


Figure 34: States comparison

4.1.2.2 With Oculus Rift:

The following sections contains the measurements in the mode for Oculus Rift. It contains fewer results than the last section since the testing was time consuming. In addition, some tests give a small amount of benefit. For instance, the relative results between buildings with or without shrunken textures should follow a similar pattern with the Oculus Rift. The differences on state 1, state 2 and flythrough is reviewed in the section without Oculus Rift, and the measurements with Oculus Rift does not appear to give more insight. Because of this, the graphs and discussion focus on the differences between the mode with and without Oculus Rift.

	State 1			
	Computer1		Computer2	
	Average	SD	Average	SD
Terrain(Ortophoto + local heights)	59.7	0.3	59.8	0.0
Terrain + buildings	14.1	0.2	14.6	1.1

Table 10: FPS in state 1 with Oculus Rift

	State 2			
	Computer1		Computer2	
	Average	SD	Average	SD
Terrain(Ortophoto + local heights)	59.9	0.1	59.9	0.1
Terrain + buildings	9.1	0.3	9.1	1.0

Table 11: FPS in state 2 with Oculus Rift

	Flythrough			
	Computer1		Computer2	
	Average	SD	Average	SD
Terrain(Ortophoto + local heights)	60.0	0.0	59.9	0.1
Terrain + buildings	15.6	0.3	15.1	2.1

Table 12: FPS for flythrough with Oculus Rift

The configuration with only terrain still gives the high frame rate in all the different cases, while the latter configuration (terrain and building) have been lowered quite much compared to the test without Oculus Rift. The differences are clear when looking at Figure 35, Figure 36 and Figure 37. In all of them, the Oculus Rift mode results in approximate half of the frame rate. In state 2, the performance is lowest with 9 FPS.

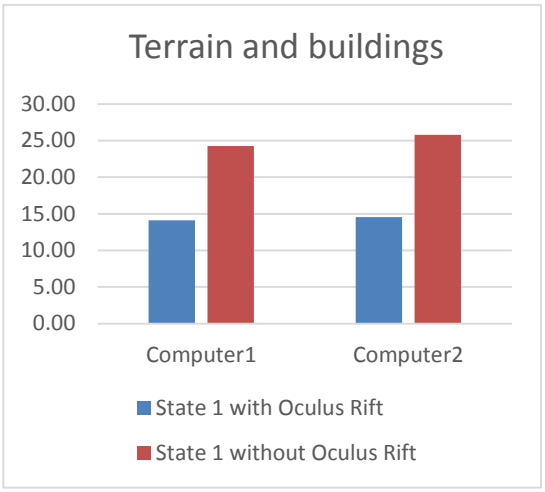


Figure 35: State 1 with or without Oculus Rift

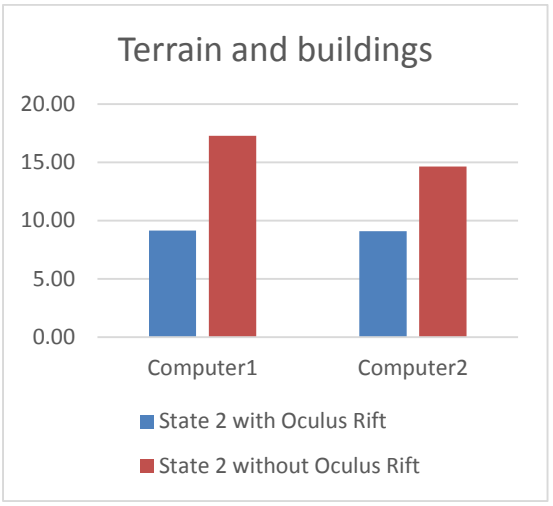


Figure 36: State 2 with or without Oculus Rift

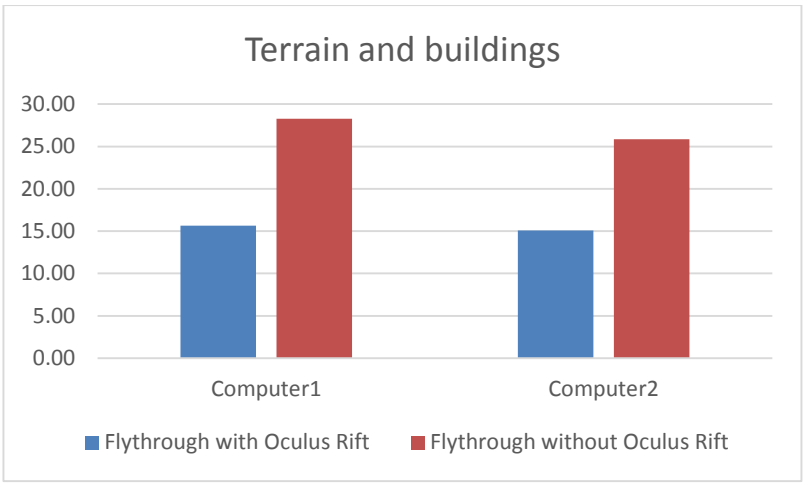


Figure 37: Flythrough with or without Oculus Rift

4.1.3 Delay in the oculus rift:

Regarding the delay of head tracking data, no quantitative data was collected. In order to get accurate quantitative data, i.e. delay in milliseconds, Latency Tester(98) or other equipment is needed. The experience in the Oculus Rift was twitchy when both terrain and buildings were rendered, even though this effect could not be seen at the computer screen. On rendering with only terrain, this unfortunate effect was smaller.

4.2 Visual results

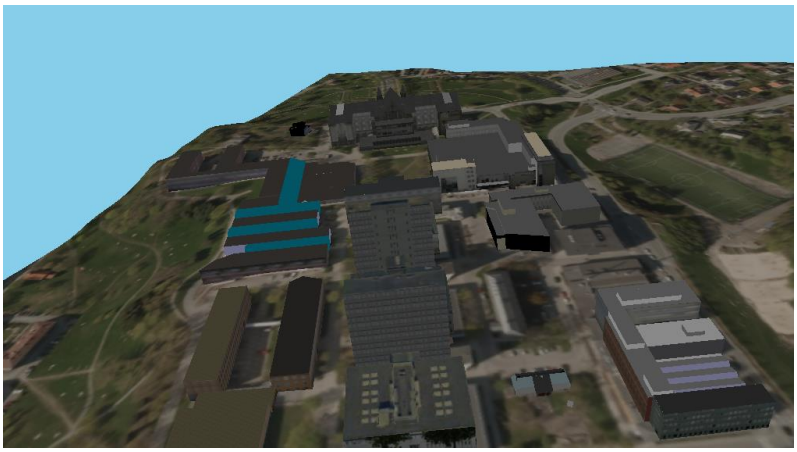


Figure 38: Gløshaugen seen from the air

The program presents terrain with texture, either orthophoto or map, with most of the buildings at Gløshaugen. The background is light sky blue, including below the terrain. The program gives the opportunity to look from the ground and from the air. It is two different modes, one for and one without Oculus Rift.

Figure 38 displays a part of Gløshaugen seen from the air. It looks mostly as it should, but there are a couple of issues. Some black walls is visible because of the issue with GitHub Pages (mentioned in the implementation chapter). The large building in the upper left, “Elektrobygget”, is placed to low due to an assumption in the implementation. In addition, the view would have looked better if all the buildings had been added.

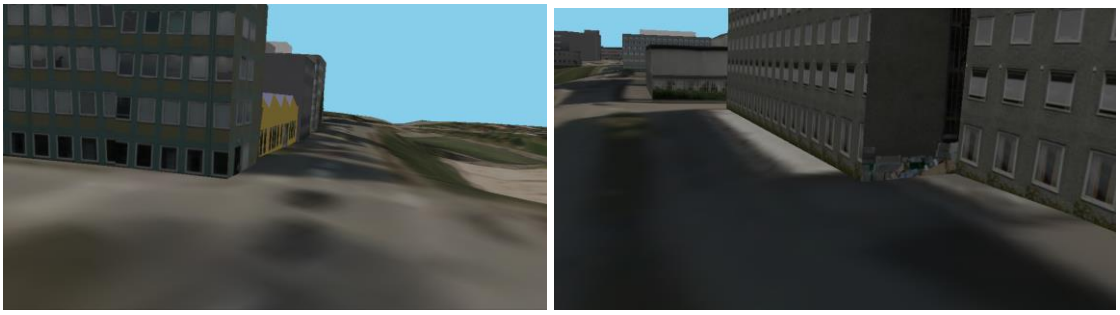


Figure 39: Building seen from the ground

Figure 39 shows an image from the ground. Here the quality of the existing buildings becomes more visible. The terrain texture is quite blurry which suggests that a higher resolution would have been more appropriate. In addition, varying quality of given coordinates becomes more apparent from the ground. One building, in Figure 40, also suggest that the conversion process in Blender is not flawless.

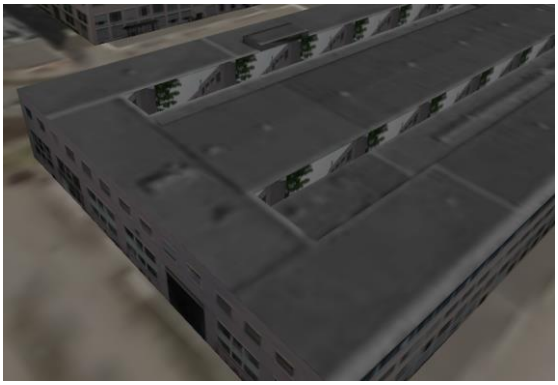


Figure 40: Unexpected wall texture

Figure 41 displays how it the computer screen looks like in the mode for Oculus Rift.

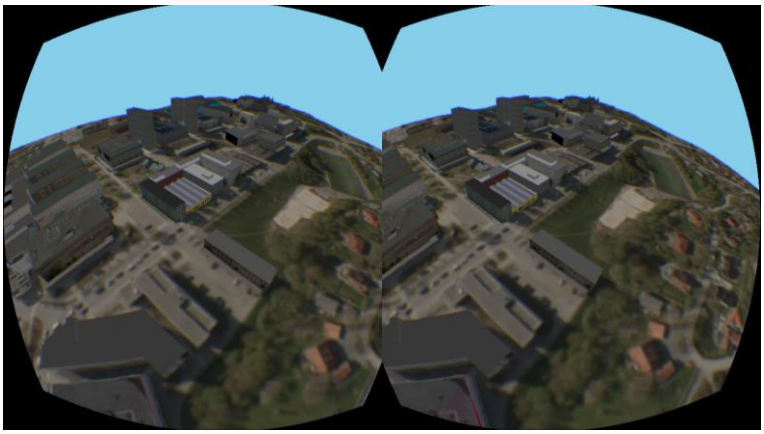


Figure 41: Gløshaugen seen from Oculus Rift

Uneven terrain or issues with the model itself give some occurrences where the ground of the building does not match the terrain. In addition, one of the buildings is placed wrong in the original file; the bottom of the building is not at the “ground”, which results in a flying building.

As seen in Figure 42, the configuration with lower resolution textures on the buildings gives a visible change in the appearance. This effect is unfortunate when being close to a building. If it had increased the performance, it could have worked well on buildings far away from the camera.

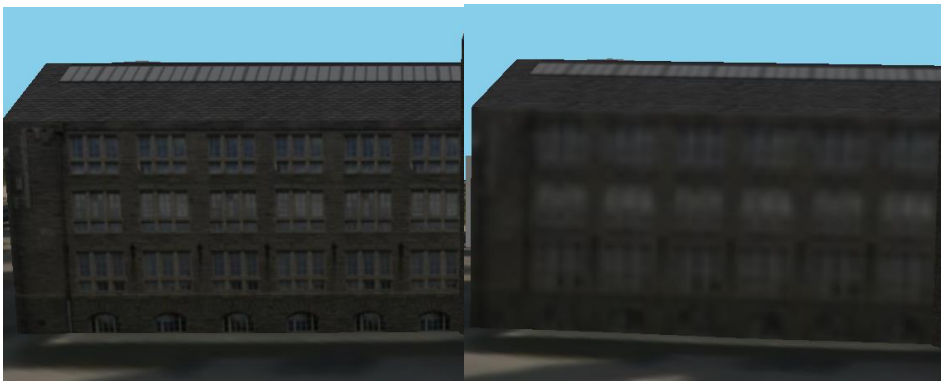


Figure 42: Comparison of texture resolutions. The original is at the left, while the lower resolution(1/4 in width and height) is at the right

The rendering without textures does not give a very appealing visual result (Figure 43). The shapes would have been more distinct with added shadows.

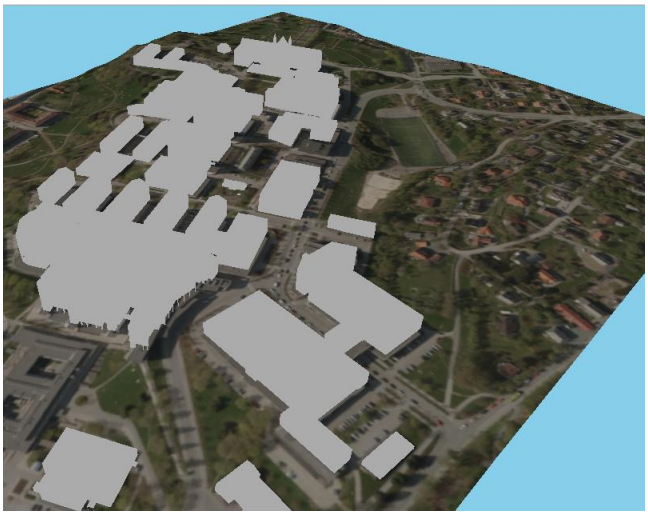


Figure 43: Rendering without textures

4.3 Navigation

The user moves in this virtual world by using the keyboard. The user can fly back, forward, left, right, up and down. In addition, it is possible to rotate is the yaw, pitch and roll axis. The collision detection prevents the user from flying through buildings.

Since the navigation is based on flying, it is difficult to navigate in a walking manner. When using the Rift, the keyboard cannot be seen, which makes it harder. Unless the user has good control over the location of different keys, the navigation will be troublesome. The fly through feature is an alternative navigation method that guides the user through an earlier recorded trip.

5 Discussion

5.1 Performance

Performance is an important factor for deciding if the program is practically to use. The results described three different elements within performance.

The startup time on about 15 seconds is noticeable, but there is much data to load. Retrieving the textures to the buildings is the most time demanding task. The terrain texture retrieved via WMS also takes a considerable time, although it does not have a big effect the total time. The loading time could perhaps improve if the buildings were loaded from more than one source. At the same time, a slow internet connection will increase the load time.

The frames per seconds, after all the resources are loaded, with and without Oculus Rift is more important. When only rendering the terrain, the FPS is as high as possible since the computer settings limits the FPS to 60. With buildings, computer 2 had disappointing results compared to the computer 1 since computer 2 appear to have much better hardware. Perhaps computer 1 has an advantage of being newer, since the WebGL technology is quite new. Upgrading the browser on computer 2 helped to some degree, which illustrates the importance of updated software when using new technology. The third computer performed best with a FPS about 50 percent better than computer 1. The disappointing results on computer 2, was the reason behind making a few test with computer 3.

As expected, the view from the camera influences the performance. More information is rendered from the aerial view in state2 compared to state1 which were a ground perspective is rendered. The fly through flies mainly on the ground, and performs better than state 2. Based on the tests, it is clear that the building textures is the biggest challenge. Rendering the building with grey color instead of textures gave a performance as good as only with terrain. Since rendering building with only one color is far from ideal, another approach were looked at. By shrinking the images, it was hoped that the performance would improve, but also here the results were disappointed. It is unclear why, but the results did only seem to approve slightly if anything. Since the process of changing all the images is cumbersome, it was not further investigated. However, it is possible that a combination of shrinking, and compressing would cause a better effect. At the same time, the texture cannot be simplified too much without ruining the buildings appearance. Even though the FPS could have been higher, the consequences of it depends on different factors. Speed is important in this case. Compared to games like first person shooter, the movement and rotation speed is quite low. This reduces the perceived effect of a low FPS.

When using the Rift, the frames per second became even lower. With 9 FPS on the lowest, it starts to become quite low. The rendering of the terrain alone gives a good performance, but all the buildings has a big effect. The FPS is close to half of what is was without the Rift. The twitchy experience with the Rift suggests that a performance increase would be quite valuable.

5.2 Visual results

The results give a good idea about the appearance of Gløshaugen. If all the buildings had been added, it would have been even better. A user can get an overview from the ground are a more detailed look from the ground. Some limitations are based on the source of the building data. As always, the quality of input affects the output. Improvements could be made so that the buildings corresponds better to the terrain, both vertically and horizontally. The former requires consideration of uneven terrain. The latter case requires either the georeferencing in the original source to improve, or to implement support for moving of the buildings within the website.

Overall, this solution like can work well to present how an area look like, or how a new building will affect the surroundings, but not to show a very accurate representation of the real world.

The visualization with configurations for better performance is not as appealing, which shows that improving one aspect may affect another aspect negatively.

5.3 Navigation

Since the user fly in the virtual world, the movement possibilities are quite good. However, it is hard to replicate the feeling of walking. At the same time, flying can give another perspective of how an area look like. The walking perspective is not better than the flying perspective in itself, it depends of the usage. The optimal would be to have the option to switch between both.

5.4 Questions

The following questions have been looked at. The answers are not the only possible solution, but suggestions, which reflect some of the experiences from this project.

- *How can a combination of terrain and 3D-models be visualized in a semi-automatic procedure using open source software?*

As shown in the chapter Frameworks, there is a lot of open source software within 3D visualization of some kind. The source of the terrain and buildings, and the framework used will affect how the process from collecting data to the visualization will look like.

Terrain data from Norway is available for download through NMA website(40), but this requires manual interaction such as registration and choosing of area. In addition, the boundaries of the download area needs to be clipped and converted to a more fitting format. This can be taken care of by a program like GDAL. A more flexible method is to use WCS to request and receive terrain data, in both a fitting area and format, through the internet. For offline use, these data can be stored locally.

The building format and structure affects how easy it is to use. If the chosen framework supports the format, and no incompatibility problems arises, then the process becomes a lot easier. However, if performance is a problem, then changing format or editing the model could improve it. Looking at the buildings in this project, the situation was not ideal. The two original data formats is very specific and therefore not supported. An attempt make to the conversion process automatic was made, but the result was varying. As described in the implementation, more effort on the problem would have made it more robust. If models is needed from several different sources, each with different format, it will be problematic if many of formats is not directly supported. This means that models created in much used program and stored in a much used and user friendly format is necessary if least possible effort (both manually work, and implementation time) is needed.

- *Which interactions methods suits the navigation in this kind of virtual reality?*

In virtual reality based on head mounted devices where you, like Oculus Rift, the view to the real world is blocked. As a result, navigation by using real world objects becomes harder. A joystick has the advantage of being easier to use when blinded. Although, the joystick might not feel natural for people with little gaming experience. If walking is supported within the program, a solution like Virtuix Omni(described in the section about Oculus Rift) sounds like an excellent solution. Then both navigation and observation becomes quite similar to the real thing. Flying in a virtual world is different and since we cannot fly in the real world, it is harder to make the navigation feels truly naturally. Perhaps moving by using the body, e.g. leaning forward or backward, would work.

- *How can other types of geographical data expand the proposed virtual world?*

This 3D model contains terrain with texture and manually made buildings. Less detailed buildings can be generated from SOSI-files (Norwegian file format). Laser files with detailed buildings or other objects, both natural and manufactured is another alternative. However, laser files can become quite big. As a result, loading them through the internet will take longer time, and hosting costs may increase. The render performance could also be an issue. Addresses is another expansion possibility, although they is probably more useful when a larger area is covered. Regarding the rendering of addresses, they do not have to be drawn on the terrain. Any decision her needs to consider how it blends with the rest of the data. There is a lot options, and too much information could easily be a problem.

6 Conclusion

In this master thesis, different frameworks have been investigated. One of those, three.js, have been chosen to make a virtual world over Gløshaugen. This world contains several buildings that students at NTNU have made. There is a mode with Oculus Rift, the new device within virtual reality, and without.

Performance is an issue when using the Rift. Without the Rift, the performance was more acceptable. The buildings textures is responsible for the performance problems, but the attempt to increase it by using smaller texture sizes did not succeed. The rendering of the buildings without the textures gave a large effect on the performance, but did not give a visually pleasing result. The web browser version influenced the performance and one of the test computers performed better than the others did.

The visual impression is acceptable for showcases, but not for a high accuracy representation of an area. This is both due to varying quality of the existing buildings, including georeferencing, but also because of a few bugs in the program. The keyboard is used for navigation, which works, but it is not ideal.

Some related questions were also looked at and discussed briefly.

7 Further work

In the current version, parameters is given through the URL. An improvement would be to have graphic interface instead. Then it will be more user friendly, and possible to change the configuration without having to reload to website.

The program would benefit from a higher performance. Improvement the texture preprocessing may help with this. Refactoring of the code and more exploration of the three.js library could potentially lead to solutions that are more efficient. For instance, three.js has a level of detail feature that could to worth looking into.

The two other Oculus Rift integrations alternatives could be implemented to see which differences that exist. Perhaps the performance in the Oculus mode will increase.

The automated process from the KMZ format to the formats used by three.js can be improved. It can also be extended to convert other formats.

Within navigation, it would be nice to be able to walk instead of flying. This could potentially be quite cumbersome, but at the same time achievable. Support for other input devices than the keyboard can also make the navigation easier.

The available terrain through WMS and WCS makes it possible to make the virtual world larger, either by requesting data from them on the fly or in advance.

8 References

1. Cruz-Neira C. Virtual Reality Overview. SIGGRAPH'93 Course, No. 23, pp. 1.1-1.18. 1993.

2. Boas Y. Overview of Virtual Reality Technologies. mms.ecs.soton.ac.uk. 2013.

3. Virtual Reality - History, Applications, Technology and Future.

4. Bowman DA, Johnson DB, Hodges LF. Testbed Evaluation of Virtual Environment Techniques Interaction. 1999. p. 26–33.

5. Usoh M, Arthur K, Whitton MC, Bastos R, Steed A, Slater M, et al. Walking > walking-in-place > flying, in virtual environments. Proc 26th Annu Conf Comput Graph Interact Tech - SIGGRAPH '99. New York, New York, USA: ACM Press; 1999;359–64.

6. Jr JJL, Keefe DF, Zeleznik RC. Hands-Free Multi-Scale Navigation in Virtual Environments. Proceedings of the 2001 symposium on Interactive 3D graphics. 2001. p. 9–15.

7. Darken RP, Peterson B. Spatial orientation, wayfinding, and representation. Handb virtual Environ. 2002;493–518.

8. Heilig ML. Sensorama simulator. US; Patent No 3,050,870, 1962.

9. Zachara M, Zagal JP. Challenges for Success in Stereo Gaming : A Virtual Boy Case Study. 2009;99–106.

10. Brunner G. Virtually perfect: The past, present, and future of VR [Internet]. ExtremeTech. [cited 2014 May 31]. Available from: http://www.extremetech.com/gaming/181454-virtually-perfect-the-promise-of-virtual-reality

11. Sony Announces New Personal LCD Monitor PC Glasstron [Internet]. Sony Corporation. [cited 2014 Jun 6]. Available from: http://www.sony.net/SonyInfo/News/Press_Archive/199809/98-101/

12. Sony "Glasstron" PLM-A35 LCD Eyeglasses [Internet]. Steve's Digicam Online, Inc. [cited 2014 May 28]. Available from: http://www.steves-digicams.com/glasstron.html

13. Hoffman HGP, Patterson DRP, Seibel EP, Soltani MMe, Jewett-Leahy LB, Sharar SRM. Virtual Reality Pain Control During Burn Wound Debridement in the Hydrotank. Clin J Pain. 2008;24(4):299–304.

14. Garcia-Palacios A, Hoffman H, Carlin A, Furness T., Botella C. Virtual reality in the treatment of spider phobia: a controlled study. Behav Res Ther. 2002 Sep;40(9):983–93.

15. Kickstarters [Internet]. Kickstarter, Inc. [cited 2014 Apr 30]. Available from: https://www.kickstarter.com/

16. Kickstarter: Oculus Rift [Internet]. Kickstarter, Inc. [cited 2014 Apr 30]. Available from: https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game

17. Solomon B. Facebook Buys Oculus, Virtual Reality Gaming Startup, For $2 Billion [Internet]. Forbes. [cited 2014 Jun 3]. Available from: http://www.forbes.com/sites/briansolomon/2014/03/25/facebook-buys-oculus-virtual-reality-gaming-startup-for-2-billion/

18. Oculus CEO clarifies: one Oculus Rift headed to consumers, supports Android and PC [Internet]. engadget. 2013 [cited 2014 Apr 30]. Available from: http://www.engadget.com/2013/11/01/oculus-rift-consumer-version-interview/

19. Stabinger S. Oculus Rift - Developer Version - Back and Control Box [Internet]. Wikipedia. [cited 2014 Apr 30]. Available from: http://en.wikipedia.org/wiki/File:Oculus_Rift_-_Developer_Version_-_Back_and_Control_Box.jpg

20. Antonov M, Mitchell N, Reisse A, Cooper L, LaValle S, Katsev M. Oculus Software Development Kit Overview v0.2.5c. Oculus VR; 2013.

21. App & game list [Internet]. RiftEnabled. [cited 2014 Apr 28]. Available from: http://www.riftenabled.com/admin/apps/

22. List of games with Oculus Rift support [Internet]. Wikipedia. [cited 2014 Jun 9]. Available from: http://en.wikipedia.org/wiki/List_of_games_with_Oculus_Rift_support

23. Virtuix Omni [Internet]. Virtuix Inc. [cited 2014 Apr 30]. Available from: http://shop.virtuix.com/

24. Kickstarter Omni: Move Naturally in Your Favorite Game by Virtuix [Internet]. Kickstarter, Inc. [cited 2014 Apr 30]. Available from: https://www.kickstarter.com/projects/1944625487/omni-move-naturally-in-your-favorite-game

25. MVN BIOMECH Awinda [Internet]. Xsens. [cited 2014 Apr 30]. Available from: http://www.xsens.com/products/mvn-biomech-awinda/

26. Forsyth T. VR Sickness, The Rift, and How Game Developers Can Help [Internet]. Oculus VR. [cited 2014 Apr 30]. Available from: http://www.oculusvr.com/blog/vr-sickness-the-rift-and-how-game-developers-can-help/

27. Byford S. Sony reveals Project Morpheus, its VR headset for PlayStation 4 [Internet]. The Verge. [cited 2014 Apr 28]. Available from: http://www.theverge.com/2014/3/18/5523984/sony-reveals-project-morpheus-its-vr-system-for-ps4

28. Musialski P, Wonka P, Aliaga DG, Wimmer M, van Gool L, Purgathofer W. A Survey of Urban Reconstruction. Comput Graph Forum. 2013 Sep 10;32(6):146–77.

29. Shan, Qi and Adams, Riley and Curless, Brian and Furukawa, Yasutaka and Seitz SM. The Visual Turing Test for Scene Reconstruction. 3DTV-Conference, 2013 International Conference on. IEEE; 2013. p. 25–32.

30. Flickr [Internet]. [cited 2014 Jun 3]. Available from: https://www.flickr.com/

31. Poullis C, You S. 3D Reconstruction of Urban Areas. 2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission. IEEE; 2011. p. 33–40.

32. Brian McClendon. The never-ending quest for the perfect map [Internet]. Google's Official Blog. 2012 [cited 2014 May 25]. Available from: http://googleblog.blogspot.no/2012/06/never-ending-quest-for-perfect-map.html

33. 3D and satellite imagery availability [Internet]. Google. [cited 2014 Jun 1]. Available from: https://support.google.com/earth/answer/2789536

34. Om Norge digitalt-samarbeidet [Internet]. The Norwegian Mapping Authority. [cited 2014 Jun 2]. Available from: http://www.kartverket.no/Geonorge/Norge-digitalt/Om-Norge-digitalt-samarbeidet/

35. Den geografiske infrastrukturen [Internet]. The Norwegian Mapping Authority. [cited 2014 Jun 2]. Available from: http://www.statkart.no/Geonorge/Den-geografiske-infrastrukturen/

36. Nasjonale kartdata blir gratis [Internet]. Norwegian Mapping Authority. 2013 [cited 2014 May 1]. Available from: http://www.statkart.no/Om-Kartverket/Nyheter/Nasjonale-kartdata-blir-gratis/

37. Miljøverndepartementet. Strategi for åpne kart- og eiendomsdata. 2013.

38. Gratis kartdata lønner seg [Internet]. Kommunal- og moderniseringsdepartementet. [cited 2014 Jun 2]. Available from: http://www.regjeringen.no/nb/dep/kmd/pressesenter/pressemeldinger/2014/Gratis-kartdata-lonner-seg.html?id=754369

39. Vennemo H, Ibenholt K, Magnussen K, Moen E, Riis C. Verdien av gratis kart- og eiendomsdata. 2014.

40. NMA's downloading site [Internet]. Norwegian Mapping Authority. [cited 2014 May 1]. Available from: http://data.kartverket.no/download/

41. SketchUp [Internet]. Trimble Navigation Limited. Available from: http://www.sketchup.com/

42. Blender [Internet]. Blender Foundation. [cited 2014 May 15]. Available from: http://www.blender.org/

43. Python [Internet]. Python Software Foundation. [cited 2014 Jun 2]. Available from: https://www.python.org/

44. Digital Elevation Model Standards [Internet]. U.S. Geological Survey. [cited 2014 May 7]. Available from: http://nationalmap.gov/standards/demstds.html

45. U.S. Geological Survey [Internet]. [cited 2014 May 7]. Available from: http://www.usgs.gov/

46. GDAL - Geospatial Data Abstraction Library [Internet]. [cited 2014 May 7]. Available from: http://www.gdal.org/

47. QGIS [Internet]. [cited 2014 May 7]. Available from: http://www.qgis.org/en/site/

48. WebGL - OpenGL ES 2.0 for the Web [Internet]. Khronos Group. [cited 2014 May 15]. Available from: https://www.khronos.org/webgl/

49. Khronos Group [Internet]. [cited 2014 May 15]. Available from: https://www.khronos.org/

50. Web Map Service [Internet]. Open Geospatial Consortium. [cited 2014 May 6]. Available from: http://www.opengeospatial.org/standards/wms

51. OpenGIS ® Web Map Server Implementation Specification version 1.3.0. Open Geospatial Consortium; 2006.

52. Open Geospatial Consortium [Internet]. [cited 2014 May 6]. Available from: http://www.opengeospatial.org/

53. Web Coverage Service [Internet]. Open Geospatial Consortium. [cited 2014 May 19]. Available from: http://www.opengeospatial.org/standards/wcs

54. UDK [Internet]. Epic Games Inc. [cited 2014 Feb 11]. Available from: https://www.unrealengine.com/products/udk/

55. Unity [Internet]. Unity Technologies. [cited 2014 Feb 11]. Available from: https://unity3d.com/

56. Other Engine Integrations [Internet]. Oculus VR. [cited 2014 Feb 28]. Available from: https://developer.oculusvr.com/forums/viewforum.php?f=39

57. List of game engines [Internet]. Wikipedia. [cited 2014 Feb 19]. Available from: http://en.wikipedia.org/wiki/List_of_game_engines

58. OpenSceneGraph [Internet]. [cited 2014 Mar 4]. Available from: http://www.openscenegraph.org/

59. Collada Plugin [Internet]. OpenSceneGraph. [cited 2014 Feb 25]. Available from: http://trac.openscenegraph.org/projects/osg/wiki/Support/KnowledgeBase/Collada

60. Installing OSG on Windows with MinGW and Eclipse (including COLLADA) [Internet]. OpenSceneGraph. [cited 2014 Feb 25]. Available from: http://trac.openscenegraph.org/projects/osg/wiki/Support/PlatformSpecifics/MingwColladaEclipse

61. Pavlik R. sketchupToOSG [Internet]. GitHub repository. [cited 2014 Feb 25]. Available from: https://github.com/rpavlik/sketchupToOSG

62. Blissing B. osgoculusviewer [Internet]. GitHub repository. [cited 2014 Feb 25]. Available from: https://github.com/bjornblissing/osgoculusviewer

63. Martz P, ameslab, ARDEC. osgBullet [Internet]. Google Project Hosting. [cited 2014 Feb 25]. Available from: https://code.google.com/p/osgbullet/

64. Erwincoumans, 2rtrius. bullet3 [Internet]. GitHub repository. [cited 2014 May 9]. Available from: https://github.com/bulletphysics/bullet3

65. three.js [Internet]. [cited 2014 May 12]. Available from: http://threejs.org/

66. GitHub [Internet]. [cited 2014 May 29]. Available from: https://github.com/

67. Knip T, Parisi T. ColladaLoader.js (library) [Internet]. GitHub repository. [cited 2014 Mar 28]. Available from: https://github.com/mrdoob/three.js/blob/master/examples/js/loaders/ColladaLoader.js

68. Sandvik B. thematic mapping blog [Internet]. [cited 2014 May 12]. Available from: http://blog.thematicmapping.org/

69. three.js: examples [Internet]. [cited 2014 May 12]. Available from: http://threejs.org/examples/

70. Sandvik B. Terrain in THREE.js [Internet]. thematic mapping blog. [cited 2014 May 11]. Available from: http://thematicmapping.org/playground/webgl/terrain/texture/jotunheimen.html

71. Ryan Spangler, Ben Purdy. oculus-bridge [Internet]. GitHub repository. [cited 2014 Mar 28]. Available from: https://github.com/Instrument/oculus-bridge

72. Siciliano(troffmo5) L. OculusRiftEffect [Internet]. Github; [cited 2014 Mar 28]. Available from: https://github.com/mrdoob/three.js/blob/master/examples/js/effects/OculusRiftEffect.js

73. Vanik B. vr.js [Internet]. GitHub repository. [cited 2014 Feb 17]. Available from: https://github.com/benvanik/vr.js

74. Jernberg(possan) P-O. THREE.OculusControls [Internet]. GitHub. [cited 2014 Feb 17]. Available from: https://github.com/mrdoob/three.js/blob/master/examples/js/controls/OculusControls.js

75. jMonkeyEngine [Internet]. [cited 2014 Feb 24]. Available from: http://jmonkeyengine.org/

76. NetBeans IDE [Internet]. [cited 2014 May 12]. Available from: https://netbeans.org/

77. jMonkeyEngine 3 Tutorial (10) - Hello Terrain [Internet]. [cited 2014 Feb 13]. Available from: http://hub.jmonkeyengine.org/wiki/doku.php/jme3:beginner:hello_terrain

78. jmonkeyengine-oculus-rift discussion [Internet]. JMonkeyEngine. [cited 2014 Mar 4]. Available from: http://hub.jmonkeyengine.org/forum/topic/oculus-rift-support/

79. Google trends comparison [Internet]. Google. [cited 2014 May 8]. Available from: www.google.no/trends/explore#geo&q=OpenSceneGraph,+THREE.js,+jMonkeyEngine&cmpt=q

80. Jasmine [Internet]. GitHub Pages. [cited 2014 Apr 21]. Available from: http://jasmine.github.io/2.0/introduction.html

81. Fredheim SH. Gloshaugen3D repository [Internet]. GitHub repository. [cited 2014 Jun 3]. Available from: https://github.com/shf123/Gloshaugen3D

82. GitHub pages [Internet]. Github, Inc. [cited 2014 Jun 3]. Available from: https://pages.github.com/

83. Fredheim SH. Gloshaugen3D [Internet]. GitHub Pages. [cited 2014 Jun 3]. Available from: http://shf123.github.io/Gloshaugen3D/

84. three.js realeases [Internet]. GitHub repository. [cited 2014 May 20]. Available from: https://github.com/mrdoob/three.js/tags

85. jQuery [Internet]. The jQuery Foundation. [cited 2014 Mar 28]. Available from: http://jquery.com/

86. Sveen AF. holsen.js [Internet]. GitHub repository. [cited 2014 Mar 28]. Available from: https://github.com/atlefren/holsenjs

87. Alteredq, Cabello R. detector.js [Internet]. GitHub repository. [cited 2014 May 20]. Available from: https://github.com/mrdoob/three.js/blob/master/examples/js/Detector.js

88. Cabello R. stats.js [Internet]. GitHub repository. [cited 2014 Mar 28]. Available from: https://github.com/mrdoob/stats.js

89. gdalwarp [Internet]. GDAL. [cited 2014 Mar 11]. Available from: http://www.gdal.org/gdalwarp.html

90. gdal_translate [Internet]. GDAL. [cited 2014 Mar 11]. Available from: http://www.gdal.org/gdal_translate.html

91. Cabello R. Three.js Blender Import/Export [Internet]. GitHub repository. [cited 2014 Mar 21]. Available from: https://github.com/mrdoob/three.js/tree/master/utils/exporters/blender

92. 7-zip [Internet]. [cited 2014 Jun 3]. Available from: http://www.7-zip.org/

93. FastStone Photo Resizer [Internet]. FastStone Soft. [cited 2014 May 12]. Available from: http://www.faststone.org/FSResizerDetail.htm

94. CPU Benchmarks [Internet]. PassMark® Software. [cited 2014 May 27]. Available from: http://www.cpubenchmark.net

95. Videocard benchmarks [Internet]. PassMark® Software. [cited 2014 May 27]. Available from: http://www.videocardbenchmark.net/

96. PassMark PerformanceTest Easy PC benchmarking [Internet]. PassMark® Software. Available from: http://www.passmark.com/products/pt.htm

97. Mål din hastighet [Internet]. Bredbåndsguiden AS. [cited 2014 Jun 6]. Available from: http://speed.bredbandsguiden.no/

98. Latency Tester [Internet]. Oculus VR. [cited 2014 May 13]. Available from: https://www.oculusvr.com/order/latency-tester/

9 Appendices

9.1 Appendix A: Performance tables

The tables here have another structure compared to the tables in 4.1 Performance. The measurements are grouped based on computers instead of combinations or states.

Start-up times:

	Computer 1									
	Combination 1					Combination 2				
	Time1	Time2	Time3	Average	SD	Time1	Time2	Time3	Average	SD
DOM-Content loaded	2.61	0.981	1.26	**1.62**	**0.871**	1.36	0.945	1.1	**1.14**	**0.210**
Terrain heights	1.241	0.445	0.635	**0.77**	**0.416**	1.225	1.192	0.991	**1.14**	**0.127**
Terrain texture	3.88	1.27	1.57	**2.24**	**1.428**	6.88	6.42	8.22	**7.17**	**0.935**
Buildings without textures	1.972	2.21	2.102	**2.09**	**0.119**	1.589	1.813	1.704	**1.70**	**0.112**
Buildings with textures	11.798	10.496	16.649	**12.98**	**3.243**	16.803	10.078	17.032	**14.64**	**3.950**
Total time	15.63	11.91	18.53	**15.36**	**3.318**	19.37	12.2	19.11	**16.89**	**4.067**

	Computer 2									
	Combination 1					Combination 2				
	Time1	Time2	Time3	Average	SD	Time1	Time2*	Time3	Average	SD
DOM-Content loaded	1.35	1.47	1.43	**1.42**	**0.061**	1.47	1.27	1.28	**1.38**	**0.134**
Terrain heights	0.395	0.471	0.406	**0.42**	**0.041**	1.172	0.914	1.358	**1.27**	**0.132**
Terrain texture	0.729	0.634	0.493	**0.62**	**0.119**	4.33	3.26	2.98	**3.66**	**0.955**
Buildings without textures	1.643	1.432	1.423	**1.50**	**0.125**	1.453	1.23	1.284	**1.37**	**0.120**
Buildings with textures	12.838	12.555	13.34	**12.91**	**0.398**	12.926	5.21	12.189	**12.56**	**0.521**
Total time	14.57	14.48	15.16	**14.74**	**0.369**	15.55	7.38	14.81	**15.18**	**0.523**

Frame per seconds measurements without Oculus Rift:

Without Oculus Rift	Computer 1														
	State1					State2					Flythrough				
	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD
Terrain(Orthopho to + local heights)	60	60	59.6	**59.87**	**0.231**	60	60	60	**60.00**	**0.000**	60	59.9	60	**59.97**	**0.058**
Terrain + buildings	24	23.7	25.1	**24.27**	**0.737**	16.5	17.2	18.1	**17.27**	**0.802**	29.4	28.1	27.3	**28.27**	**1.060**
Terrain + buildings with shrunken textures	24.5	24.8	24.8	**24.70**	**0.173**	17.6	17.9	17.3	**17.60**	**0.300**	28.9	29.1	28.5	**28.83**	**0.306**
Terrain + buildings with grey walls	59.8	59.8	60	**59.87**	**0.115**	59.4	59.6	59.8	**59.60**	**0.200**	60	60	60	**60.00**	**0.000**

Without Oculus Rift	Computer 2														
	State1					State2					Flythrough				
	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD
Terrain(Ortophoto + local heights)	60.1	59.8	60	**59.97**	**0.153**	59.8	60	60	**59.93**	**0.115**	60.0	59.8	60	**59.93**	**0.115**
Terrain + buildings	26.6	24.4	26.4	**25.80**	**1.217**	14.2	14.5	15.2	**14.63**	**0.513**	26.2	24.3	27.1	**25.87**	**1.429**
Terrain + buildings with shrunken textures	23.6	24.5		**24.05**	**0.636**	15.3	16.8		**16.05**	**1.061**	27.3	28.2		**27.75**	**0.636**
Terrain + buildings with grey walls	60	60	60	**60.00**	**0.000**	60	60	60	**60.00**	**0.000**	59.9	60	59.9	**59.93**	**0.058**

Without Oculus Rift	Terrain and buildings														
	State1					State2					Flythrough				
	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD
Computer 2 with updated browser	29.3	28.7	28.4	**28.80**	**0.458**	19.6	19.8	19.8	**19.73**	**0.115**	29.6	28.6	29.9	**29.37**	**0.681**
Computer 3	35.4	36.3	37.6	**36.43**	**1.106**	27.7	24.9	27.2	**26.60**	**1.493**	40.2	42.2	38.2	**40.20**	**2.000**

Frame per seconds measurements with Oculus Rift:

With Oculus Rift	Computer 1														
	State1					State2					Flythrough				
	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD
Terrain(Ortophoto + local heights)	60	59.4	59.8	**59.73**	**0.306**	60	59.8	59.8	**59.87**	**0.115**	60	60	60	**60.00**	**0.000**
Terrain + buildings	13.9	14.2	14.3	**14.13**	**0.208**	9.16	8.81	9.4	**9.12**	**0.297**	15.4	15.6	15.9	**15.63**	**0.252**
Terrain + buildings with grey walls	60	59.6	60.1	**59.90**	**0.265**	59.8	59.8	59.8	**59.80**	**0.000**	59.9	59.9	59.9	**59.90**	**0.000**

With Oculus Rift	Computer 2														
	State1					State2					Flythrough				
	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD	FPS1	FPS2	FPS3	Average	SD
Terrain(Ortophoto + local heights)	59.8	59.8	59.8	**59.80**	**0.000**	59.9	59.9	59.8	**59.87**	**0.058**	59.8	59.9	59.9	**59.87**	**0.058**
Terrain + buildings	15.3	13.3	15.1	**14.57**	**1.102**	10.1	8.1	9	**9.07**	**1.002**	16.4	12.7	16.1	**15.07**	**2.055**

9.2 Appendix B: Running locally

The website can be run locally be using Python's server functionality. In this case, by running the command "python -m SimpleHTTPServer" from the base of the repository.