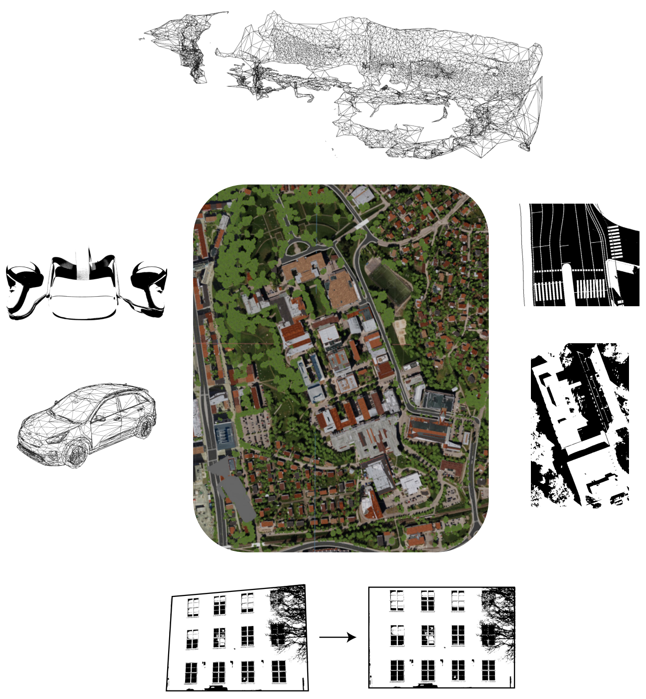
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Aleksander Knutsen

Gløshaugen's Digital Twin

Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2022



Aleksander Knutsen

Gløshaugen's Digital Twin

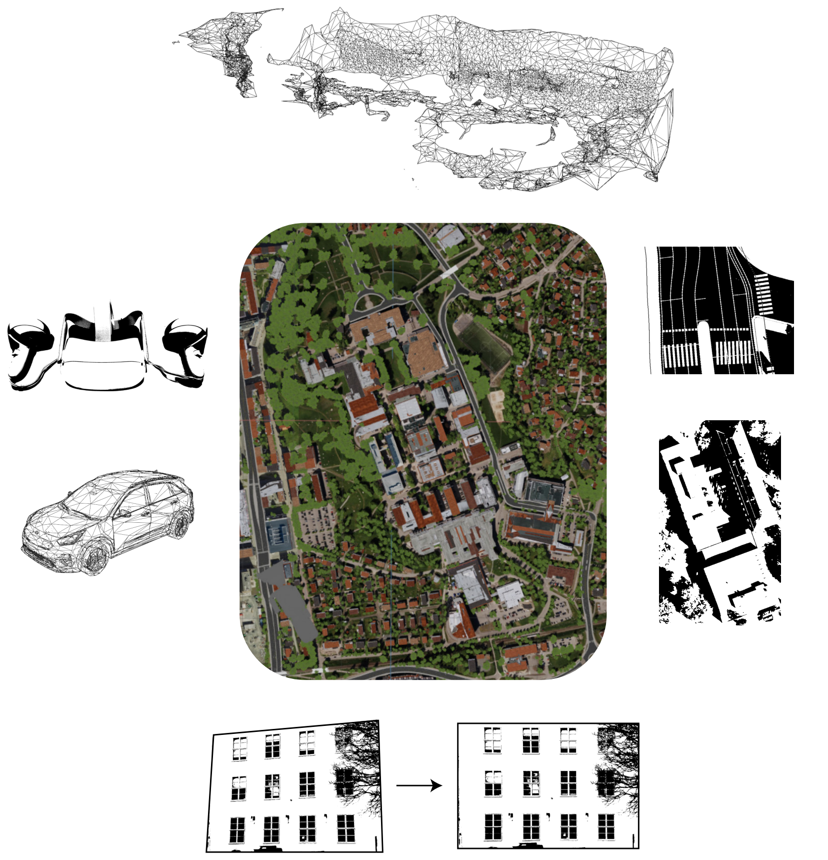
Master's thesis in Computer Science
Supervisor: Gabriel Kiss
Co-supervisor: Frank Lindseth
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Gløshaugen's Digital Twin

Aleksander Knutsen

2022/06/15



‘Quant à Phileas Fogg, il semblait que ce typhon fût partie de son programme.’

‘As for Phileas Fogg, it seemed just as if the typhoon were a part of his programme.’

— Jules Verne
Around the World in Eighty Days

Abstract

Digital twins, i.e. digital, continuously up-to-date digital representations of real-world physical entities, are increasingly used for purposes like simulation, monitoring, prediction, and visualization. This Master's thesis revolves around research and development around digital twins of geographical locations, specifically an area around Gløshaugen in Trondheim, Norway.

Development was done mainly in the NVIDIA Omniverse software platform. Software was created for receiving real-time positional data and update locations of objects in the digital representation. Building geometry and roof textures were fetched and embedded into the digital scene automatically. For wall texturing, a manual method using photography, perspective correction, and image stitching was designed. Also, an automatic alternative for wall texturing was devised using photogrammetry. The scene was also rendered using virtual reality technology.

Research showed that real-world location updates, embedding of building geometry, and roof texturing could be largely automated. The manual method of wall texturing produced good results, and has the potential to run in real-time if automated. The photogrammetric method, although automatic, proved to produce distorted results with only one front-facing on-vehicle camera while driving. It is also slow and thus inappropriate for real-time processing.

Keywords: *Digital twin, computer vision, autonomous vehicles, simulation, photogrammetry, texturing, virtual reality, geography, GNSS, road networks*

Sammendrag

Digitale tvillinger, dvs. digitale, kontinuerlig oppdaterte digitale representasjoner av fysiske enheter fra den virkelige verden, blir i stadig økende grad brukt for formål innenfor simulering, overvåkning, prediksjoner, og visualisering. Denne masteroppgaven omhandler forskning og utvikling rundt digitale tvillinger av geografiske lokasjoner, spesifikt et område rundt Gløshaugen i Trondheim, Norge.

Utvikling ble hovedsakelig utført i NVIDIA Omniverse-programvaren. Kode ble laget for å motta posisjonsdata i sanntid og oppdatere posisjoner til objekter i den digitale representasjonen. Bygningsgeometri og takteksturer ble hentet og integrert inn i den digitale scenen automatisk. For veggteksturering ble det utformet en manuell metode ved bruk av fotografi, perspektivkorreksjon og sammenføyning av bilder. Et automatisk alternativ for veggteksturering ved bruk av fotogrammetri ble også utformet. Scenen ble også gjengitt ved bruk av VR-teknologi.

Undersøkelser viste at posisjonsoppdateringer fra den virkelige verden, integrering av bygningsgeometri, og takteksturering kunne i stor grad automatiseres. Den manuelle metoden å utføre veggteksturering produserte gode resultater, og har potensiale til å kjøre i sanntid ved automatisering. Den fotogrammetriske metoden, skjønt automatisk, viste seg å gi forvrengte resultater ved bruk av kun ett frontvendt kamera festet på et kjøretøy. Den er også treg og derfor upassende for sanntidsprosessering.

Nøkkelord: *Digital tvilling, datasyn, autonome kjøretøy, simulering, fotogrammetri, teksturering, virtuell virkelighet, geografi, GNSS, veinettverk*

Preface

This thesis was written as part of the Master's thesis at the Norwegian University of Science and Technology (NTNU), at the Department of Computer Science (IDI). This report, written for the course 'TDT4900 Computer Science, Master's Thesis', is a culmination of 6 years of higher education within the course programme of Computer Science (MTDT) and beyond.

All development was done with the Windows 10 operating system. This document itself was written in LaTeX, using the Overleaf editor. Inkscape and Adobe Illustrator were used for producing vector graphics illustrations, and occasional bitmap graphics were created with Blender. General image editing was executed with GIMP and Adobe Photoshop.

Acknowledgements

The list of people to extend gratitude towards is long—here comes an attempt at completing it.

First of all, thank you to those that have followed me the most closely throughout this process: My main supervisor Gabriel Kiss, my co-supervisor Frank Lindseth, and PhD candidate Mamoona Birkhez Shami. They have met with me every other week throughout a year, giving invaluable feedback and direction for this project. Their expertise within their fields and their access to resources have greatly helped this thesis reach its final form and become what it is today.

To Rune Strøm Brekke, whose Master's thesis and project files acted as the foundation for this very project. Thank you for the permission to use the files and further develop the project. Getting to know the scene's coordinate system alone proved to be essential for georeferencing!

Geodata has also been very accommodating—their GIS ambassadors were quick to provide licenses to necessary software, and Anne Fiske from Geodata Support was able to give access to their 3D Clip&Ship export service for building models.

A vast amount of gratitude must be directed towards the student organization called Programvareverkstedet (PVV). Its premises have acted as both my main office and my main source of socialization, not only during the writing of this thesis, but also throughout most of my life as a student. Its members have acted as nothing short of a family.

Some specific names from PVV's member base must be mentioned. Daniel Løvbrøtte Olsen, Eirik Wittersø, and Peder Bergebakken Sundt have all acted as rubber ducks, helping me devise methods to solve many a problem and organize many a thought, especially regarding wall texturing. Thank you to Felix Albrigtsen, Amalie Mansåker, and Emma Lu Eikemo, for testing the VR portion of this thesis. Thank you also to Lisa Jonassen, Sindre Østby, Bjørnar Kaarevik, and Anna Bjørnerås Hendseth for lots of encouragement and interest in the progression of the project. Adrian Gunnar Lauterer has contributed in all the aforementioned areas. Jon Martinus Rødtang has in addition performed a general quality and formatting check. Daniel Løvbrøtte Olsen has also contributed with extensive proofreading and suggestions for improvements—truly a thousand thanks!

And to everyone else who deserves a mention: Thank you so much to all!

Contents

Figures

Tables

Algorithms

Code Listings

Acronyms

AI artificial intelligence. 3, 11, 89, 91, 93, 94

API application programming interface. xvi, 30, 45, 107, 108

AR augmented reality. 24, 29, 67, 68

GNSS Global Navigation Satellite System. 6, 16, 30, 36, 39, 41, 43, 61, 79, 80, 89, 91, 92

GPS Global Positioning System. 16, 79

GPU graphics processing unit. 20

HMD head-mounted display. 21, 29, 30, 67

IP Internet Protocol. 68, 108

NAPLab NTNU Autonomous Perception Laboratory. 30, 32, 35, 39, 41, 61, 73, 79, 94

NASA National Aeronautics and Space Administration. 23

NN neural network. 3, 11, 12, 30, 53, 81–83, 89, 91

NTNU Norwegian University of Science and Technology. 30, 73

REST representational state transfer. 30, 45, 107

RGB red, green, blue. 10, 82

ROS Robot Operating System. 94

RTX Ray Tracing Texel eXtreme. 30

TCP Transmission Control Protocol. 68

TIFF Tag Image File Format. 33, 34

UAV Unmanned Aerial Vehicle. 23, 51, 91, 94

URL Uniform Resource Locator. 46

USD Universal Scene Description. 27, 28, 30, 33, 36–38, 41, 46, 49, 67

UTM Universal Transverse Mercator. 5, 13, 15, 37, 38, 41, 46, 49, 51, 52, 62

VR virtual reality. 4–7, 21, 22, 24, 28–30, 67, 78, 92–94, 105

WGS84 World Geodetic System 1984. 13, 14, 16, 37, 41, 60

XR extended reality. 24, 29, 67, 68, 92

Glossary

5G The fifth generation of cellular broadband infrastructure. 23, 29, 79

affine transformation A transformation that preserves distance ratios and collinearity (all points along a line before the transformation are also in a line after the transformation). The set of affine transformations comprise translation, rotation, scaling, shearing, and reflection. 61

altitude In geographical sciences, height above a predetermined limit; often stated in meters above mean sea level. 37, 41, 79, 80, 89, 110

axis-aligned bounding box A bounding box whose edges are parallel to the coordinate axes. 66

bounding box The smallest possible box enclosing all data points of interest. 5, 11, 52, 66

Euler angles Three values used to represent rotation around the X, Y, and Z axes respectively. 20

georeferencing The act of assigning geographical locations to data objects (e.g. images or 3D objects). 6, 24, 37, 46, 51, 61, 62, 73, 89

gimbal lock The loss of one degree of freedom when two rotation axes coincide after a rotation. 20

homogeneous coordinates 3D location coordinates represented by an extra dimension in order to represent translations in transformation matrices. 19

latitude In geographical sciences, the measure of location in the north-south direction. 12, 37, 38, 60

LiDAR A distance measurement tool where laser lights are emitted, bounced off of surfaces, and returned to a receiver. Distances are computed from the elapsed time between emission and reception. 10, 23, 30, 82

locomotion In VR technology, the act of moving within a virtual world by continuous motion. 67

longitude In geographical sciences, the measure of location in the east-west direction. 12, 37, 38, 60

material In 3D computer graphics, a description of a mesh's visual properties like color, reflections, light emission, and so on. 17

mesh A digital three-dimensional object made up of vertices, edges, and faces. xiv, 17, 18, 20, 27, 28, 34, 35, 45–49, 51, 54, 58, 60–64, 66, 71–73, 75, 76, 83, 85–90, 92, 94

orthophoto A top-down photograph of a geographical area, typically taken by a satellite or an airplane. 6, 33–35, 45

photogrammetry The science of reconstructing 3D objects from a set of 2D images. 7, 21, 23, 34, 35, 61–63, 66, 73, 75–78, 89–94

photorealistic rendering The act of producing computer-generated images that are nearly indistinguishable from a real-life photograph. 3, 27

quaternion A way to represent rotations around arbitrary axes without risking encountering gimbal locks. 18, 20, 42

raster image Digital image stored pixel by pixel, as opposed to a vector image which is stored with geometric shapes. 33

raytracing A 3D rendering technique where light rays are simulated to construct photorealistic images. 27

render The act of synthesizing an image from digital 3D objects. 7, 11, 28, 29, 81, 92

saturation In color theory, the degree of chromatic purity, i.e. the difference from the shade of gray having the same brightness. 60, 71, 73

scene In 3D computer graphics, the collection of geometry, lights, and everything else making up a digital 3D world. 4, 27, 28, 30, 32, 33, 36–38, 41, 46, 49, 53, 61, 62, 67, 78, 79, 81, 91–93

scene graph A data structure used to represent logical and spatial relations between objects in a 3D scene. 27

texel The most fundamental unit of a texture (akin to a pixel in a digital two-dimensional image). 18, 35, 81, 82

texture An image mapped to the surface of a digital three-dimensional object. 5–7, 17, 18, 21, 23, 24, 28–30, 33–35, 51–54, 58–61, 71–78, 81–94

Part I

Introduction

‘Sic parvis magna.’

‘Greatness from small beginnings.’

— Sir Francis Drake
SOMETIME IN THE 1500S

Chapter 1

Introduction & Motivation

1.1 Project Motivation

This thesis explains the process of developing a simple digital twin of the Gløshaugen area in Trondheim, Norway. The project will be centered around the traffic aspect of the digital twin. Such a traffic-centered twin has a manifold of practical purposes. It enables real-time monitoring of aspects such as congestion, vehicle count, speed, and much more. Using cameras producing sufficient image quality, aspects such as structural integrity of buildings and roads could be monitored as well. Such a digital twin can also be used for predictive analyses, especially if utilizing artificial intelligence (AI).

A digital twin should, by definition, be updated in real-time by a range of different sensors. On a large enough scale, with enough entities carrying appropriate sensors, this can be done completely automatically. This has the potential for reducing large amounts of expenses in the long run that would otherwise be used for salaries, specialized equipment, and other resources.

A digital road model can contain metadata related to vehicle behaviour, especially in terms of restricting their movements regarding lanes, traffic signals, traffic rules, etc. Using such metadata provides a foundation for vehicle simulation. Simulation capabilities provide an environment in which AI agents can be trained and algorithms can be verified. Combine this with photorealistic rendering, and one can generate realistic images from the perspective of a human driver. Such renderings can in turn be used for training and validating neural networks (NNs) for use with autonomous vehicles. Simulation and photorealistic rendering combined also provides an arena to generate weather effects and mimic lighting conditions at different times of day. By training neural networks on a variety of different scenarios, their accuracy can in turn increase.

The ability to fabricate training and validation data in this way has a number of benefits. For training neural networks, a large amount of images is required. Collecting these images usually requires real-life driving, with camera-equipped vehicles. This can be cost-demanding in terms of equipment, fuel and time consumption, human drivers, and any risks involved with driving in traffic.

Once an autonomous vehicle has been trained with the appropriate data, it can be put into practical use in real-world traffic. While driving, a camera records real-time video of its surroundings, which the vehicle uses to make predictions based on what it perceives. For such predictions to be as good as possible, a number of criteria ideally need to be fulfilled. The road should be free of any coverage like snow or water puddles. The car should have a clear line of sight in front of it, free from weather phenomena like heavy downfall or mist. The sun should also not be angled so as to severely overexpose images taken from the car's camera(s). However, all of these conditions are relatively commonplace, especially in Norwegian climate. This means that a conventional autonomous car could potentially perform sub-par compared to locations with other, more forgiving climates. However, with an accurate digital twin having proper metadata about road lanes, driving directions and the like, an autonomous vehicle could use the twin along with its own positional data to align itself in the proper lanes, even in the cases of low visibility caused by, for example, mist or heavy downfall.

Although a digital twin is meant to reflect the real world as closely as possible, its usage can be extended beyond this strict linkage. For example, by changing the road network topology, one can simulate how traffic flow will be affected were these changes to be realized. New buildings could be added along with metadata like expected visitor times, durations, and amounts, and could potentially uncover otherwise unforeseen weaknesses in future traffic infrastructure like bottlenecks or dangerous intersections.

Such an environment can also benefit from being rendered in virtual reality (VR). VR visualization can enhance the act of monitoring conditions within the digital twin. For example, if a structural weakness has been detected in a building or road, one can inspect the area in virtual reality, possibly without approaching the real-life area. Preventative measures can then be taken before any accidents occur. VR can also be used in this context for city planning, for example for visualizing future building projects or road network changes. Multiple alternatives can be visualized and compared, without running the risk of spending large amounts on preliminary construction. VR can also be used for overlaying a variety of data to the user, like temperature, weather forecasts, or air pollution.

1.2 Research Questions

To aid in the development and research of this thesis, a number of research questions were formulated. The research questions, as shown below, represent the main questions that this thesis attempts to answer:

RQ1: Can dynamic location data of a physical entity be transferred to a digital 3D scene in order to update it in real-time?

RQ2: Can a three-dimensional model of a geographical area be acquired or created, especially with terrain, buildings, and roads; and to what degree can this be automated?

RQ3: Can any photo of a building be applied as a texture to a corresponding 3D object so that the building looks realistic enough to be near indistinguishable from reality?

RQ4: Can we use photos taken from a driving car to update building textures automatically in real-time?

RQ5: Can a digital twin be visualized in VR alongside real-time scene updates?

1.3 Foundation & Contributions

This thesis will partly consist of practical development regarding a geographical digital twin. The area in focus is Gløshaugen, Trondheim and its surroundings. The area contains the largest campus of NTNU, which is advantageous considering the close proximity to the university's resources. Within the area is also Elgeseter gate, which is part of the main vehicular artery to the city of Trondheim. This road often sees large volumes of traffic, which is of great benefit when analyzing and simulating traffic. The exact area considered in this thesis is shown in Figure 1.1, and is defined by the following bounding box given in UTM33N coordinates (see Section 2.3.1 for more details):

$$\begin{aligned}\text{Northwest corner: N 7040993, E 270211}\\ \text{Southeast corner: N 7039779, E 271077}\end{aligned} \tag{1.1}$$

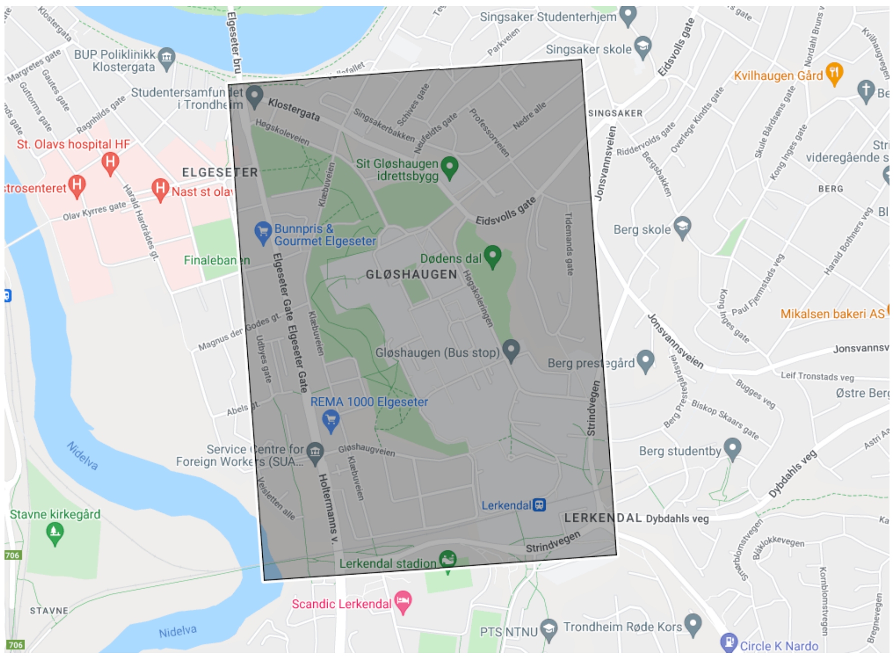


Figure 1.1: Map of the area covered by this thesis' digital twin development.

The practical work of this thesis is based on the work performed by Rune Strøm Brekke from their Master's thesis from 2021 [1]. Brekke created a three-dimensional digital model of the Gløshaugen area, with terrain, buildings, and vegetation. Buildings contained procedural textures, and the terrain was draped with an orthophoto. This thesis is also based on my project work from the course 'TDT4501 Computer Science, Specialization Project' attended in 2021 [2]. Here, a road model was created, complete with textures as well as behavior rules for simulated vehicles.

As part of the development in this thesis, the road network was properly georeferenced. In order to improve the accuracy of the building textures, methods for roof and wall texturing for buildings were explored. One method uses manual plane-to-plane homography, and another uses automatic photogrammetry and georeferencing. Additionally, code was created to update the locations of 3D digital car models. The scene was also set up to be visualized in VR.

1.4 Thesis Structure

The thesis is structured as follows:

PART I is the current part, and contains introductory material.

- **Chapter 1** contains an introduction to the main themes of the thesis. It also presents research questions, the foundation of which this thesis is written, and finally this very structure overview.
- **Chapter 2** introduces theoretical background knowledge useful for understanding the rest of the thesis.
- **Chapter 3** outlines previous work in the fields related to the main themes of this thesis.

PART II is the main part of the thesis. It outlines the necessary prerequisites and explains the development process conducted as part of this thesis.

- **Chapter 4** outlines the software, hardware, services, and assets used in the development process.
- **Chapter 5** introduces the most common file formats, data types, and data sources involved with the project.
- **Chapter 6** features the steps performed on the starting assets in order to prepare them for the rest of the development process.
- **Chapter 7** shows the steps performed in developing software for updating the position of a three-dimensional object from GNSS coordinates.
- **Chapter 8** contains steps for downloading and georeferencing building objects from an online service.
- **Chapter 9** explains how to apply high resolution roof textures to buildings automatically.
- **Chapter 10** features experiments performed in order to assess the viability of using photography to update textures of 3D building objects.

- **Chapter 11** introduces an unsupervised method for updating building textures using photogrammetry.
- **Chapter 12** explains the steps taken to render the scene in virtual reality.

PART III contains results and discussions concerning the development process, as well as a conclusion and an outline for future work.

- **Chapter 13** shows results from the methods and experiments performed in Part II, along with videos showing results from all major aspects of development combined.
- **Chapter 14** discusses findings from the results, as well as problems and shortcomings with the proposed methods. It also discusses whether and to what degree the research questions were answered.
- **Chapter 15** presents a summary of the thesis at large, and presents suggestions for further development and research.

PART IV consists of the thesis' appendices.

Chapter 2

Background Theory

2.1 Digital Twins

At the heart and core of this thesis is the concept of a digital twin. Its definition has varied across time, applications, and research fields. The concept consists of two words, which are defined by Merriam-Webster as such:

- **DIGITAL** (*adjective*) — *'characterized by electronic and especially computerized technology'*; *'composed of data in the form of especially binary digits'* [3]
- **TWIN** (*noun*) — *'one of two persons or things closely related to or resembling each other'* [4]

A digital twin can in its most basic terms be defined as being an accurate digital counterpart of a physical entity. M. Grieves of Florida Institute of Technology proposes that any digital twin is composed of three primary constituents, illustrated in Figure 2.1: [5]

- A **physical space**, where a real physical entity or asset exists;
- A **virtual space**, where a digital representation of that physical entity or asset resides;
- A **link** between between the physical and the virtual space, where data and information can be bidirectionally exchanged.

There are in principal no limitations as to what can be a physical entity. Constructions like buildings or vehicles, to geographical areas like a city center or a farm field, even living entities like animals or humans can be such an entity having a virtual counterpart.

The bidirectionality aspect of the link between the two spaces is rephrased by M. Singh et al., who proposes as part of the definition of a digital twin that:

> *'...It is not just the Digital Twin which mimics its physical twin but any changes in the Digital Twin are mimicked by the physical twin too.'* [6]

W. Kritzinger et al. confirms this in their proposal that a digital model be classified into three subclasses based on its level of integration with the physical world (see Figure 2.2): [7]

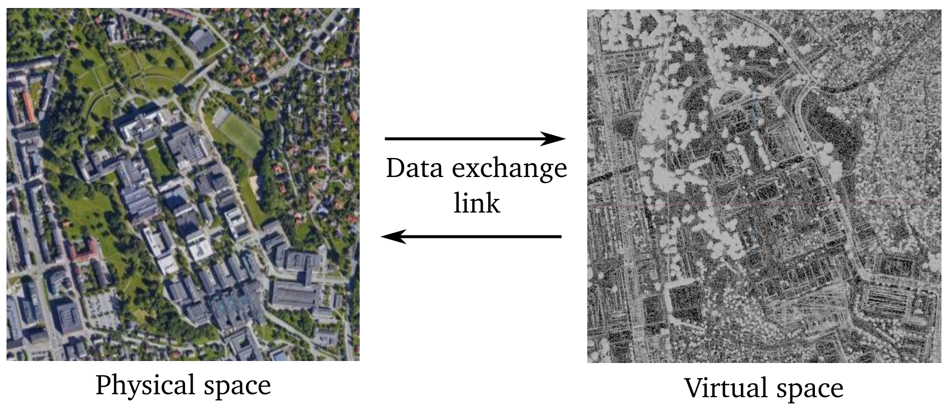


Figure 2.1: The three basic components of a digital twin, according to M. Grieves.

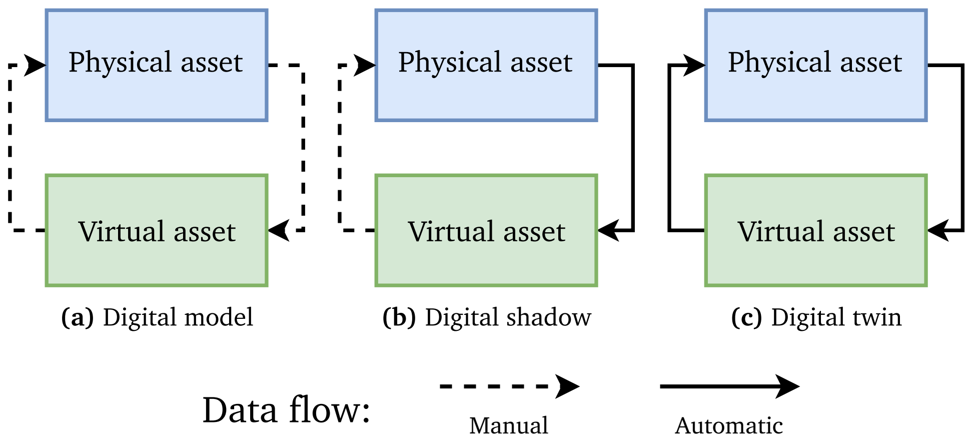


Figure 2.2: Illustration showing the different integration levels of a digital twin, as proposed by Kritzinger et al.

- ***Digital Model*** — A digital model represents a physical entity, but has no automatic transfer of data in either direction.
- ***Digital Shadow*** — A digital shadow builds on the definition of a digital model, in that in a digital shadow there is a one-way automatic flow of data from the physical entity to the virtual entity. The reverse direction of data flow is still manual.
- ***Digital Twin*** — A digital twin is characterized by the automatic flow of data being bidirectional between the physical and the virtual asset.

One common way that data can be sent from the physical to the virtual world automatically is by using sensors. Radars and LiDARs, RGB cameras, heart rate monitors, wind and precipitation sensors, and thermometers are all examples of such sensors. It is critical that the transfer of data in any direction should happen in real-time [5][8]. The fulfillment of this requirement facilitates the possibilities of using a digital twin for monitoring, simulation, and prediction [9][10].

Digital twins have a wide range of applications, ranging from industrial engineering and manufacturing to healthcare and aviation [5][7][11][9]. Some conceptual examples are:

- An industrial mechanical arm can send data from its sensors and motors to a digital twin, which reflects the arm's physical pose at any moment in time. The pose should also be changeable inside the digital twin, and its physical state should instantaneously reflect the changes performed in the digital twin.
- A biological digital twin can monitor a person's vital signs like heart rate or the presence of allergens in the body. The digital twin can react by activating an automatic pacemaker in case of a heart failure, or injecting epinephrine in case of a severe allergic reaction.
- An aerial drone can photograph farmlands and update a twin with data like irrigation levels or damaged crops. These data can be used for automatic decision-making, for example by watering dry areas or planting new seeds.

2.2 Autonomous Vehicles & Computer Vision

Autonomous vehicles are vehicles characterized by their ability to drive without human input. They are able to perform independent decision-making in all aspects of driving, be it keeping distance to other vehicles, bikers, and pedestrians; stopping in front of light signals; and predicting and avoiding accidents.

In order for autonomous vehicles to perform their tasks, they are typically trained by an artificial neural network (NN). The branch of artificial intelligence (AI) called computer vision is of particular importance. In computer vision, neural networks are used for interpreting and understanding visual aspects of the real world. The main data source used for these networks are images and videos recorded by a front-facing camera equipped to the car.

Computer vision NNs require annotated images for it to be trained. The process of annotating an image usually involves placing bounding boxes around objects in images, and labeling these boxes with whatever is enclosed by the bounding box. The labels are typically taken from a pool of predefined categories. In the context of traffic, common labels can be 'vehicle', 'pedestrian', and 'cyclist'. A neural network is then trained on these labels and bounding boxes, allowing them to later predict new bounding boxes from the same category pool on unseen images. Figure 2.3 shows an example of predictions performed by a neural network on an unseen image.

It is of utmost importance that labels are correctly identified in real-time while driving. For example, if the AI fails to identify a pedestrian walking across the street on a red pedestrian light, the car might cause an accident by colliding with the person. This implies that the network must be trained on images from a variety of situations. If using renders from a digital twin as training data, this also means that these renders should be as photorealistic as possible.

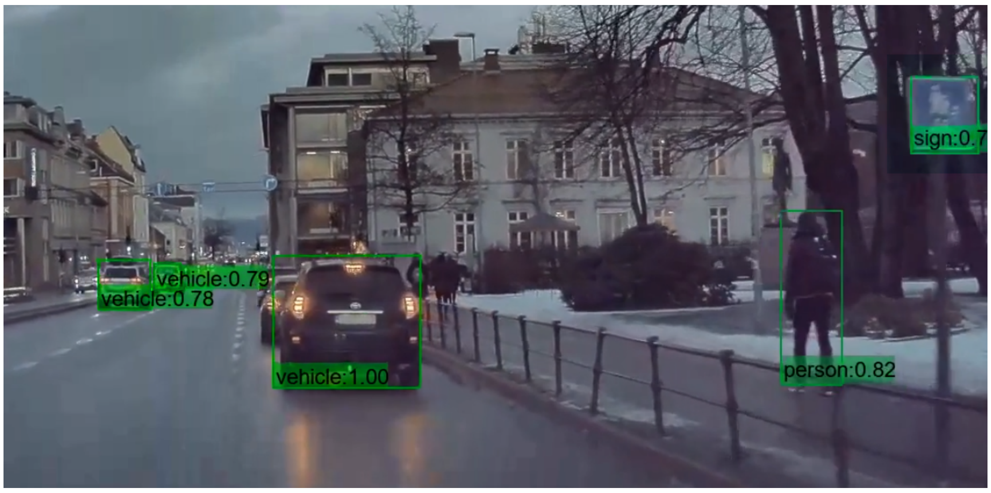


Figure 2.3: Screenshot from a video taken from a car driving along Elgeseter gate in Trondheim, Norway. The image has been tagged by predictions from an artificial NN trained to recognize vehicles, persons, signs, and bicycles. The figure is part of the author's results from the final project in the course TDT4265 Computer Vision and Deep Learning, Spring 2020. The image has been edited to have blurred vehicle registration plates for privacy reasons.

Autonomous vehicles can also benefit from so-called high-definition maps, which are accurate digital descriptions of a road network. These maps are often accurate to the centimeter, and can contain information like lane positions, traffic signs, and road shape. A high-definition map can act as a guide or a ground truth for the vehicle, meaning it can perform well even in low-visibility situations and environments.

2.3 Geodesy

2.3.1 Spatial Reference Systems

Considering that this thesis revolves around a digital twin of a particular geographical area, it is imperative to have a way to refer to geographical locations. Locations on the surface of Earth are usually defined by two values: **Longitude**, which denotes the position along the east-west axis; and **latitude**, which denotes the position along the north-south axis.

If the planet had been a perfect sphere, all locations on the surface would have a unique mathematical spherical coordinate representation. However, the Earth is an irregularly shaped ellipsoid, caused by mountains, ocean trenches, and centrifugal forces from the Earth's rotation around its own axis [12]. Therefore, there exist multiple coordinate systems, i.e. multiple ways to use longitude and latitude to refer to geographical locations.

One of the most common coordinate systems, called World Geodetic System 1984 (WGS84), uses longitudes and latitudes measured in degrees, for referring to a location on a sphere or ellipsoid. It defines the Earth's geographic north pole to be 90° north, its south pole to be 90° south, and the latitude in the middle of these (i.e. the equator) to be at 0°. The prime meridian (i.e. the longitude defined as having 0°) runs through Greenwich, England, perpendicular to the equator. The longitudes range from 180° west to 180° east. WGS84 coordinates can optionally be referenced with degrees, minutes, and seconds, as an alternative to decimal degrees. In this case, one degree is divided into 60 arcminutes (often called simply 'minutes', denoted by a single apostrophe), and each arcminute is divided into 60 arcseconds (often called simply 'seconds', denoted by a double apostrophe).

Figure 2.4 shows an example of how the WGS84 coordinate system works. The coordinates shown on the figure, denoted by the place where the two red circles intersect, are defined by the coordinates

$$\text{N } 34°58'1.9'', \text{ E } 135°47'7.84'' \tag{2.1}$$

where N denotes degrees, minutes, and seconds north from the equator, and E denotes degrees, minutes, and seconds east from the zero meridian. u in the figure illustrates the longitudinal rotation at the center of the Earth, and v represents the latitudinal rotation. The black circles represent the prime meridian and the equator respectively.

Another commonly used coordinate system is called Universal Transverse Mercator (UTM). The central concept around the workings of UTM is dividing the Earth into 60 different zones, each spanning 84° north to 80° south and 6° in longitude. Coordinates are defined in meters north and east. Each zone's center meridian is defined to be 500 000 meters east. On the northern hemisphere, the maximum north value is set to be 9 300 000 meters north from the equator, and on the southern hemisphere, equator is set to have a north value of 10 000 000 meters north. These definitions imply that the system can be used without the concern of negative coordinate values. To eliminate ambiguities regarding whether given coordinates are given on the northern or southern hemisphere, zones are often split along the equator, where the northern part is denoted with an N suffix, and the southern part with an S suffix.

Figure 2.5 shows an example of a location on Earth using the UTM31N coordinate system. Figure 2.5a shows the area covered by UTM31N. The example location is defined by the coordinates

$$\text{N } 4827064, \text{ E } 378369 \tag{2.2}$$

meaning that the location lies 4 827 064 m north of the equator, and 121 631 m west of the center meridian.[1] These numbers are illustrated in Figure 2.5b.

[1] $500\,000\,\text{m} - 378\,369\,\text{m} = 121\,631\,\text{m}$

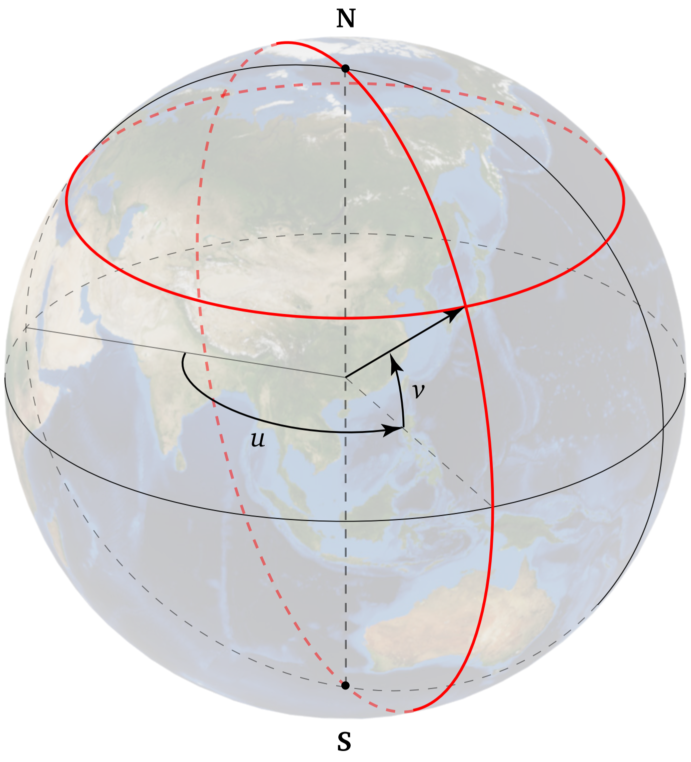
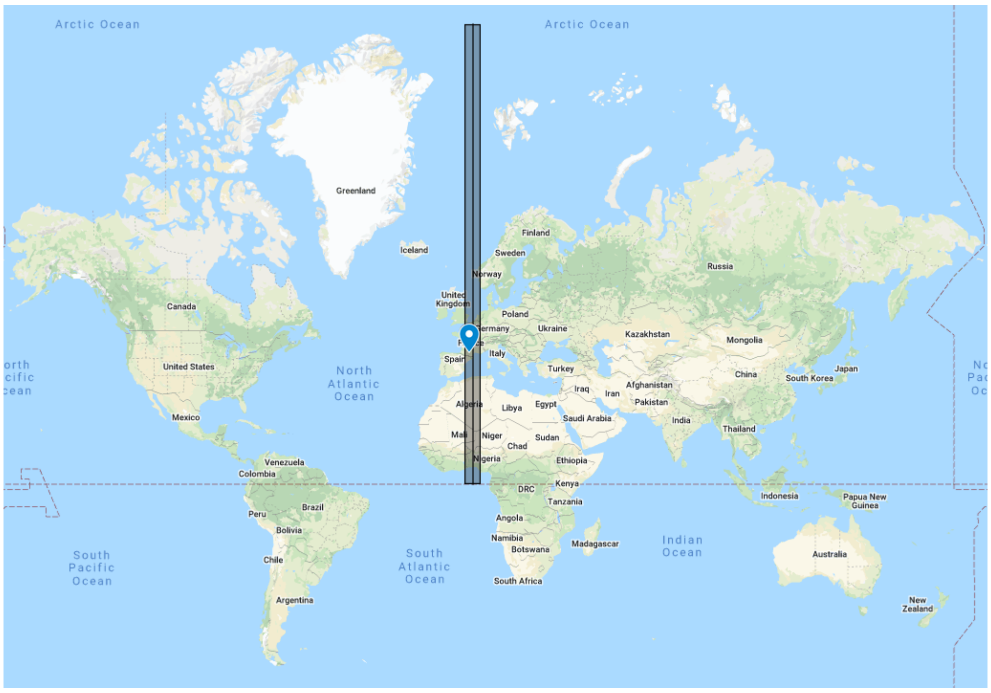
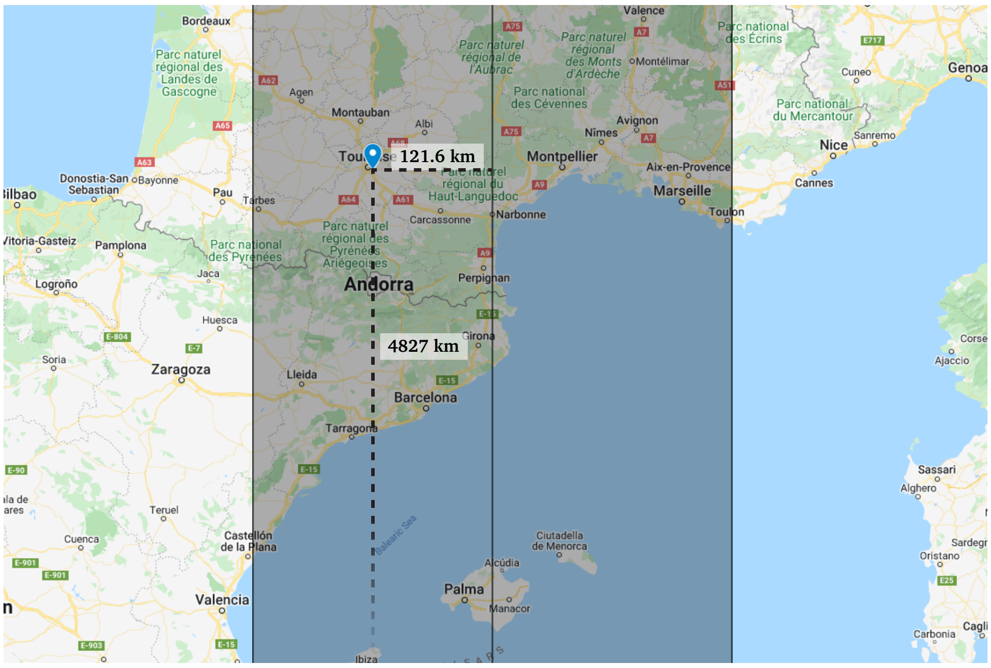


Figure 2.4: Example of a location on the surface of Earth, represented with spherical coordinates like in the WGS84 coordinates system.



(a) The full area defined by UTM31N, along with the chosen location.



(b) The chosen location as defined by the distance from equator and from UTM31N's center meridian.

Figure 2.5: Illustration of how a location is defined in the UTM coordinate system.

2.3.2 GNSS

A Global Navigation Satellite System (GNSS) is a system for automatically determining the coordinates of geographical locations on Earth. Traditionally, the most widespread GNSS has been the Global Positioning System (GPS). It is developed and owned by the government of the United States, originally for military use. GPS uses the WGS84 coordinate system [13]. Other well-known systems are Galileo, developed by the European Union; and GLONASS, developed by Russia [14] [15].

A GNSS typically comprises three segments: [16]

1. **Space segment**, consisting in turn of a number of artificial satellites in Earth orbit.
2. **Control segment**, consisting of ground stations used to ensure that the satellite signals are accurate.
3. **User segment**, consisting of electronic devices utilizing the satellites to determine their own location and time.

The satellites each require highly stable atomic clocks for accurate timekeeping. Each satellite also emits a signal propagating with a known speed. The time of signal emission from the atomic clock is part of this information. When these signals reach the Earth, they can be picked up by GNSS receivers. The receiver can calculate the difference between the current time and the timestamp present in the GNSS signal in order to determine the propagation time of said signal. Using four or more satellites, the receiver is able to compute its latitude, longitude, altitude, and time [17].

In isolation, a GNSS can have an accuracy of a few tens of centimeters to several meters [18] [19]. However, using 5G network signals and/or CPOS technology,[2] the accuracy can be increased to less than a centimeter [20] [21]. Corrections from 5G networks is based on relative positions compared to other devices connected to the network. Conversely, CPOS utilizes a 'virtual base station', which in reality is a web server sending correction data to the GNSS receiver.

2.4 3D Graphics Concepts

This thesis will not act as a course, an introduction, or an encyclopedia for three-dimensional graphics. However, because three-dimensional data is central to this thesis, a short summary of the most important features will be presented here. This will provide a common foundation when referring to important aspect of 3D graphics.

[2]Note that CPOS technology only covers mainland Norway.

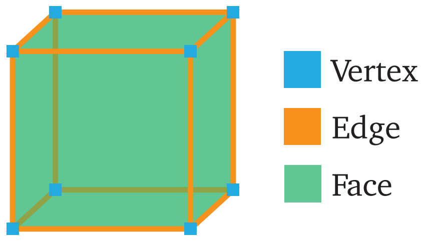


Figure 2.6: Illustration showing the most basic geometric components of a mesh.

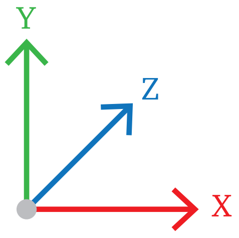


Figure 2.7: Figure showing a widget commonly used in 3D graphics to denote coordinate axes and their positive direction.

2.4.1 Geometry

An object in 3D graphics is commonly called a **mesh**. A mesh is usually built up with three different components, as shown in Figure 2.6:

- **Vertex** (plural **vertices**) — A vertex, in and of itself, is a 0-dimensional point in space.
- **Edge** — An edge is a 1-dimensional edge in space. An edge only exists between two vertices.
- **Face** — A face makes up a closed set of three or more edges. The surface of a mesh consists of a collection of such faces.

All these components exist in three-dimensional space. In order to refer to the location of these components, as well as the mesh itself, it is common to use a coordinate system. Convention dictates that the basis vectors for this coordinate system are called **X**, **Y**, and **Z** respectively (see Figure 2.7).[3]

There are also some common attributes that can be derived from these components. Each face has a **normal vector**, or simply a **normal**, indicating which direction it is facing. Each vertex can have its own **vertex normal**, calculated from the normals of adjacent faces.

2.4.2 Surface Appearance

As for the surface appearances, meshes often have **materials** applied to them. A material is a description of how the mesh reacts to light, i.e. a description of its visual appearance. This encompasses the mesh's surface color, reflective and refractive properties, self-emission, 'shininess', and so on.

Sometimes, one desires to drape a two-dimensional image around a mesh to give the illusion of having a more complex surface than what is represented by geometry alone. Such draped images are often called **textures**. Textures are made

[3]Note that it is *not* standardized which axis points in which direction. The most common discrepancy is which axis to point upwards. Some application use the Y axis as the up axis, while others use the Z axis.

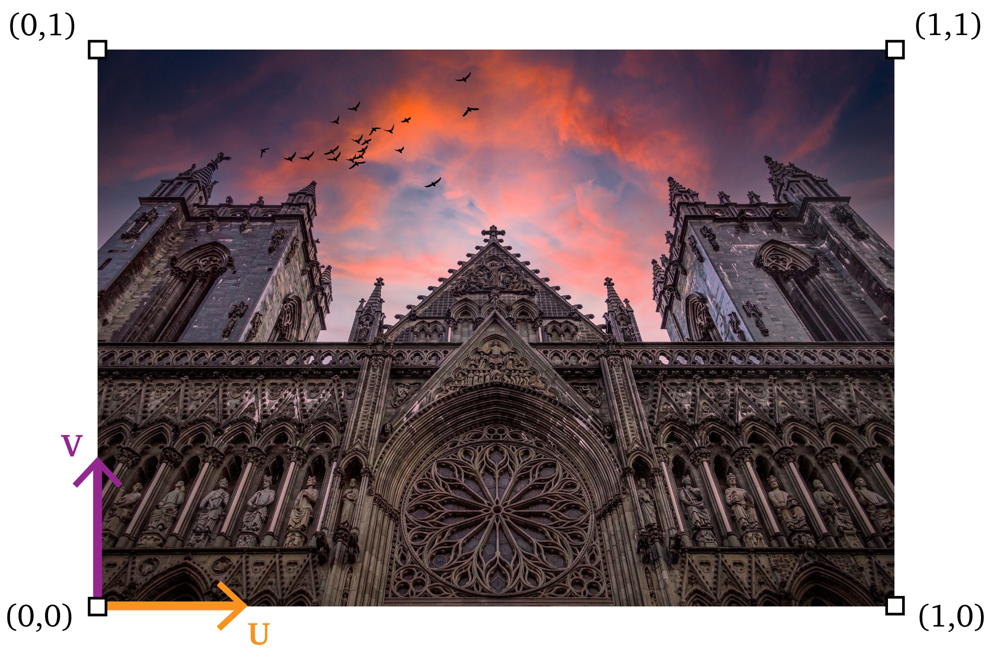


Figure 2.8: Example of an image with the corners marked with their corresponding UV coordinates. A UV coordinate widget is shown at the image's origin. The image is taken by the author themselves and used with permission.

up of fundamental units called **texels**, which are analogous to pixels being the fundamental unit of general digital images.

For representing which parts of an image should be applied to which parts of the three-dimensional mesh, one uses what is commonly called **texture coordinates** or **UV coordinates**.[4] UV coordinates are given in the range of 0 to 1 relative to the image used as a texture. For example, if the bottom left corner has UV coordinates $(0,0)$, then the opposite (top right) corner has coordinates $(1,1)$ (see Figure 2.8). Each vertex in the part of the mesh to be textured is assigned a (u,v) tuple. All texels positioned inside the bounds of the vertices' corresponding face are sampled and applied to that face. An illustration of this is shown in Figure 2.9.

2.4.3 Transformations

3D transformations are of paramount importance within the realm of computer graphics. They are used for any translation (movement), rotation, and scaling of object within a scene, as well as for giving the illusion of perspective. Such transformations are usually represented by matrices, and a transformation can be performed on a mesh by multiplying the transformation matrix with a matrix consisting of the vertex coordinates of all vertices in that mesh.

[4]We use U and V for coordinate names in order to distinguish them from the X, Y, and Z axes used for representing 3D space, and W often used for quaternion rotation.

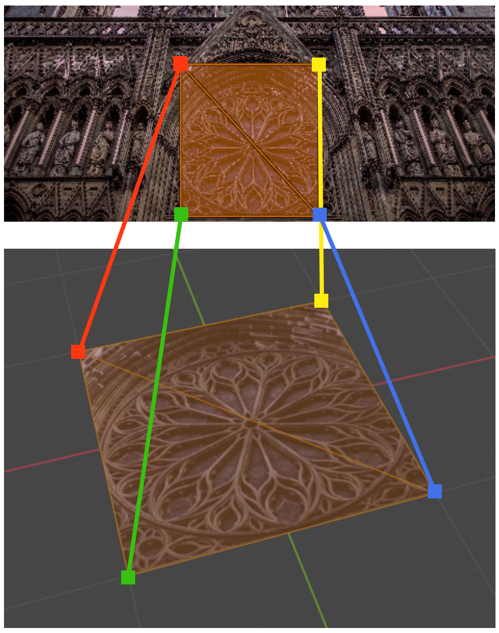


Figure 2.9: Figure showing how UV coordinates are applied to mesh vertices. Squares having the same color represent corresponding vertex-UV pairs.

In three-dimensional computer graphics, a transformation is usually represented as a 4×4 matrix, rather than a 3×3 matrix as intuition would suggest. This enables representing translations via matrix multiplications. Consider Equation 2.3. Here, $\mathbf{M}$ is a 3×3 matrix representing a general three-dimensional transformation; p is the coordinates of a vertex to be transformed; $\vec{t}$ is a translation vector; and p' is the new vertex coordinates after performing the transformation. Next, consider Equation 2.4. Here, $\mathbf{T}$ is instead used as the transformation matrix. $\mathbf{T}$ is a 4×4 matrix with the translation vector embedded in its fourth column. p_h is the coordinates of the vertex, with an added dimension with a value of 1. p'_h is the result of the matrix multiplication, again with a extra dimension. We see that the first three rows of p'_h is equal to p'. After performing the transformation in Equation 2.4, we can in practice ignore the fourth row of p'_h when applying the result to the vertices. Coordinates with such an additional dimension are called **homogeneous coordinates**.

$$
\begin{aligned}
p' = \mathbf{M}p + \vec{t} &= \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \\ c_{3,1} & c_{3,2} & c_{3,3} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \\
&= \begin{bmatrix} c_{1,1}x + c_{1,2}y + c_{1,3}z + t_x \\ c_{2,1}x + c_{2,2}y + c_{2,3}z + t_y \\ c_{3,1}x + c_{3,2}y + c_{3,3}z + t_z \end{bmatrix}
\end{aligned}
\tag{2.3}
$$

$$\begin{aligned} p'_h = \mathbf{T}p_h &= \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} & t_x \\ c_{2,1} & c_{2,2} & c_{2,3} & t_y \\ c_{3,1} & c_{3,2} & c_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} c_{1,1}x + c_{1,2}y + c_{1,3}z + t_x \\ c_{2,1}x + c_{2,2}y + c_{2,3}z + t_y \\ c_{3,1}x + c_{3,2}y + c_{3,3}z + t_z \\ 1 \end{bmatrix} \end{aligned} \tag{2.4}$$

Representing transformations with only matrix multiplications is beneficial from a performance perspective. GPUs are often specialized on performing few types of operations of large amounts of data. Having to deal with only matrix multiplication, instead of both multiplications and additions, means that the GPU can be more specialized in multiplications. An entire transformation can also be represented by one single matrix, instead of one for translation and one for all other transformations. This makes transformations considerably faster to compute.

When performing transformations, it is important to consider the space in which the transformation takes place. When focusing on a mesh in isolation, coordinates are given in **object space**. Coordinates in object space are typically given relative to a specified object center. When assembling multiple object together in a scene, coordinates must be transformed to **world space**. Here, all coordinates are usually given relative to a defined world center. In **camera space**, coordinates are transformed to be relative to a camera object, where entities visible to the camera are usually defined in the range of -1 to 1. Finally, in **viewport space**, coordinates are transformed to pixel coordinates.

2.4.4 Quaternions

Quaternions are a mathematical construct often used within computer graphics to represent 3D rotation. Quaternions can be represented by a scalar part s and a vector part $\overrightarrow{v}$:

$$q = (s, \overrightarrow{v}) \tag{2.5}$$

A rotation by an angle θ around an arbitrary axis $\overrightarrow{n}$ can then be expressed as

$$q_r = \left(\cos\frac{\theta}{2}, \overrightarrow{n}\sin\frac{\theta}{2}\right) \tag{2.6}$$

The main advantage of using quaternions over Euler angles is the absence of gimbal lock. Gimbal lock occurs when two of the rotational axes of an object aligns, removing one degree of freedom. Quaternions solve such ambiguities by adding another dimension to the definition of the rotation.

2.5 Photogrammetry

Photogrammetry is the science of using a collection of two-dimensional images to reconstruct a three-dimensional scene. It works by gathering visible features from images, and matching these between images. By knowing the properties of the camera lens and observing how features differ in location between images, one can compute a reconstruction of the scene's three-dimensional geometry. By sampling pixels from the original image, one can also apply textures to the resulting geometry.

2.6 VR

Virtual reality (VR) is a technological concept that aims to 'embed' a user into a virtual world, where the world can be shown and interacted with via kinetic input. Modern, consumer level virtual reality suites usually consists of a head-mounted display (HMD)—often called a 'headset'—and two handheld controllers, one for each hand. The HMD contains two lenses—one for each eye—and translates and rotates a virtual camera corresponding to how the user moves their head while wearing the HMD. The headset is meant to be worn on the user's head, in front of the eyes, so that the user sees what is being rendered to the lenses. Moving the hand-held controllers often enables moving objects in the virtual world correspondingly. Said objects are often meant to represent human hands, or an extension thereof (e.g. tools). Many controllers also have buttons and analog sticks for sending additional commands to the virtual world. Figure 2.10 shows an example of the most common hardware components of a VR suite.

A number of early VR kits require the use of base stations, which are sensors used for tracking the positions of the headset and handheld controllers. In addition, early VR kits typically require the use of an external computer, herein called a host computer, in order to run VR software. Many newer headsets support stationless tracking via cameras embedded in the HMD. They also often have sufficient computing power to run independently, although the only natively supported software are usually games bought in an online store.

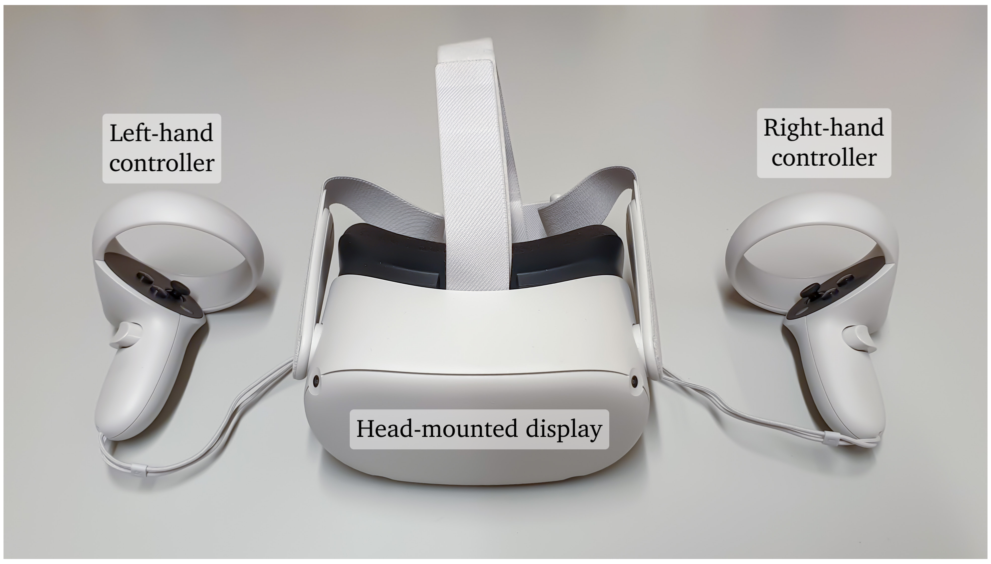


Figure 2.10: Illustration of the most important hardware components of a VR suite. The suite shown is called Meta Quest 2.

Chapter 3

Related Work

The earliest form of digital twins have existed since 1970, when the National Aeronautics and Space Administration (NASA) used a digital twin of the Apollo 13 spacecraft to guide its astronauts to safety after a critical accident on board [5]. The concept entered the field of product lifecycle management in the early 2000s under names like 'Mirrored Spaces Models' [22]. The first known use of the name 'digital twin' appears in a 2010 publication by NASA presenting a draft of a roadmap for its own technological evolution [6][23].

An example of geographical digital twin technology in use today comes from Institute for Automotive Engineering at RWTH Aachen University in Germany. Per 2021, they use LiDAR and 5G technology combined with digital twins in the development of infrastructure to integrate autonomous vehicles [24].

One important aspect of this thesis' digital twin is updated, realistic, and high resolution building textures. Some previous work has been made in building texturing using either a video stream or an unordered sets of images, and pose estimation [25]. However, this assumes that camera position and orientation can be accurately retrieved, which is not always the case. Other methods mainly use street-level photography with automatic mosaicking, but without any automatic application to 3D objects [26]. This thesis will explore some alternatives, both manual and automatic, for application of wall textures to 3D objects without the need for pose estimation or camera calibration.

Photogrammetry is a technique being increasingly utilized in the creation and update of digital twins. Unmanned Aerial Vehicles (UAVs) have been used for generating digital twins of man-made constructions such as bridges, and for subsequent quality assessments and monitoring of said constructions [27]. Progress has also been made for using photogrammetry and digital twins for underwater purposes, taking into consideration refraction phenomena between different physical media as well as high-pressure underwater environments [28]. However, little work has been made using photogrammetry at street level, especially using camera-equipped vehicles, to generate textured three-dimensional models of buildings. This thesis will explore the viability of generating textured buildings using photogrammetry with street-level imagery taken while driving.

Building reconstruction in and of itself is also relevant for geographical digital twins. Automatic building reconstruction is possible using elevation data, satellite images, and/or aerial images [29] [30]. For this thesis, pre-reconstructed buildings are retrieved from an online service only.

Extended reality (XR) has also seen increasing development in the context of digital twins. Industrial applications are becoming increasingly prevalent, incorporating both augmented reality (AR) and virtual reality (VR) technology [31] [32]. Other use cases, especially within VR, are remote surgery [33], traffic simulation and accident prevention [34], and more general situations and processes that require human input [35]. VR has even been used in combination with digital twins in contexts like street network modelling, movement patterns, wind flow simulation, and even qualitative perception data, in a case study involving the small German town of Herrenberg [36] [37]. This thesis will likewise revolve around a geographical VR-enabled digital twin, with an emphasis on vehicle positions, building georeferencing, and texturing.

Part II

Materials & Methods

'...if you are a student you should always get a good night's sleep unless you have come to the good part of your book, and then you should stay up all night and let your schoolwork fall by the wayside, a phrase which means "flunk".'

— Lemony Snicket
'THE AUSTERE ACADEMY'
A SERIES OF UNFORTUNATE EVENTS

Chapter 4

Tools & Assets

4.1 Software

4.1.1 Universal Scene Description

Universal Scene Description (USD)[1] is a framework developed by Pixar meant to act as a standard file format for creating and storing 3D data. It combines different 3D content types, like meshes, animation, particles, and physics simulations into one standardized format. Put in another way, USD exists to alleviate the need to compose 3D scenes containing many different content types, and make those scenes interchangeable between different 3D software and platforms. It also contains a highly optimized scene graph, enabling users to collaborate on one and the same USD scene in real-time.

4.1.2 Omniverse

At the core of the project is a software platform by NVIDIA named Omniverse.[2] It is a real-time development platform designed for content creation, simulation, photorealistic rendering, and collaboration across different 3D creation software. The USD file format is used extensively throughout the platform. For this thesis, three key sub-units of the platform were used:

- **Nucleus** is the core of the collaborative aspect of Omniverse. It is based on a publisher/subscriber model, where Omniverse clients can publish changes to the Nucleus that subscriber clients can acquire. This transfer takes place in real-time, and can be bi-directional, meaning that a client can be both a subscriber and a publisher simultaneously. Nucleus contains a file system structure containing all assets to keep synchronized.
- **RTX Renderer** enables the creation of photorealistic renders using real-time raytracing.

[1]`https://graphics.pixar.com/usd/release/index.html`
[2]`https://developer.nvidia.com/nvidia-omniverse-platform`

- **Connect** is a series of extensions and plugins that enables communication between 3rd party applications via Nucleus. Communication typically takes place in the form of updating the contents of a 3D scene. Consequently, Nucleus enables such communication to happen in real-time.

Omniverse also features a collection of applications for different purposes. The following were used in this project:

- **Cache** was used to improve loading times and reduce bandwidth between clients and servers.
- **Create** was the main piece of software used to view, edit, and render the project scene.
- **Drive** was used in order to access the Nucleus server files as if they were mounted on a Microsoft Windows drive.
- **XR** was used to render the USD scene in VR.

4.1.3 RoadRunner

MathWorks RoadRunner[3] is a software used for modelling road networks, either from scratch or using satellite imagery of Earth's surface. Roads can be edited with simulation metadata, like speed limits, driving directions, and traffic light behavior. Although no road network was modelled in this thesis, the road network from the author's specialization project was used, see Section 6.1.

4.1.4 CityEngine

ArcGIS CityEngine[4] is a software used for large-scale modelling of cities. Like with RoadRunner, cities can either be generated from scratch or use real-world data to create a model of an already existing city. One feature in particular that benefited this thesis, is automatic generation of building textures. Although these textures do not reflect real-world textures, they serve as useful placeholders in cases where no real-world textures have been collected. No city models were created this way in the thesis, although the foundation scene from Brekke's Master's thesis contained building meshes automatically textured with CityEngine [1].

4.1.5 Meshroom

Meshroom[5] is an open-source software used for performing photogrammetry. It takes a collection of images as input, and can output reconstructed point clouds, textured three-dimensional meshes, and camera positions via pose estimation. For this thesis, Meshroom was primarily used to reconstruct textured 3D representations of real-world buildings.

[3]https://www.mathworks.com/products/roadrunner.html
[4]https://www.esri.com/en-us/arcgis/products/arcgis-cityengine/overview
[5]https://alicevision.org/

4.1.6 Steam and SteamVR

Steam[6] is an online service for distributing and playing video games, created and owned by video game developer and publisher Valve Corporation. The service ships with its own virtual reality platform called SteamVR,[7] enabling VR gaming on personal computers. Many game engines and 3D development software supports SteamVR, among others Omniverse through the Omniverse XR application.

4.1.7 CloudXR

NVIDIA has developed a platform named CloudXR[8] for streaming XR content between devices. It works by having a server computer running an XR software, sending rendered images to a client computer. The client then displays the images on a screen, be it a smartphone or a tablet screen in the case of AR or an HMD for VR. The connection need not be wired—in fact, the platform has been developed with 5G wireless technology in mind. The Omniverse platform supports CloudXR integration via the Omniverse XR application.

4.1.8 Blender

For 3D editing, troubleshooting, and visualization, the open source 3D creation suite Blender[9] was used. Despite Blender having numerous features spanning animation, digital sculpting, and movie editing, it was primarily used for UV editing (see Section 10).

4.1.9 Adobe Photoshop

Adobe Photoshop[10] is a general-purpose 2D image manipulation program. Photoshop was mainly used for stitching and color-correction of manually collected building textures (see Section 10).

4.1.10 Python

Python [11] was the main programming language used for the development portion of the thesis. The reasoning for this is twofold. Firstly, it is known for its ease of use and high abstraction level. Secondly, it is one of the languages supported by the Connect portion of Omniverse, meaning that custom connectors can be coded with Python.

[6] `https://store.steampowered.com/`
[7] `https://store.steampowered.com/steamvr`
[8] `https://www.nvidia.com/en-us/design-visualization/solutions/cloud-xr/`
[9] `https://www.blender.org/`
[10] `https://www.adobe.com/products/photoshop.html`
[11] `https://www.python.org/`

4.2 Hardware

The main piece of hardware used for development was an MSI Creator 7 computer containing an NVIDIA GeForce RTX 2060 graphics card. In addition, a desktop PC with an NVIDIA GeForce GTX 1080 graphics card was available for use from Department of Computer Science's VR laboratory.

NTNU hosts a research group called NTNU Autonomous Perception Laboratory (NAPLab). [38] The purpose of the group is to conduct research on and develop autonomous vehicles, and adapt these to particularities present in Norwegian driving conditions. NAPLab owns a car equipped with sensors like LiDARs, cameras, and GNSS receivers. Data from these sensors can be used for updating a digital twin in real-time. Specifically, the outputs of the camera and the GNSS receiver were used in this thesis. In addition to the vehicle's camera, a Samsung Galaxy S9+ smartphone was used for photography.

For the virtual reality implementation of the digital twin, the Meta Quest 2 VR suite was used. The hardware consists of a head-mounted display (HMD) and two handheld controllers, without base stations.

4.3 Services

3D Clip&Ship[12] is a service from Geodata enabling users to select areas on a map and export features like 3D buildings, terrain, vegetation, and more from that area. The data can be exported to a variety of formats, among others the OBJ format (see Section 5.1.2). The service also contains a REST API for automated export. Access to this service was granted on a two-month basis from Geodata.

4.4 Assets

As a basis for this thesis, an already existing USD scene of Gløshaugen was used. The buildings themselves are modelled after the real world; the building textures, however, have been automatically generated by CityEngine. Although real-world textures are important for digital twins themselves, they are of lesser priority for the performance of autonomous vehicles. For this purpose, it is sufficient that the textures appear realistic enough that the objects they are applied to can be recognized as buildings by a neural network.

The scene contains a separate object for a partial road network around the Gløshaugen area. This object was modelled in RoadRunner as part of this thesis' specialization project [2]. The model in its isolation can be seen in Figure 4.1. The Gløshaugen model and the road model were also combined as part of that project. The result is used as a foundation for this thesis, and is shown in Figure 4.2. A 3D model of NAPLab's car was also supplied and is shown in Figure 4.3.

[12]`https://www.arcgis.com/apps/webappviewer/index.html?id=304346a17c7b463fa76bd6820e39a184`

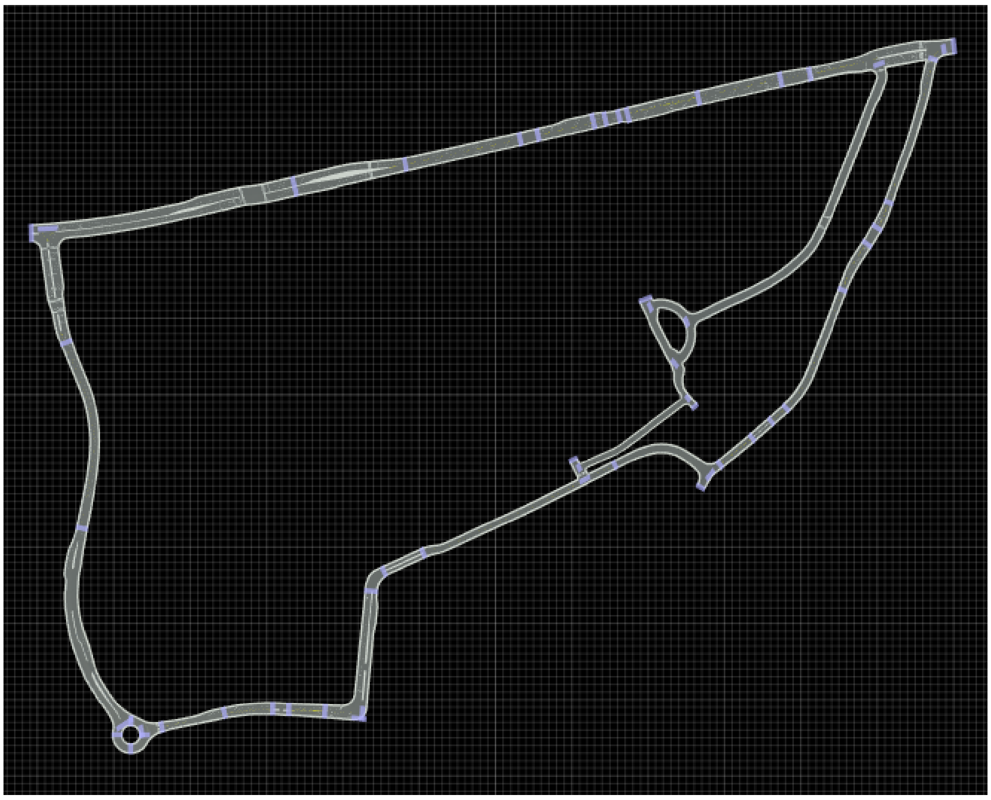


Figure 4.1: The road network from RoadRunner, seen from a top-down perspective.

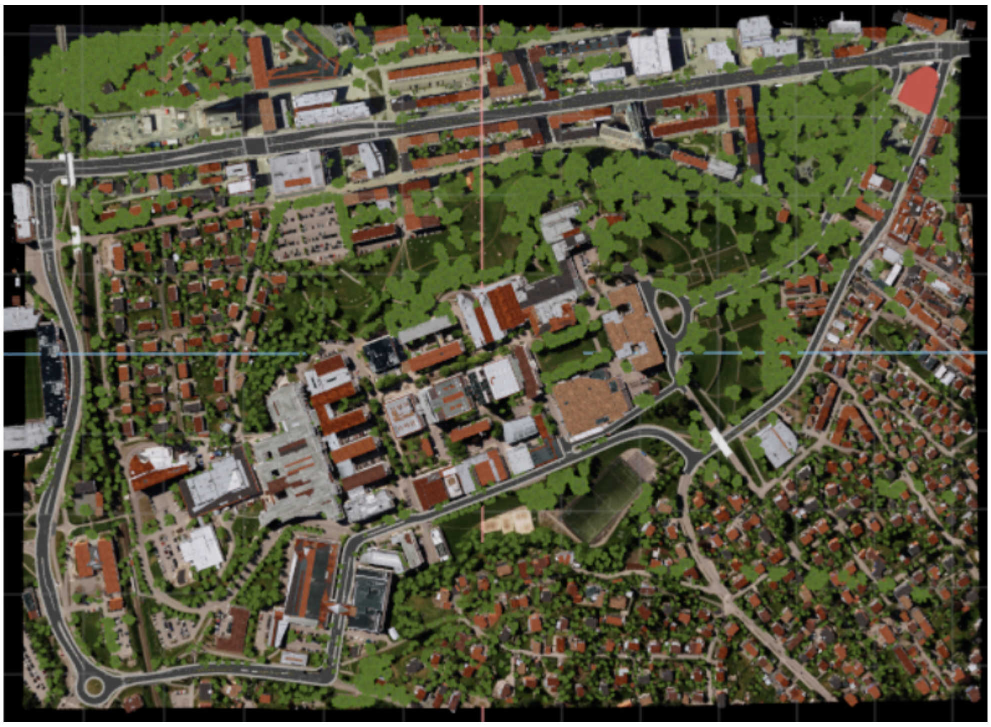


Figure 4.2: The thesis' foundation scene from top-down perspective.

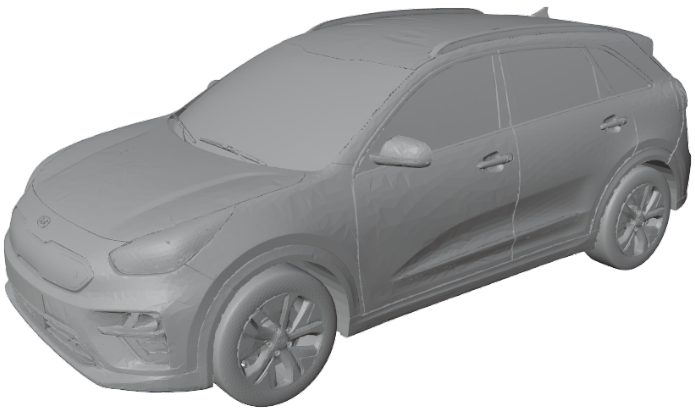


Figure 4.3: 3D model of NAPLab' car.

Chapter 5

Data

In order to create a digital twin, one needs a plethora of data, often originating from different sources. This chapter aims to give an insight into what data is necessary for the approach presented in this thesis.

5.1 File Formats

5.1.1 USD

The Universal Scene Description (USD) file format is arguably the most important file format for this project. All data is eventually collected into a USD scene in Omniverse and stored in USD files.

5.1.2 OBJ

Wavefront OBJ is a common file format to store polygonal three-dimensional meshes. In its most basic implementation, an OBJ file contains information such as object names, the position of each vertex, texture coordinates, normals, and faces. Optionally, an OBJ file can be accompanied by an MTL file describing the objects' surface shading properties. In this project, OBJ files were used as an intermediary file format because of its widespread use in a variety of 3D software ecosystems.

5.1.3 TIFF and GeoTIFF

The Tag Image File Format (TIFF) is a common image file format used to store raster images. One extension to this format, called GeoTIFF, is used to store raster graphics that has a geographical location, such as a map or an orthophoto. Typically, a GeoTIFF file contains data like its corners' geographical coordinates, and which coordinate system the coordinates are represented with.

5.2 Data Description and Sources

5.2.1 Terrain

Terrain data in the form of height maps can be seen as a figurative and literal foundation for the rest of the twin. It represents the ground level of the model, and can be used to generate a three-dimensional mesh of the ground. It also dictates in which altitude to place features like buildings and roads. Such height maps can be exported from the website GeoNorge[1] as GeoTIFF files. A 3D mesh incorporating such a terrain map can be fetched from 3D Clip&Ship in the OBJ format.

5.2.2 Orthophoto

An orthophoto, in the context of geography, is an image of a geographical area taken from a top-down perspective. An orthophoto is typically taken from a satellite or an airplane flying over the area. Such a photo can be draped over a terrain mesh to show ground colors, and can also act as a reference image when modelling road networks.

The web service Norge i bilder[2] has been identified as the main source of orthophotos. This service offers photos with resolutions up to 4 cm × 4 cm. These photos are also used as roof textures on buildings exported from the Clip&Ship service. Alternatively, even higher resolution images can be collected from aerial drone footage.

5.2.3 3D Buildings

The main source of 3D mesh data for buildings for this project is 3D Clip&Ship. The data can be exported as clean, low-polygonal meshes as opposed to more noisy meshes resulting from photogrammetry. The data was exported with the OBJ file format. Optionally, the buildings can be exported with roof textures.

5.2.4 Road Metadata

Road metadata, in the context of digital twins, involves data such as center lines, the number of road lanes and their positions and sizes, driving direction, speed limits, and more. For this thesis, this data was manually created by modelling the roads in the RoadRunner software. Terrain and orthophoto were needed in order to model them accurately in all three spatial dimensions.

[1] `https://www.geonorge.no/`

[2] `https://norgeibilder.no/`

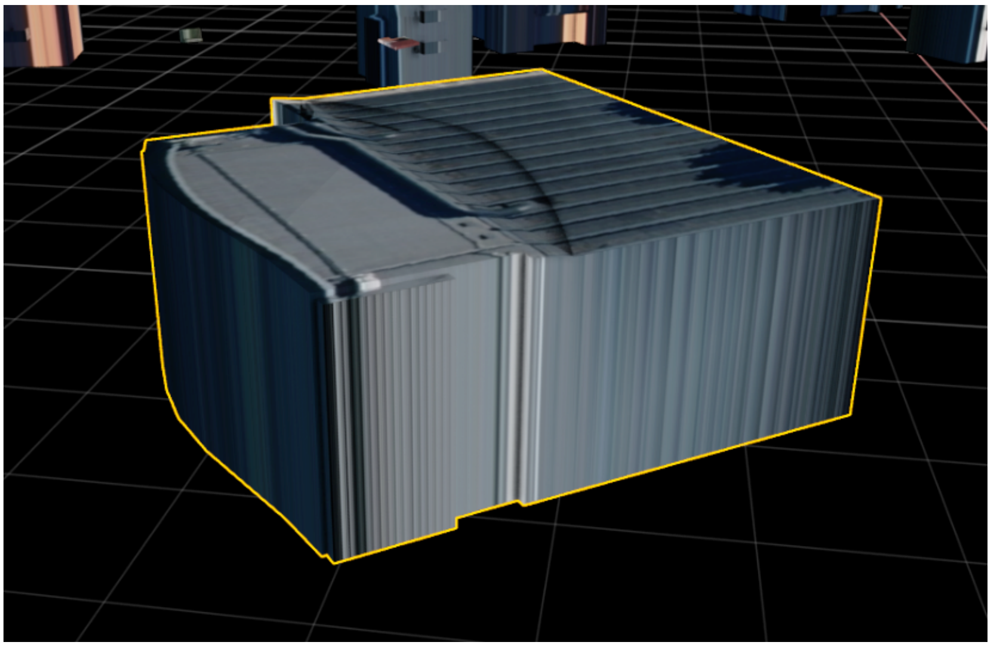


Figure 5.1: An example of a building that has been textured by draping an orthophoto on the mesh.

5.2.5 Building Textures

The 3D Clip&Ship service contains wall and roof textures for its building meshes. However, for the textures to be useful for training autonomous vehicles, they need to have a certain level of realism. Figure 5.1 shows an example of a textured building taken directly from the 3D Clip&Ship export. The building is satisfactorily textured on the roof, but the walls contain a striped pattern instead of the real-world wall texture. This phenomenon appears when the texel values from the edges of the roof are repeated across the entire wall surface in the Y axis. This is done because the buildings have been draped with the orthophoto only, and thus lack proper wall textures.

An alternative wall texturing scheme was used. First, CityEngine was used to automatically generate textures for the building meshes, so as not to leave any building textureless. This was performed by R. Brekke in their Master's thesis [1]. For real-world textures, an experiment using manually collected images was performed using a smartphone camera, see Chapter 10. If wall texturing was to be automated, video frames from NAPLab's on-board cameras could be used. Images from such cameras were also used in the automated photogrammetry method presented in Chapter 11.

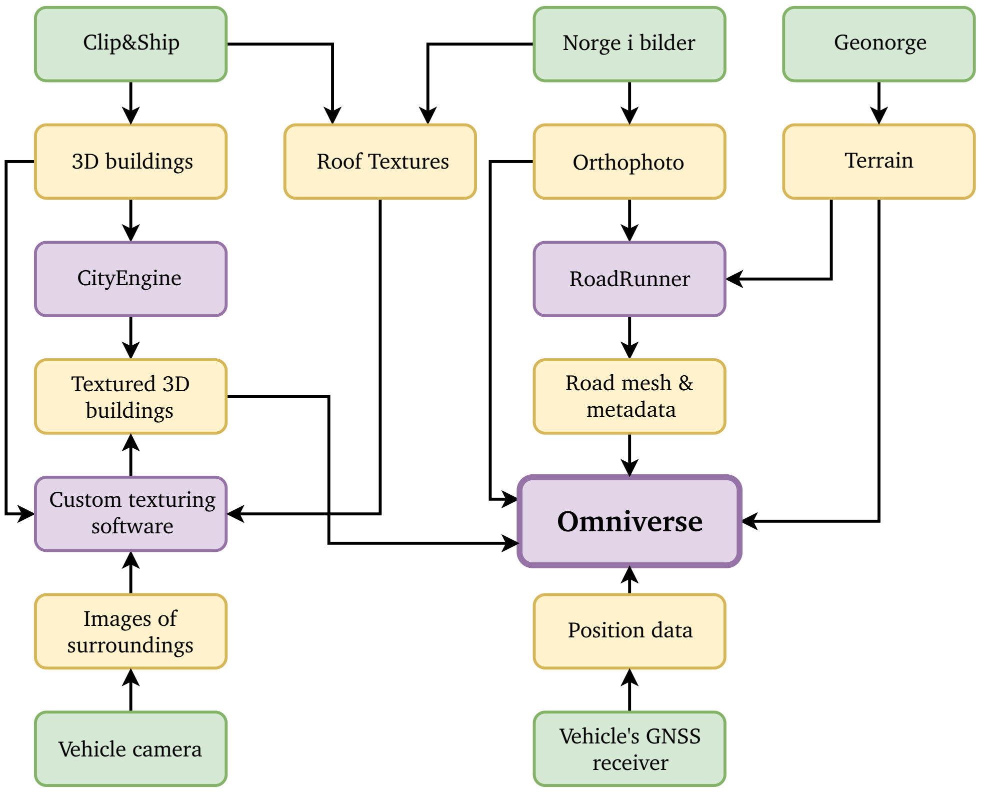


Figure 5.2: Flowchart illustrating the data flow of this thesis' digital twin. Yellow boxes represent data, green boxes represent data sources, and purple boxes represent software used to manipulate data. An arrow pointing from a data source to a piece of data denotes that the data can be retrieved from that data source. An arrow pointing from a piece of data to a software denotes that the data is used as input to that software. An arrow pointing from a software to a piece of data denotes that the data is an output after being manipulated by that software.

5.3 Data Flow

This section aims to give an overview of how data is processed, and which software and services are used to perform this processing.

3D buildings are imported to CityEngine, and textures are automatically generated and applied. Alternatively, images from the surroundings as well as roof textures from orthophotos are used for texturing. The resulting objects are then imported to Omniverse. Orthophotos and terrain meshes are used directly in Omniverse, as well as in RoadRunner. In Roadrunner, they are used to create a 3D model of the road network, along with simulation metadata. These models are subsequently imported into Omniverse. Position data from a GNSS receiver is used to update locations of vehicle objects in the USD scene.

Chapter 6

Scene Preparation

6.1 Road Alignment

In order to have a sound foundation to send and receive live data, having an accurate three-dimensional model of the area is an important prerequisite. In the foundation assets from the specialization project, the road network model from RoadRunner is not properly aligned to the rest of the scene. In that project, the road network was manually transformed so as to approximate the correct location. As part of this thesis, to achieve a more accurate alignment, the road network was aligned to the rest of the scene via georeferencing.

The main problem to overcome was the fact that the two scenes use different coordinate systems. The main scene uses coordinates derived from the UTM33N system. During Brekke's thesis, while exporting to USD, the coordinates were set to use a certain offset, so as to align the geometric center of the scene to the coordinates $(0, 0, 0)$. The reason for this was to avoid performance problems in USD due to large numerical values of the UTM coordinates. Using an offset will ensure that the coordinate values remain small enough not to cause performance issues. The offsets can be seen in Table 6.1. Note that the absolute values of the longitude and latitude values from Table 6.1 are equal to the center coordinates of the main scene. It was also noted that positive X values denote eastwards direction, positive Y values denote upwards in altitude, and positive Z values denote southwards.

The RoadRunner scene is expressed in a derivation of the WGS84 coordinate system. This system also uses an offset to define its center. In order to incorporate this into the USD scene, the coordinates were converted to the UTM33N coordin-

Coordinate	**USD axis**	**Offset**
Longitude	X	−270630.659
Latitude	Z	−7040355.576
Altitude	Y	−46.670

Table 6.1: Table showing the coordinate offsets of the main scene, expressed in UTM33N coordinates.

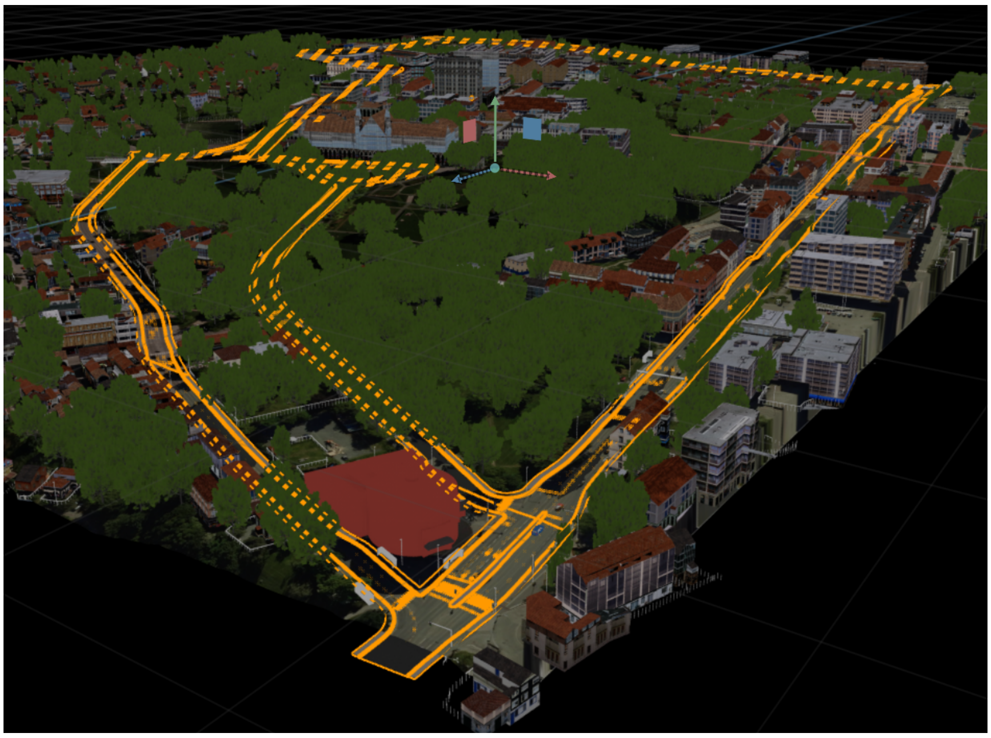


Figure 6.1: The result of combining the CityEngine scene with the RoadRunner road network by geographic projection. The road network mesh is highlighted in orange.

ate system. The scene's coordinate system is explicitly defined using a custom projection string which can be used to convert between the two coordinate systems. This string also explicitly defines the coordinates of the scene's center, meaning that offsets will automatically be taken into account when projecting to UTM.

Then, in order to properly align the road network to the rest of the USD model, the same offset as in Table 6.1 has to be applied to the longitude and latitude coordinates. This projection and offset was performed on all vertices of the road network mesh, on the X and Z coordinates specifically. Finally, the entire road network object was translated by the Y offset from Table 6.1 to align the road network in altitude. The result is shown in Figure 6.1.

6.2 3D Car Model

The car model shown in Figure 4.3 was used as a visual cue for received position data (see Chapter 7 for more details). However, the raw model file was deemed to be excessively dense in terms of geometry, containing almost 1.2 million faces. Therefore, using Blender, the model was decimated to have 1 % of its original face count. It was also scaled by a factor of 0.01, from centimeter coordinates to meters. The origin of the object was moved to coincide with the location of the

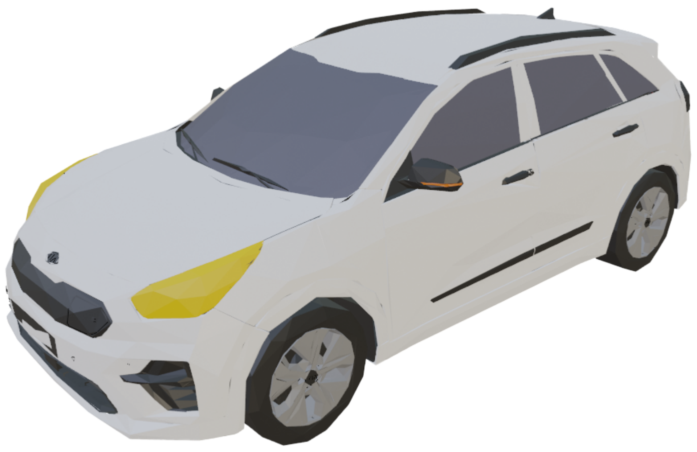


Figure 6.2: 3D model of NAPLab's car after preparing for use in Omniverse.

GNSS receiver. The faces were also crudely colored. The resulting model is shown in Figure 6.2. Finally, the object was exported to the OBJ file format and imported into Omniverse.

Chapter 7

Position Data

This chapter explains the development undertaken while attempting to answer Research Question RQ1 *(Can dynamic location data of a physical entity be transferred to a digital 3D scene in order to update it in real-time?)*.

In addition to the pre-packaged connectors in the Omniverse Connect module, Omniverse features an example connector with code as a basis to create custom connectors. It can be accessed by downloading 'Connect Sample Omniverse Connector' from the Omniverse launcher. It contains example code in both C++ and Python, and shows how to programmatically create and live-update an Omniverse scene. The base code acted as a foundation to develop a custom connector to receive data from NAPLab's autonomous car.

The data received from NAPLab's GNSS sensors are given in the WGS84 coordinate system. Incorporating the data into the digital model necessitates converting the coordinates from WGS84 to the USD scene's coordinate system. The coordinates were offset with the values from Table 6.1 in order to be aligned with the rest of the scene. Because north is defined as negative Z in the scene, the Z values were inverted after conversion.

The 3D model of NAPLab's car, as introduced in Section 6.2, was used in the USD scene. This model's location was programmed to be updated based on the location values received. For prototyping, coordinate values were read from a CSV file at a pre-determined time interval.

In cases where altitude readings are not available, the altitude of the car was calculated using a height map of the area. The height map was downloaded from Geonorge, and has a resolution of $10\,\text{m} \times 10\,\text{m}$. Using this map directly would, because of its low resolution, cause the car model to abruptly change altitude at regular intervals of space. In order to ensure smoother transitions in the height values, the map was interpolated using bicubic interpolation. Samples were then taken from the interpolated map using UTM coordinates, and the altitude of the car model was updated with the sampled values.

The GNSS readings were also used to rotate the car to coincide with its driving direction. This problem can be reduced to finding the necessary rotation to align the Z axis of the car to the vector that represents the difference between two temporal-adjacent sets of coordinates. The concept is illustrated in Figure 7.1. Worth noting is that a GNSS reading is only considered for rotation if it is different than the previous reading.

We use one vector $\overrightarrow{v_1}$ which is equal to the unit vector along the Z axis. Then, for every coordinate reading, we construct a vector $\overrightarrow{v_{2,i}}$ which is the difference between the current (c_i) and the previous unequal (c_{i-1}) coordinate reading:

$$\overrightarrow{v_1} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\mathsf{T} \tag{7.1}$$

$$\overrightarrow{v_{2,i}} = c_i - c_{i-1},\ i = 2, \dots, n \tag{7.2}$$

where n is the total amount of coordinate readings in which no two temporal-adjacent readings are equal. Then, for each reading, we compute:

$$\overrightarrow{v_{xyz,i}} = \overrightarrow{v_1} \times \overrightarrow{v_{2,i}} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^\mathsf{T} \tag{7.3}$$

$$w_i = \sqrt{\left|\overrightarrow{v_1}\right|^2 \cdot \left|\overrightarrow{v_{2,i}}\right|^2} + \overrightarrow{v_1} \cdot \overrightarrow{v_{2,i}} \tag{7.4}$$

where $\overrightarrow{v_{xyz,i}}$ and w_i are the vector and scalar parts of the quaternion respectively. Concatenating this will give a quaternion representing the rotation from $\overrightarrow{v_1}$ to $\overrightarrow{v_{2,i}}$:

$$q_i = \left(w_i, \overrightarrow{v_{xyz,i}}\right) = \left(w_i, \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^\mathsf{T}\right) \tag{7.5}$$

Using this method will, however, give an infinite amount of solutions. The Z axis of the car will be aligned to the difference vector $\overrightarrow{v_{2,i}}$; however, the car can still be rotated an arbitrary amount along its own Z axis and the solution would still be valid. Therefore, using the assumption that the car will drive on near-flat roads most of its journey, we disable rotation along the X axis by setting $x_i = 0$:

$$q_{wyz,i} = \left(w_i, \begin{bmatrix} 0 & y_i & z_i \end{bmatrix}^\mathsf{T}\right) \tag{7.6}$$

Normalization is then applied to give the following quaternion:

$$q_{n,i} = \frac{q_{wyz,i}}{\left|q_{wyz,i}\right|} \tag{7.7}$$

Finally, to smooth out the rotations applied on the car model, a running average is taken using the last ten quaternions:

$$q_{f,i} = \frac{\sum_{j=0}^{9} q_{n,i-j}}{10} \tag{7.8}$$

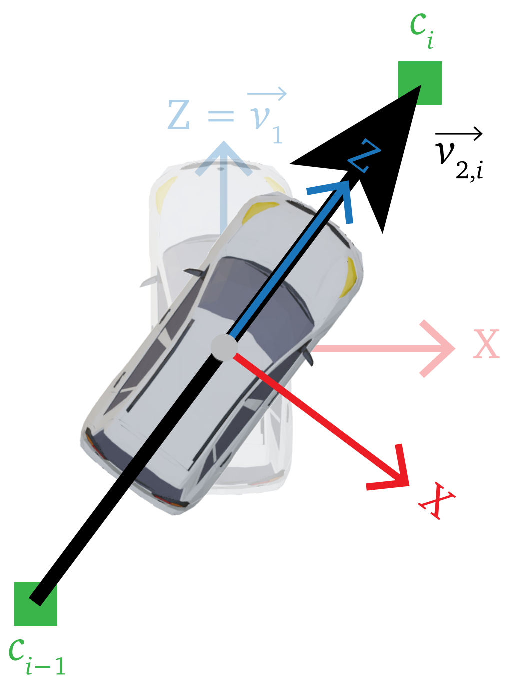


Figure 7.1: Figure illustrating the concept of using two temporal-adjacent GNSS readings to rotate the vehicle model.

Chapter 8

Building Geometry

This chapter addresses the acquisition of building meshes from Research Question RQ2 *(Can a three-dimensional model of a geographical area be acquired or created, especially with terrain, buildings, and roads; and to what degree can this be automated?)*. Terrain meshes can be acquired and processed using the same method.

In order to avoid the 3D meshes of buildings to become outdated as real life construction and demolition takes place, a pipeline was created to automatically download and integrate updated building models. The models are acquired from the 3D Clip&Ship service from Geodata. Because the service comes with a REST API, the process of fetching the buildings can be automated. The API accepts a number of settings to determine what to export, among others:

- A list of coordinates that constitutes the external boundaries of the area to be exported;
- The geographic projection that the data should be represented with;
- What kind of data to export, for example 3D buildings, terrain model, vegetation, and orthophotos;
- File formats for the resulting export.

See also Appendix C for an API reference.

8.1 Geometry Retrieval

In order to access the Clip&Ship service, a user account was created on Geodata's online platform called Geodata Online. Subsequently, a Geodata employee granted access to the service on a two-month basis. In addition, because a download link to the finished export is sent by mail, a dedicated email account was created. Omniverse Drive was also set up in order to access Omniverse Nucleus via a Windows drive. Optionally, one can run a script to encrypt and locally store the passwords to the Geodata Online platform and the email client.

The data were subsequently downloaded and imported to Omniverse Create. The steps performed are outlined in Sequence 8.1. A corresponding sequence diagram can be seen in Figure 8.1.

Sequence 8.1 - Automatic Retrieval of Building Meshes

1. *If any passwords were locally stored, they were decrypted and loaded.*
2. *An access token was generated at an API endpoint. Usage of all other API endpoints requires the presence of such a token. Jobs are assumed to take less than 24 hours to compete; thus, the token was configured to be valid for 24 hours.*
3. *An export job was submitted to the Geodata servers. Among the specified settings were UTM33N for the coordinate system and OBJ for the file format (Clip&Ship cannot export to USD, but Omniverse can import OBJ files).*
4. *The status of the job was checked every 10 minutes. The API provides an endpoint specifically for this task. The status check lasts as long as the job has status as uncompleted.*
5. *The email inbox was checked every minute. This continuous check is necessary because empiric experiments revealed that a few minutes may pass from the status turns completed to the email is received.*
6. *The download URL was fetched from the received email.*
7. *The files were downloaded and uncompressed.*
8. *The files were copied to the local cache. Omniverse will cache OBJ files automatically when adding a reference to them; however, it will not cache accompanying MTL files, which are necessary to render meshes properly.*
9. *The files were uploaded to Omniverse Nucleus via Omniverse Drive.*

8.2 Building Georeferencing

After acquiring all relevant files and uploading them to Nucleus, the 3D meshes were georeferenced. The exported files contains data to perform this transformation automatically. Directly from the export, the 3D models have their center at the origin $(0, 0, 0)$, and they are scaled so that all coordinate values lie in the range $[-0.5, 0.5]$. A file called `global.fwt` contains a 4×4 transformation matrix to transform these coordinates back to the original coordinate system.

The steps in Sequence 8.2 were taken to georeference the 3D buildings to the USD scene. An illustration of the transformations applied are shown in Figure 8.2.

Sequence 8.2 - Automatic Georeferencing of Building Meshes

1. *A reference to the relevant OBJ file was created in Omniverse Create.*
2. `global.fwt` *was parsed into a transformation matrix.*
3. *The translation portion of the transformation matrix was modified to take the offsets from Table 6.1 into account.*
4. *The Y and the Z components of the translation were swapped. Where the Clip&Ship export uses Z as its up axis, the Omniverse scene uses the Y axis.*
5. *The matrix was combined with a rotation matrix of* $270°$ *along the X axis to account for the inconsistency of which coordinate axis points upwards.*
6. *Finally, the transformation was applied on the 3D meshes.*

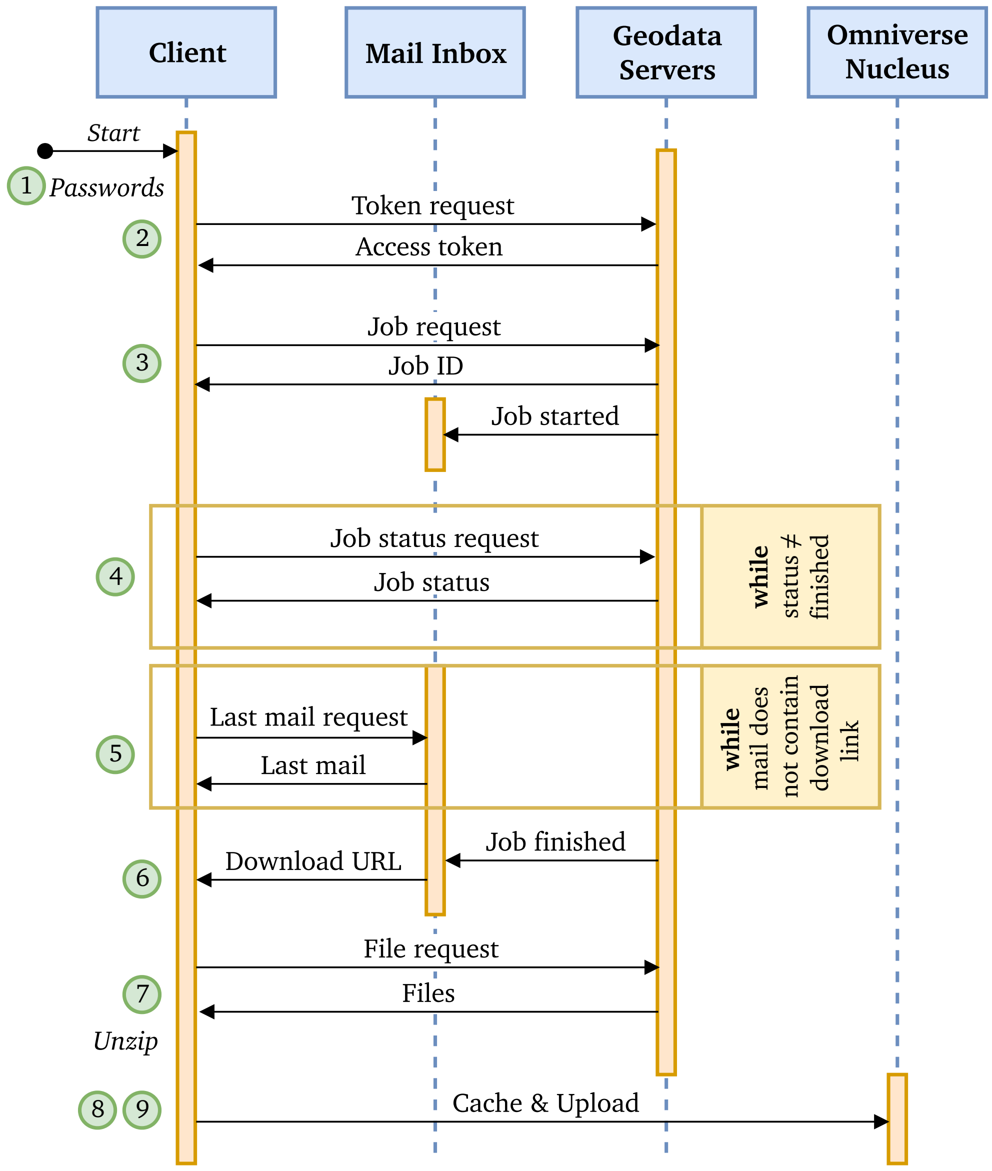


Figure 8.1: Sequence diagram of the pipeline for fetching updated 3D meshes of buildings. The numbers in the green circles correspond to the list numbers in Sequence 8.1.

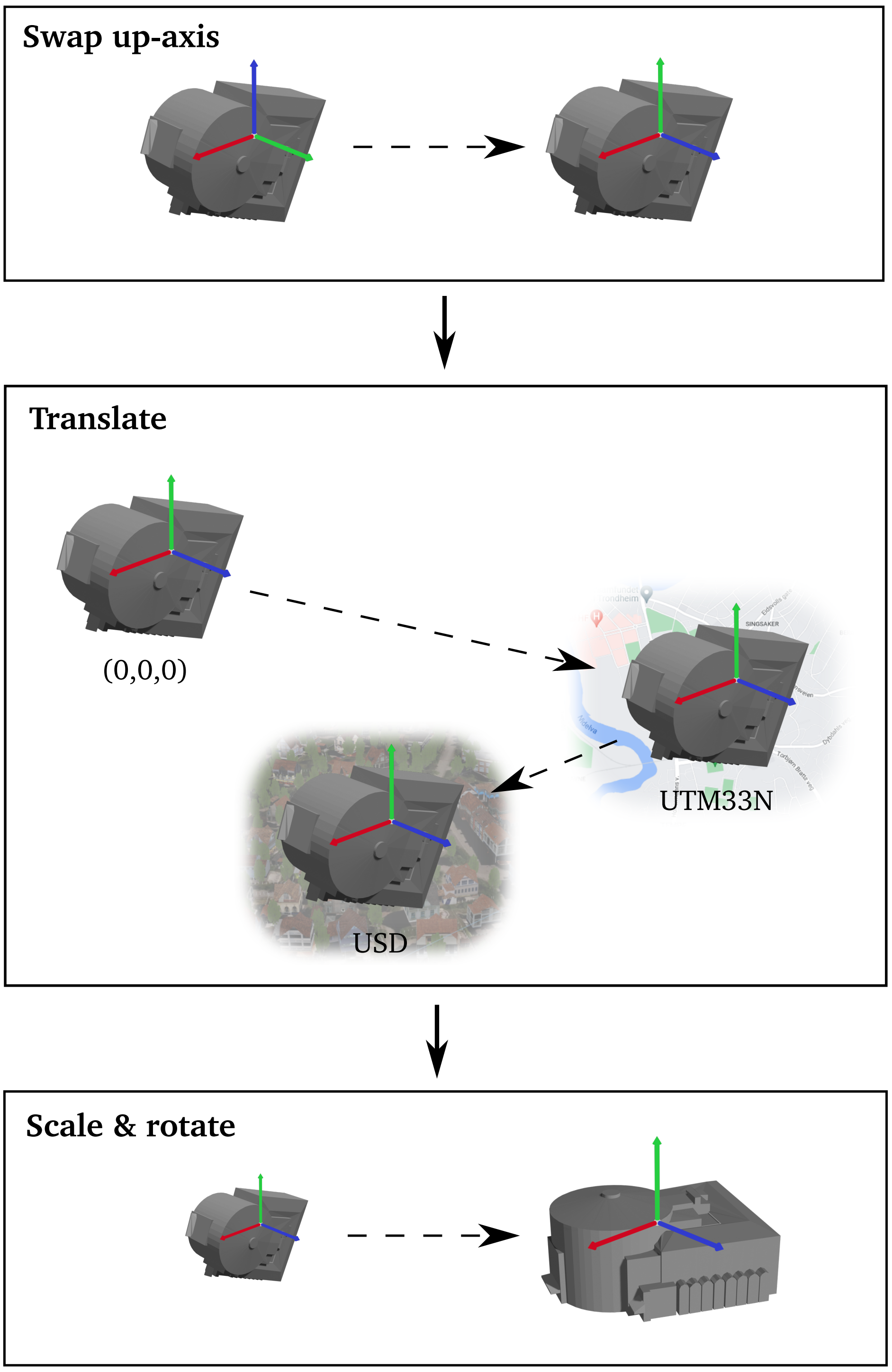


Figure 8.2: Summary of the transformations performed on the building meshes after importing to Omniverse Create. The red, green, and blue arrow widget represent the world coordinate system's X, Y, and Z axes respectively.

For the rest of this section, when referring to the X, Y, and Z axes, we will take that to mean those axes belonging to the world coordinate system. The transformation matrix applied to the buildings can be expressed like this:

$$\mathbf{M} = \mathbf{R}_x(\theta) \cdot \mathbf{S} \cdot \mathbf{T} \tag{8.1}$$

where $\mathbf{R}_x$ is a rotation matrix along the X axis, θ is the angle of rotation along the X axis, $\mathbf{S}$ is a scale matrix, and $\mathbf{T}$ is a translation matrix. Going in a bit more detail, we give the definitions of rotation, scale, and translation matrices:

$$\mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.2}$$

$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.3}$$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.4}$$

Here, s_x, s_y, and s_z are the scaling factor along the X, Y, and Z axes respectively; and t_x, t_y, and t_z denote the translation along the X, Y, and Z axes respectively. We define the translations to be the net translation from the origin to the correct position in the USD scene, expressed in UTM33N coordinates. We assume that

$$s := s_x = s_y = s_z \tag{8.5}$$

so as to avoid stretching of the exported meshes. We also must take into consideration that when translating, the Y and Z axes must be swapped in order to accommodate for the inconsistency of which axis is the up axis. Finally, with a rotation of $270° = \frac{3}{2}\pi$ along the X axis, we set

$$\theta := \frac{3}{2}\pi \tag{8.6}$$

This gives us the following modified matrices:

$$\mathbf{R}'_x\left(\frac{3}{2}\pi\right) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.7}$$

$$\mathbf{S}' = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.8}$$

$$\mathbf{T}' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_z \\ 0 & 0 & 1 & t_y \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.9}$$

Multiplying these matrices together gives us the final transformation matrix:

$$\mathbf{M}' = \mathbf{R}'_x \cdot \mathbf{S}' \cdot \mathbf{T}' = \begin{bmatrix} s & 0 & 0 & st_x \\ 0 & 0 & s & st_y \\ 0 & -s & 0 & -st_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8.10}$$

Chapter 9

Roof Texturing

This chapter marks the start of examining Research Question RQ3 *(Can any photo of a building be applied as a texture to a corresponding 3D object so that the building looks realistic enough to be near indistinguishable from reality?)*. Specifically, textures are applied to the roofs of 3D buildings. The method outlined here uses the exported building meshes from Chapter 8, and replaces their roof textures automatically using georeferenced images. The new textures should ideally be of higher resolution than those already present.

For roof texturing, Clip&Ship has an option for exporting building meshes with pre-applied roof textures. These are the highest resolution textures from the web service Norge i bilder (4 cm × 4 cm). In theory, even higher resolution textures can be collected by, for example, UAVs. As a proof of concept, Python code was developed to automatically apply textures to the building meshes from Section 8. The textures can theoretically come from any source; however, two main assumptions must be fulfilled in order for the code to produce the correct results:

1. The building meshes extracted in Chapter 8 must be exported with already existing roof textures, and without wall textures. The script assumes that all textured faces belong to the roof part of the mesh. This is simply a way to distinguish between the roof and the rest of the mesh.
2. The image to be used as a texture must already be georeferenced.

The method involves transforming vertex coordinates to UTM33N using the `global.fwt` file. Coordinates are then normalizing them to the range $[0, 1]$ relative to the corners of the texture, which are also given in UTM33N. The normalized coordinates are used directly as UV coordinates. The process is summarized here and in Figure 9.1:

Sequence 9.1 - Roof Texturing

1. *The OBJ file was read, and data like materials, vertex coordinates, and already existing UV coordinates were extracted.*
2. *For each material, vertex and UV coordinates belonging to each face of that material were found.*

3. *Vertex coordinates were transformed to UTM33N using* `global.fwt`.
4. *A bounding box was computed for each material using the vertices' UTM33N coordinates.*
5. *The image textures were cropped to the extent of its respective bounding box.*
6. *The original textures were overwritten by the new cropped textures.*
7. *Vertex coordinates were normalized to the range* $[0,1]$ *based on each material's bounding box, and used directly as the new UV coordinates.*
8. *The OBJ file was overwritten with the new UV coordinates.*

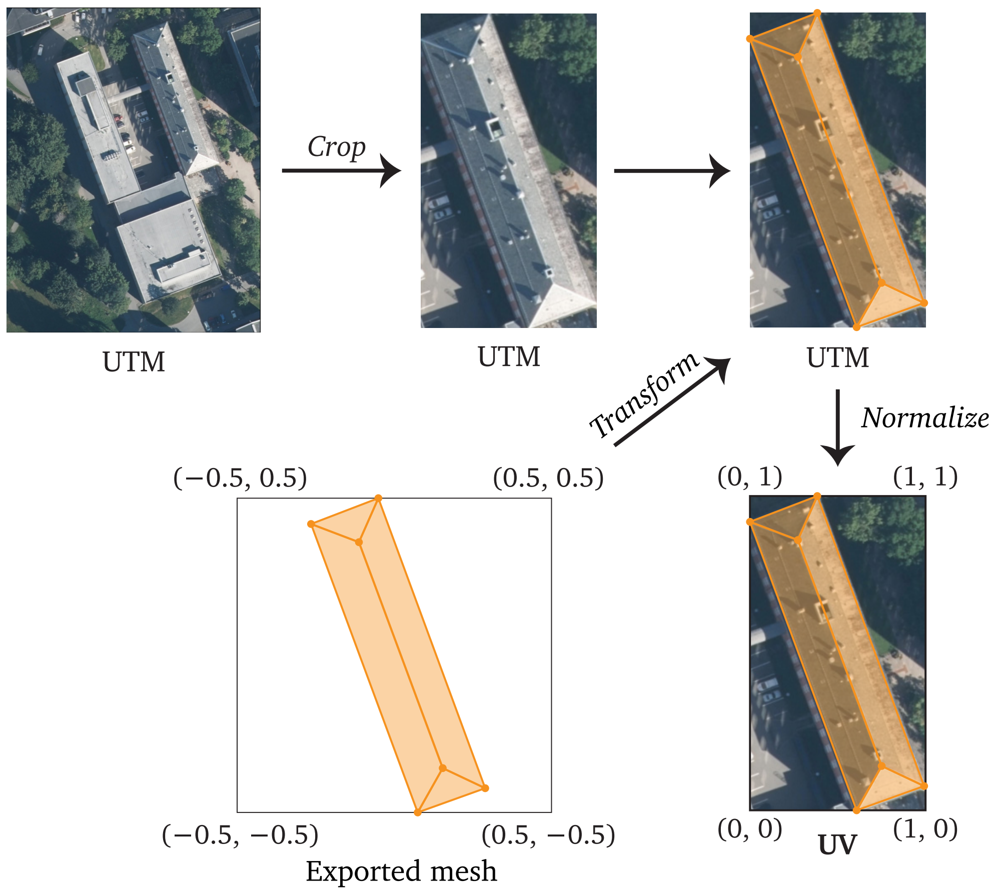


Figure 9.1: Figure illustrating the concept of applying roof textures automatically.

Chapter 10

Projective Wall Texturing

This chapter will mainly focus on answering Research Question RQ3 *(Can any photo of a building be applied as a texture to a corresponding 3D object so that the building looks realistic enough to be near indistinguishable from reality?)*. Some initial insight is also given into Research Question RQ4 *(Can we use photos taken from a driving car to update building textures automatically in real-time?)*.

For a geographical digital twin to be useful for neural network training and validation, it is important that the buildings look as realistic as possible. Wall textures need to have a sufficiently high resolution, and should contain features commonly found on walls like windows, balconies, and doors. neural networks should be able to distinguish these features and recognize that the buildings are, in fact, buildings. Automatically generated textures, like those generated by City-Engine, are deemed to fulfill these criteria. However, in order for the scene to be a true digital twin, it is important that the textures on the buildings mirror those present in the physical world.

Ideally, this would be done automatically by one or more cameras moving in the physical world. This is a difficult problem compounded of numerous sub-problems. Therefore, experiments were conducted to examine the feasibility of texturing buildings. The experiments were conducted in the form of performing the texturing manually, in order to get an overview of the necessary steps involved in the process. All building models used in this chapter were exported from 3D Clip&Ship with the procedure outlined in Chapter 8.

10.1 Texture Mapping Experiment

First, a single flat surface of a single building was photographed. The surface in question was chosen to be the south facing wall of the building at Sem Sælands vei 5, located at coordinates

$$\text{N}63°25'0.46'', \text{E}10°24'12.51'' \tag{10.1}$$

Ideally, a texture for square-shaped wall should also be square-shaped. However, because of perspective distortion occurring when viewing a wall at an angle, walls that are square in the physical often appear as irregular quadrilaterals in images. An example of this can be seen in Figure 10.1. This effect can become quite large when dealing with photos taken from a camera-equipped vehicle, especially when only using front- or back-facing cameras. These cameras are oriented parallel to the driving direction, while building walls often face perpendicular to the driving direction. Considering this, a qualitative experiment was conducted in order to compare the effects of viewing angles to the quality of the resulting texture. Sequence 10.1 shows the steps comprising the experiment, and the results are shown in Section 13.1. Figure 10.2 shows a screenshot from the procedure.

Sequence 10.1 - Texture Mapping Experiment

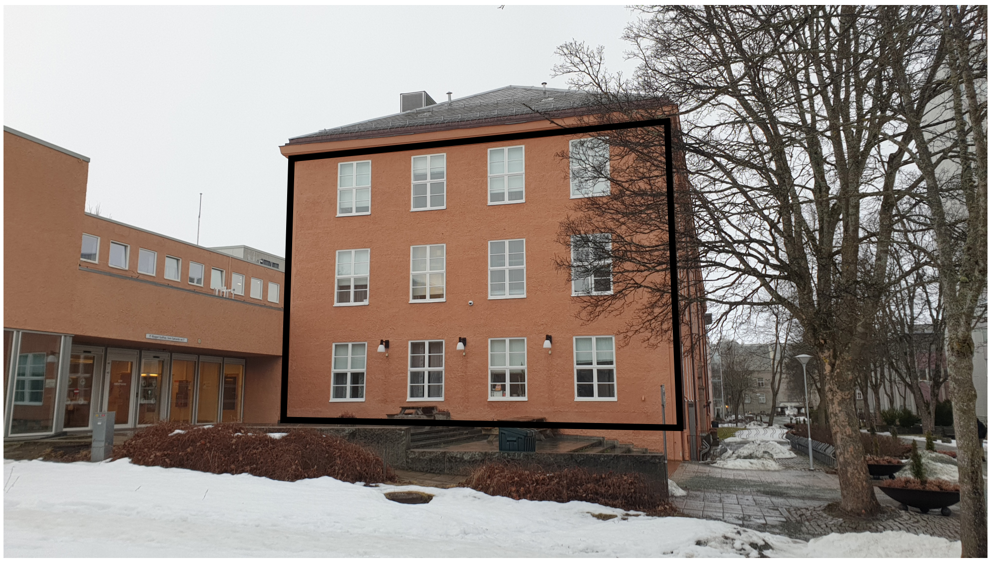
1. *The wall located at the coordinates in Equation 10.1 was photographed at three different angles.*
2. *The photos were imported to Blender, along with a 3D mesh of the corresponding building.*
3. *The relevant face was isolated from the mesh.*
4. *A new material was added to the mesh, and set to sample colors from a texture.*
5. *Blender's camera was aligned to point at the face, parallel to the face's normal.*
6. *While looking through the camera, the command 'Project from View' was executed. This enables UV editing in Blender's UV editor.*
7. *The face's UV coordinates were edited to align with the building wall.*
8. *The image was rendered and saved.*

10.2 Perspective Correction

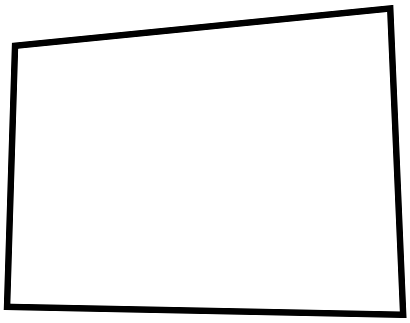
Using images as textures without performing corrections may result in parallel lines becoming non-parallel (see Figure 13.1 for an illustration, and Sections 14.2 and 14.3 for a discussion on the phenomenon). An algorithm was created for correcting the images (see Algorithm 10.1). The algorithm takes as input the four corners of a square—usually a single building wall—as well as an image. From those four corners, it corrects the perspective distortion of the input image. The image is subsequently cropped to those same corners in order to simplify the assignment of UV coordinates. Cropping this way means that the texture will cover the entire wall, meaning that in most cases, the only UV coordinates that need be assigned are $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$ for each respective corner.

The algorithm computes the lengths of the top and left edges respectively, and uses these as the width and height of the output. It is assumed that these edges are hypotenuses in their respective triangles. We can therefore use the Pythagorean equation to compute the length of the hypotenuses:

$$l = \sqrt{(\Delta x)^2 + (\Delta y)^2} \tag{10.2}$$



(a) Photo of a wall outlined in black.



(b) Wall outline without photo.

Figure 10.1: Example of a square wall projected to an irregular quadrilateral when taking a photo.

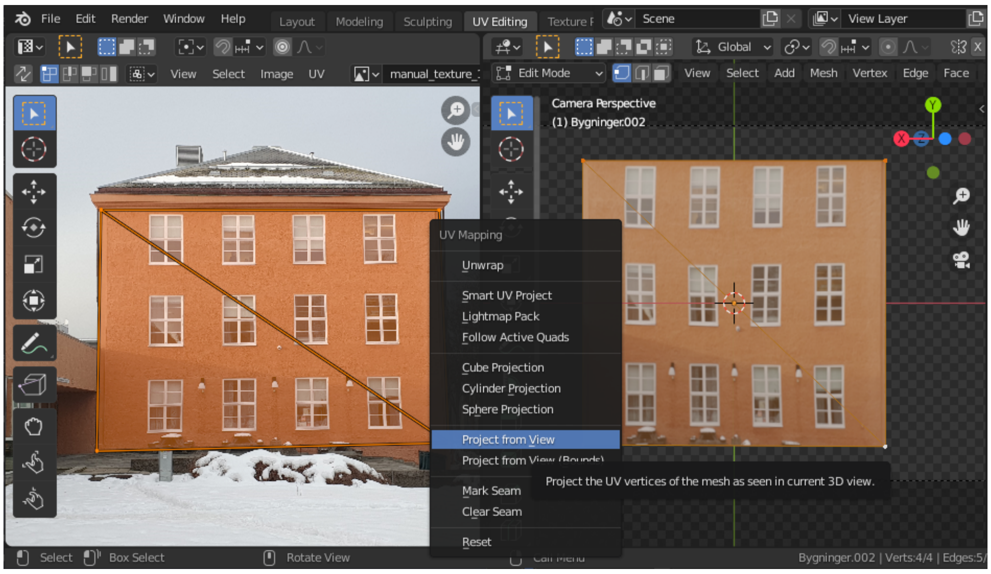

File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture P
Scene
View Layer
View Select Image UV
manual_texture
Edit Mode
Global
View Select Add Mesh Vertex Edge Face
Camera Perspective
(1) Bygninger.002
UV Mapping
Unwrap
Smart UV Project
Lightmap Pack
Follow Active Quads
Cube Projection
Cylinder Projection
Sphere Projection
Project from View
Project from View (Bounds)
Mark Seam
Clear Seam
Reset
Project the UV vertices of the mesh as seen in current 3D view.
Select Box Select Rotate View
Bygninger.002 | Verts:4/4 | Edges:5/

Figure 10.2: Screenshot from the UV editing process in Blender. The UV editor is shown in the leftmost subwindow with UV vertices aligned to the building's corners.

The concept is illustrated in Figure 10.3.

Once the corners of the output image are computed, we apply a plane-to-plane homography to the image. In the rest of this section, we will use the term 'input coordinates' to mean a wall corner's image coordinates before homography is applied, and 'output coordinates' to mean its image coordinates after application.

First, a transformation matrix is computed using the input and output corners. To compute the matrix, we set up four matrix equations, one for each corner:

$$\mathbf{d}_i = \mathbf{P} \cdot \mathbf{s}_i, i = 1,2,3,4 \tag{10.3}$$

where $\mathbf{d}_i$ is the ith corner's output coordinates; $\mathbf{s}_i$ is the ith corner's input coordinates; and $\mathbf{P}$ is the homography matrix needed to transform the input coordinates to the output coordinates. The equations can alternatively be written like

$$\begin{bmatrix} c_i x_i' \\ c_i y_i' \\ c_i \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & t_1 \\ a_3 & a_4 & t_2 \\ p_1 & p_2 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, i = 1,2,3,4 \tag{10.4}$$

Here, (x_i, y_i) are the input coordinates of the ith corner; (x_i', y_i') are the output coordinates of that corner; and c_i is a scaling coefficient. In $\mathbf{P}$, a_1–a_4 denotes affine transformation parameters like rotation, scaling and shearing; t_1 and t_2 are translation parameters; and p_1 and p_2 are projection parameters [39]. Solving these equations for $\mathbf{P}$ will give the final transformation matrix. The results of applying perspective correction are presented in Figure 13.1.

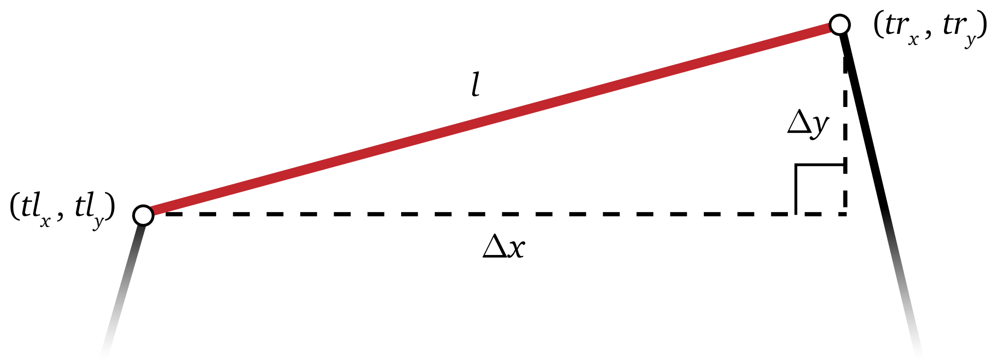


Figure 10.3: Illustration of treating edges of a wall as hypotenuses to find the dimensions of the output. Width derivation is shown here as an example. (tl_x, tl_y) denotes the top left corner of the wall, and (tr_x, tr_y) denotes the top right corner. Δx and Δy denote the differences in x and y coordinates, respectively, between the top left and the top right corner. Finally, l is the hypotenuse in the right triangle with Δx and Δy as its catheti. The equivalent approach is used for calculating the height of the output.

Algorithm 10.1 Perspective Correction

```
$img \leftarrow \text{LOAD\_IMAGE}()$

$tl, tr, br, bl \leftarrow img.input\_corners$
$w \leftarrow \sqrt{(tl_x - tr_x)^2 + (tl_y - tr_y)^2}$
$h \leftarrow \sqrt{(tl_x - bl_x)^2 + (tl_y - bl_y)^2}$

$x_{min} \leftarrow tl_x$
$x_{max} \leftarrow tl_x + w - 1$
$y_{min} \leftarrow tl_y$
$y_{max} \leftarrow tl_y + h - 1$

$output\_corners \leftarrow ((x_{min}, y_{min}), (x_{max}, y_{min}), (x_{max}, y_{max}), (x_{min}, y_{max}))$
$m \leftarrow \text{COMPUTE\_MATRIX}(img.input\_corners, output\_corners)$
$\text{TRANSFORM\_IMAGE}(img, m)$
$\text{CROP\_IMAGE}(img, x_{min}, x_{max}, y_{min}, y_{max})$
```

10.3 Manual Texturing in Omniverse

A single perspective-corrected texture was then applied to the corresponding mesh face in Omniverse Create. This software lacks the user interface to edit individual vertices and texture coordinates, meaning that this must be performed programmatically. As a proof of concept, the following steps were performed in order to use an image to texture a single face of a building;

Sequence 10.2 - Manual Texturing

1. *The relevant image was uploaded to Nucleus.*
2. *The corresponding mesh's existing material was modified to sample colors from a texture.*
3. *The 'points' attribute of the mesh, containing all its vertex coordinates, was edited and immediately reverted. This creates a 'delta' in that mesh (i.e. a change from the original OBJ reference). This is necessary in order to access the 'points' attribute via code.*[1]
4. *For each vertex, a cube was created with the same coordinates as the vertex (see Figure 10.4 for an illustration). This simplifies the task of determining which vertices to assign UV coordinates to. The cubes were named according to the corresponding vertex's index.*
5. *The cubes belonging to the mesh face to be textured were manually identified by their indices.*
6. *A list of the UV coordinates* $(0, 0)$ *was initialized.*
7. *The list positions with the same indices as those identified in step 5 were assigned UV coordinates based on the coordinate scheme from Figure 2.8.*

10.4 Image Collection, Preparation, and Application

As part of this experiment, three different buildings were manually textured. This was done in order to uncover practical advantages and drawbacks with the texturing phase. The three buildings are shown in Figure 10.5, and their addresses and coordinate locations are shown in Table 10.1.

To collect textures, photos were taken from street level, either while standing in the road lane for less trafficked roads, or on the edge of the pavement for roads with heavier traffic. All photos were subsequently corrected using plane-to-plane homography. In order to simplify this step, a tool was developed that allows for rapid manual perspective correction. It works by displaying the non-transformed photos one by one on screen, along with a black-bordered box. By clicking and dragging on the image, the box corner closest to the mouse cursor will move along with the cursor. By right-clicking, Algorithm 10.1 will be executed, using

[1]The fact that the existence of a delta is necessary in order to access an attribute of a reference is considered a bug. Creating a delta is only a workaround for this issue.

[2]`https://norgeskart.no/`

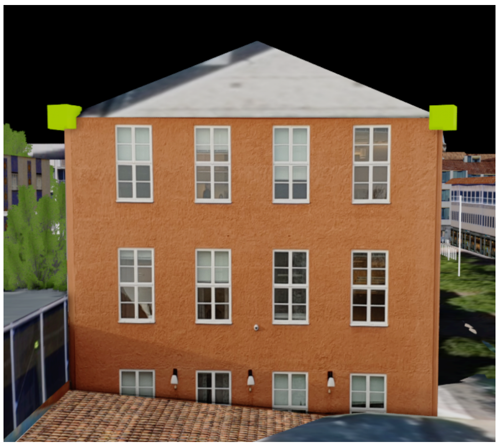


Figure 10.4: Illustration of green guiding cubes used to determine which vertices to assign UV coordinates to.

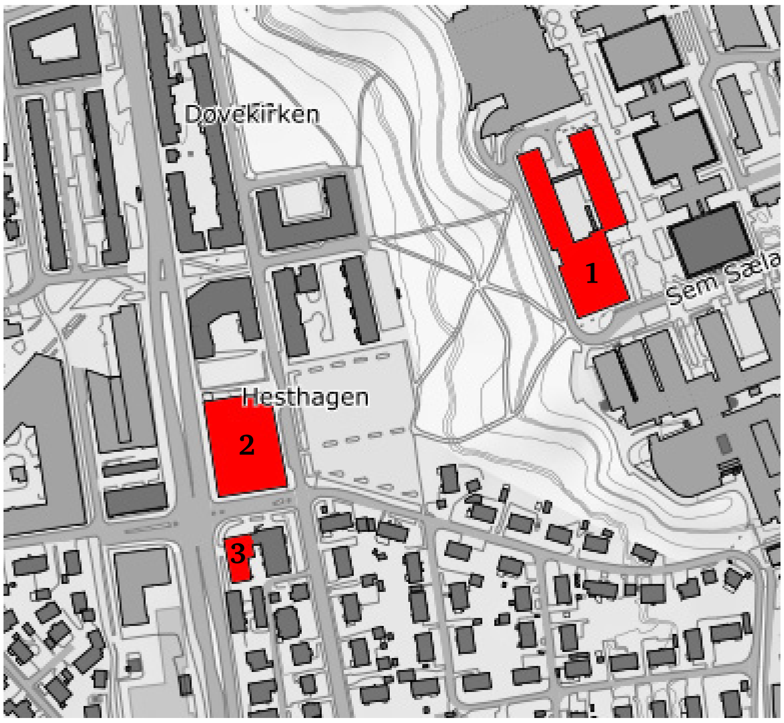


Figure 10.5: Map of the buildings that were manually textured. The numbers in the map correspond to the building numbers in Table 10.1. The map was fetched from Norgeskart.no.[2]

Building no.	**Street name**	**Street no(s).**	**Latitude (N)**	**Longitude (E)**
1	Sem Sælands vei	5, 7, 9	63°24′59.47″	10°24′11.41″
2	Klæbuveien	64, 66, 68, 70, 72	63°24′55.48″	10°23′55.80″
3	Holtermanns veg	2	63°24′53.05″	10°23′55.74″

Table 10.1: Table showing the addresses and coordinates of the manually textured buildings. The building numbers correspond to the numbers shown in the map in Figure 10.5. The information was fetched from Norgeskart.no. Latitudes and longitudes are given in the WGS84 coordinate system.

the corner coordinates as inputs in the matrix equations from Equation 10.3. The output of the script is a cropped image whose borders correspond to the borders of the black box. The algorithm can be executed as many times as deemed necessary per image. When the quality of the output is satisfactory, the image can be closed, and the next one will appear on screen. This repeats until all images have been consumed by the algorithm.

The walls comprising multiple images were then stitched together in Adobe Photoshop. The sub-images were manually color-corrected to unify their exposure and saturation. This reduces the occurrence of highly visible sub-image borders across the texture. Figure 13.2 shows a stitched texture before and after exposure and saturation correction.

Blender was used in order to quickly assign UV coordinates to each building wall. All finished textures were imported to Blender and applied to the corresponding mesh face. After the entire building was textured, the mesh was exported to OBJ, and a reference was created in Omniverse Create to that OBJ file. Finally, the transformation matrix from the Clip&Ship export's `global.fwt` file was parsed and applied to the building.

Chapter 11

Photogrammetric Wall Texturing

This chapter revolves around Research Question RQ4 *(Can we use photos taken from a driving car to update building textures automatically in real-time?)*. An alternative method for applying textures to 3D buildings is introduced. For this purpose, a method for wall texturing using photogrammetry was developed. This method regards the buildings objects from exports such as Clip&Ship as 'ground truth' objects for where the building should be located. An external photogrammetry software then creates a textured 3D representation using a video of a geographical area taken from the perspective of a driving car. This mesh is georeferenced into the Omniverse scene, and the appropriate parts of that mesh's geometry is projected onto the 'ground truth' object. This results in a flat, textured mesh laying on top of the already existing ground truth mesh.

11.1 Mesh Construction and Georeferencing

Video frames recorded from the front-facing camera of NAPLab's car was used as input to the photogrammetry software. First, every 10$^{\text{th}}$ frame was extracted from the video, reducing the amount of images to process from 30 per second to 3 per second. Then, images were gradually imported into the photogrammetry software Meshroom, giving a reconstructed point cloud with relative camera positions. This point cloud was converted to a textured mesh, resulting in a textured OBJ file.

Along with the recorded video, NAPLab's car outputs a file with timestamps for each frame of the video. The GNSS receiver also records timestamps for each of its measurements. The video frames' timestamps were first paired with each camera position in the reconstructed scene. The camera positions are given in an arbitrary local coordinate system. Then, the timestamps are matched up with the timestamps of the GNSS recordings in order to give a real-life location to the camera positions. Timestamps from the camera and the GNSS device do not perfectly match, so a linear interpolation is performed on the GNSS coordinates.

We can use an affine transformation to georeference the constructed mesh. We find the affine transformation matrix by considering three camera positions and solving the following equation for **T**:

$$u = \mathbf{T}p \tag{11.1}$$

where u is the UTM33N coordinates of the chosen cameras, p is their local coordinates, and $\mathbf{T}$ is the transformation matrix that transforms the coordinate values from local to UTM33N coordinates. Coordinate values should ideally have as wide a range as possible in order to minimize precision errors. This method assumes that the first, last, and middle frame fulfill this requirement sufficiently. The resulting matrix was then applied to the imported mesh in Omniverse Create. The Y dimension was found to be excessively scaled; thus, the Y component of the scaling was replaced by the average of the X and the Z components. This was done with the assumption that the mesh should roughly keep its aspect ratio after georeferencing. The result is a georeferenced three-dimensional reconstruction of a portion of the real world.

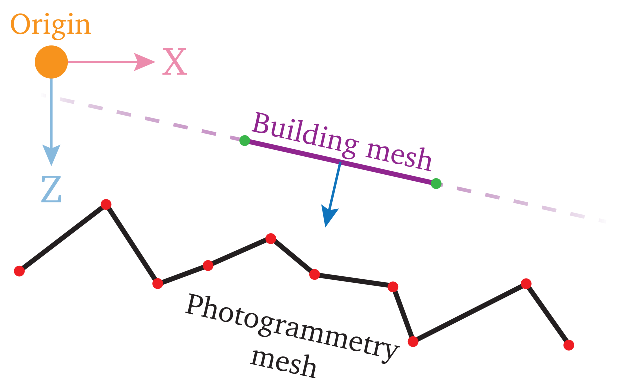
11.2 Projection to Ground Truth Geometry

The following section will refer to two different meshes. One will representing the 'ground truth' of a building's geometry and will be called the **building mesh**. The other will be a reconstruction resulting from photogrammetry and will be named the **photogrammetry mesh**.

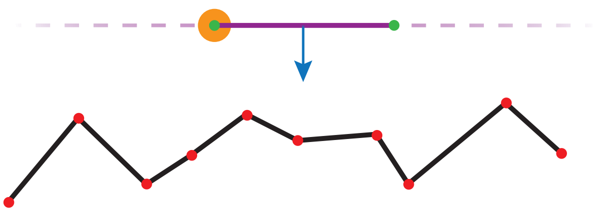
The following sequence outlines the procedure of aligning the geometry of the photogrammetry mesh to the building mesh (see Figure 11.1 for an illustration):

Sequence 11.1 - Alignment of Photogrammetry Mesh to Building Mesh

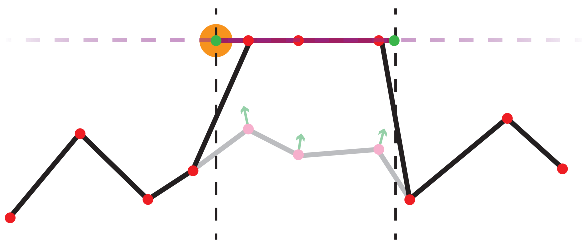
1. *The building mesh was translated so that one of the vertices of the face was positioned at the scene's origin.*
2. *The building mesh was rotated so that the face's normal was oriented parallel to one of the coordinate axes. In this case, the Z axis was arbitrarily chosen.*
3. *The same translation and rotation was applied to the photogrammetry mesh.*
4. *A filter was applied to determine which vertices of the photogrammetry mesh to transform. More specifically, those vertices having X and Y coordinates inside the current face were marked for transformation.*
5. *If more than half of the filtered vertices had their vertex normals pointing in the general direction of the corresponding face of the building mesh, then all of those vertices were marked for transformation. If not, then none of them were.*
6. *The marked vertices were scaled by a factor of 0 along the Z axis.*
7. *The transformations performed in steps 1–3 were reverted.*
8. *All vertices unaffected by the scaling in step 6 were removed from the photogrammetry mesh.*



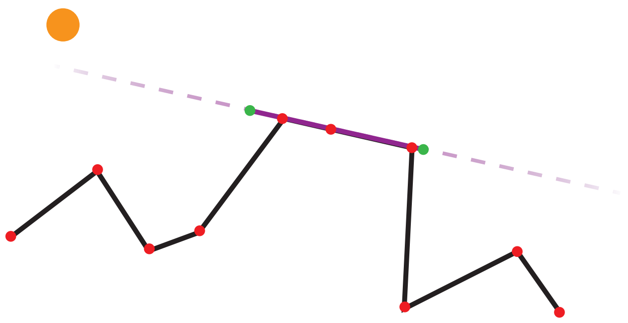
(a) Example start state.



(b) Translation to origin, and rotation to align face normal to Z axis (steps 1–3).



(c) Filtering of vertices and projection to building mesh (steps 4–6).



(d) Final result, after un-rotation and un-translation (step 7).

Figure 11.1: Illustration showing the core concepts of aligning the photogrammetry mesh to the building mesh. See Sequence 11.1 for reference.

11.2.1 Translation to Origin and Rotation to Z Axis

The transformation matrix in step 1 is the following translation matrix:

$$\mathbf{M_t} = \begin{bmatrix} 1 & 0 & 0 & v_1 \\ 0 & 1 & 0 & v_2 \\ 0 & 0 & 1 & v_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.2}$$

where $[v_1, v_2, v_3]^\mathsf{T}$ is the vertex coordinates of one of the faces in the building mesh. The transformation matrix used in step 2 is a rotation matrix consisting of two rotations. The first rotation, along the X axis, aligns the face's normal vector to the YZ plane. The other rotation, along the Y axis, aligns the normal to the Z axis. These transformation can be represented as such:

$$\begin{aligned} \mathbf{M_r} &= \mathbf{R_y}(\theta_2) \cdot \mathbf{R_x}(\theta_1) \\ &= \begin{bmatrix} \cos\theta_2 & 0 & \sin\theta_2 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_2 & 0 & \cos\theta_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_1 & -\sin\theta_1 & 0 \\ 0 & \sin\theta_1 & \cos\theta_1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{11.3}$$

where θ_1 is the amount of rotation to be performed along the X axis, and θ_2 is the subsequent rotation amount along the Y axis. $\sin\theta_1$ and $\cos\theta_1$ can be found by considering the face's homogeneous normal vector

$$\overrightarrow{v} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^\mathsf{T} \tag{11.4}$$

and projecting that vector onto the YZ plane:

$$\overrightarrow{v}' = \begin{bmatrix} 0 & y & z & 1 \end{bmatrix}^\mathsf{T} \tag{11.5}$$

Then, consider a triangle spanned between the Z axis, $\overrightarrow{v}'$, and a line parallel to the Y axis going through the endpoint of $\overrightarrow{v}'$. We can use trigonometry to rewrite $\sin\theta_1$ and $\cos\theta_1$:

$$\sin\theta_1 = \frac{y}{\sqrt{y^2+z^2}} \qquad \cos\theta_1 = \frac{z}{\sqrt{y^2+z^2}} \tag{11.6}$$

The intermediate vector after the first rotation can now be expressed as

$$\overrightarrow{v_1} = \mathbf{R_x}(\theta_1) \cdot \overrightarrow{v} = \begin{bmatrix} x \\ 0 \\ \sqrt{y^2+z^2} \\ 1 \end{bmatrix} \tag{11.7}$$

For the second rotation, we can once again, we can consider a triangle spanned between the Z axis, $\overrightarrow{v_1}$, and a line parallel to the Y axis going through the endpoint of $\overrightarrow{v_1}$. Using trigonometry, we can rewrite $\sin\theta_2$ and $\cos\theta_2$:

$$\sin\theta_2 = \frac{x}{\sqrt{x^2+y^2+z^2}} \qquad \cos\theta_2 = \frac{\sqrt{y^2+z^2}}{\sqrt{x^2+y^2+z^2}} \tag{11.8}$$

For simplicity, we use

$$\alpha = \sqrt{x^2+y^2+z^2} \qquad \beta = \sqrt{y^2+z^2} \tag{11.9}$$

The sines and cosines can then be written as

$$\begin{aligned} \sin\theta_1 &= \frac{y}{\beta} & \cos\theta_1 &= \frac{z}{\beta} \\ \sin\theta_2 &= \frac{x}{\alpha} & \cos\theta_2 &= \frac{\beta}{\alpha} \end{aligned} \tag{11.10}$$

The final rotation matrix then becomes

$$\begin{aligned} \mathbf{M_r} &= \begin{bmatrix} \frac{\beta}{\alpha} & 0 & \frac{x}{\alpha} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x}{\alpha} & 0 & \frac{\beta}{\alpha} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{z}{\beta} & -\frac{y}{\beta} & 0 \\ 0 & \frac{y}{\beta} & \frac{z}{\beta} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \frac{\beta}{\alpha} & -\frac{xy}{\alpha\beta} & -\frac{xz}{\alpha\beta} & 0 \\ 0 & \frac{z}{\beta} & -\frac{y}{\beta} & 0 \\ \frac{x}{\alpha} & \frac{y}{\alpha} & \frac{z}{\alpha} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{11.11}$$

This gives us a transformation matrix that can be explicitly and directly constructed from the face's normal coordinates alone.

There are two edge cases that must be handled. One occurs when $\alpha = 0$, i.e. when $x = y = z = 0$. This happens when the face's normal vector has no length or rotation. This is considered an error, and is handled by returning the identity matrix. The other edge case happens when $\beta = 0$, i.e. when $y = z = 0$ and $x \neq 0$. In this case, the face's normal is parallel with the X axis. In order to rotate the vector to the Z axis, one only needs one rotation 90° along the Y axis. The final rotation matrix then becomes

$$\mathbf{M_r} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.12}$$

11.2.2 Vertex Filtering

The filtering is performed in two steps. First, an axis-aligned bounding box is created around the face in question. Because a face in and of itself has no volume, its axis-aligned bounding box also has no volume. Therefore, the Z component of the bounding box was extended by a fixed amount. All vertices located outside of this bounding box remained unmarked for processing.

Then, a more fine-tuned filtering was performed. Because the face has been aligned to be parallel to the Z axis, only the X and Y coordinates of the photogrammetry mesh's vertices were sufficient for filtering. The face was treated as a 2D polygon, and a point-in-polygon test was performed for every vertex in the photogrammetry mesh, ignoring all Z values.

Subsequently, a decision was made as to whether any transformation of the photogrammetry mesh's vertices would be performed at all. All vertex normals belonging to the filtered vertices were compared to the face normal of the current face in the building mesh. If more than half of the vertex normals were at a less than 90° angle compared to the face normal, then *all* of the vertices would be transformed. Otherwise, *none* of the vertices would be transformed. The angle between the vectors were computed by simple dot products, where a dot product less than 0 represents an angle of less than 90°.

11.2.3 Zero-Scaling

After filtering, the remaining vertices were scaled with a factor of 0 along the Z axis:

$$\mathbf{T_s} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11.13}$$

Because one of the building mesh's faces are parallel to the world's Z axis and touching the world's origin, this transformation will in practice project the geometry of the photogrammetry mesh to that of the building mesh. Then, both meshes were transformed back to their original location and rotation. Finally, all vertices of the photogrammetry mesh not affected by the scaling were removed.

Chapter 12

Virtual Reality

This chapter will explore Research Question RQ5 *(Can a digital twin be visualized in VR alongside real-time scene updates?)*. Basic VR functionality was implemented for the USD scene. Omniverse contains an application called Omniverse XR for this purpose. Omniverse XR supports head mount and controller tracking natively, as well as movement and teleportation. As a proof of concept, the scene was rendered using both SteamVR and CloudXR.

The SteamVR solution was run locally with a wired connection to the HMD. In Omniverse XR, VR mode features both HMD and controller tracking, as well as locomotion and teleportation. Controller models are visible in-world, along with labels informing the user of the available control bindings. The most important bindings are shown in Table 12.1 (see also Appendix B for a reference to the physical input controls).

As for the CloudXR solution, Omniverse XR only supports running CloudXR in Tablet AR mode. The rendered frames received by the client are meant for smartphones and tablet devices, having no locomotion, teleportation, or controller tracking. HMD tracking is supported, although with a limited field of view.

To set up CloudXR functionality, two separate computers were used. The computer running the Omniverse XR application was designated to be the server. The application runs the CloudXR server automatically, and informs which network

	Control binding	Effect
R ▲	Right stick, up	Teleport
R ◀▶	Right stick, left/right	Rotate left/right (30°)
L ⬍	Left stick, up/down (oriented horizontally)	Move forward/backward
L ⬍	Left stick, up/down (pointing vertically)	Move up/down
L ◀▶	Left stick, left/right	Pan left/right
L	Left trigger	Faster movement speed

Table 12.1: The most important control bindings available in VR mode in the Omniverse XR application.

ports to use when connecting to the server. The server's IP address was found by running the `ipconfig` command in a command line interface. The other computer was installed with the CloudXR client. Tablet AR was then started server-side in the XR application, and the client was connected using the server's IP address and TCP port.

Part III

Results, Analysis, & Summary

'If there is any truth in the world, it lies when I'm with you, and if I find the courage to speak my truth to you one day, remind me to light a candle in thanksgiving at every altar in Rome.'

— Elio Perlman
'Call Me By Your Name'
André Aciman

Chapter 13

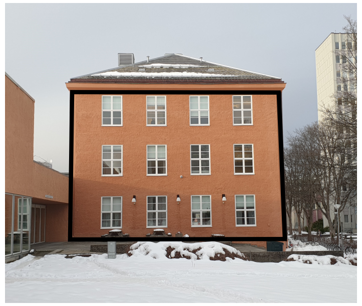
Results

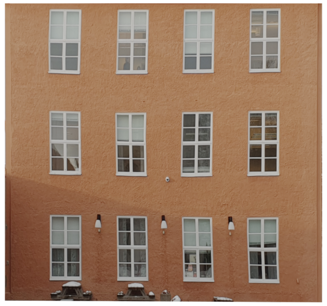
The results of the thesis are provided in this chapter. Note that the results are mainly visual and qualitative.

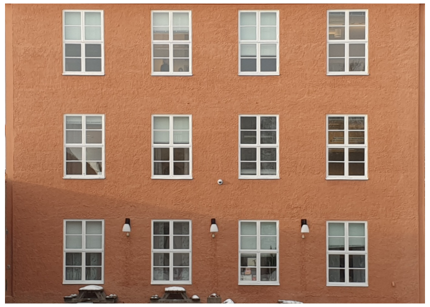
13.1 Projective Wall Texturing

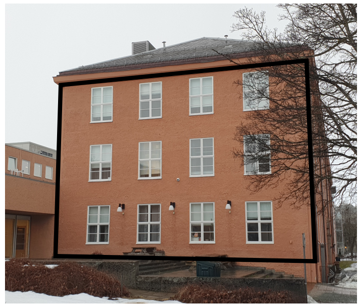
This section presents the results of the experiments performed in Chapter 10. Figure 13.1 shows results of applying a texture to a wall, before and after correcting for perspective distortion. Subfigures 13.1a, d, and g shows unprocessed images, with the wall of interest outlined in black. Subfigures 13.1b, e, and h shows the respective results after applying the unprocessed images on a square mesh face. Finally, subfigures 13.1c, f, and i shows the respective results after applying plane-to-plane homography to the unprocessed images.

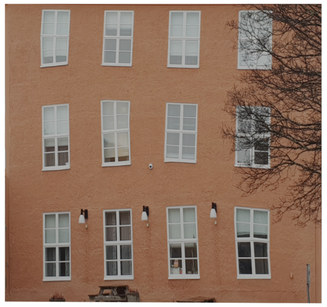
Figure 13.2 shows the result of performing exposure and saturation correction to a stitched texture. Furthermore, Figure 13.3 shows the manually textured buildings introduced in Figure 10.5 and Table 10.1. The buildings in the images have been automatically downloaded from Clip&Ship (see Chapter 8), and are shown with both roof textures and manually applied wall textures (see Chapters 9 and 10 respectively).

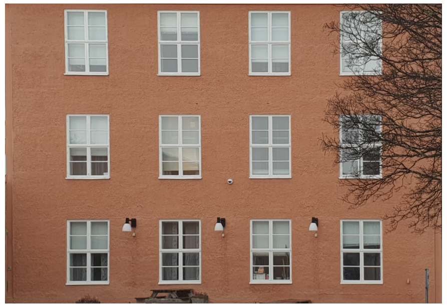

(a)

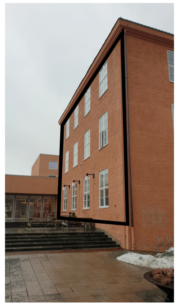

(b)

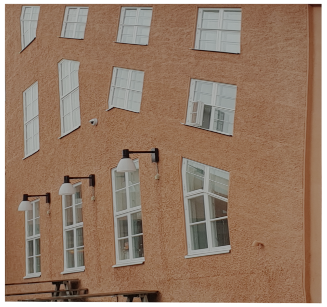

(c)


(d)


(e)


(f)


(g)


(h)

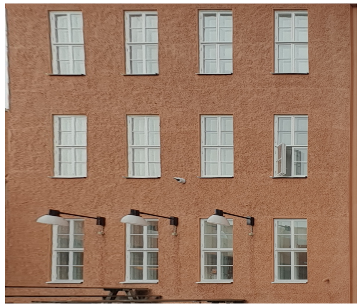
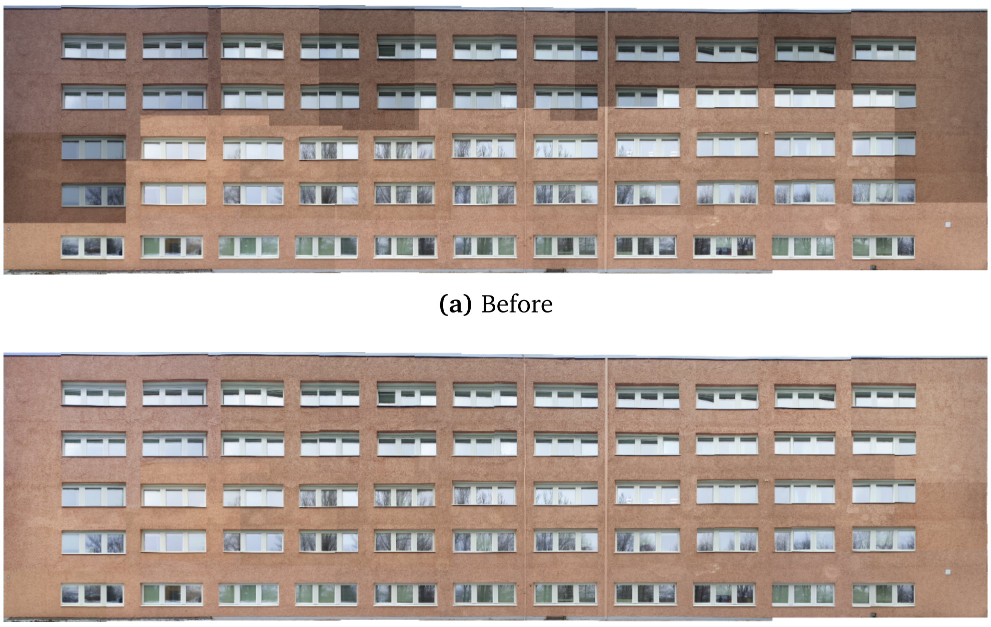

(i)

Figure 13.1: Results of applying a perspective distorted textures to a flat mesh face, along with subsequent perspective correction. Subfigures a, d, and g show the photos used for texturing, with the building face bordered in black. Subfigures b, e, and h show the respective results of applying the textures on the mesh face. Finally, Subfigures c, f, and i show the respective textures after performing plane-to-plane homography.



(a) Before

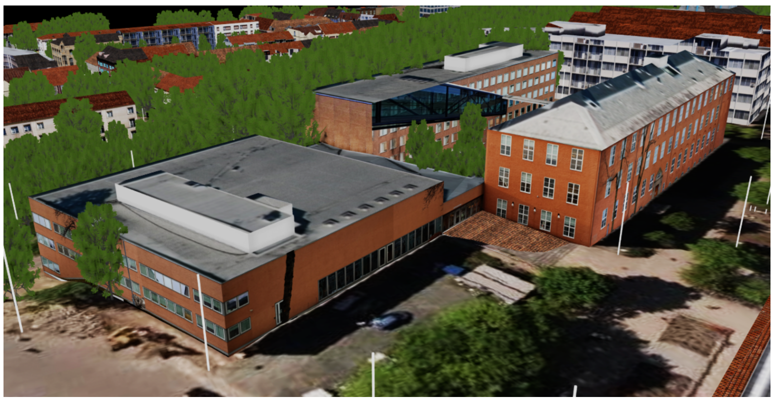
(b) After

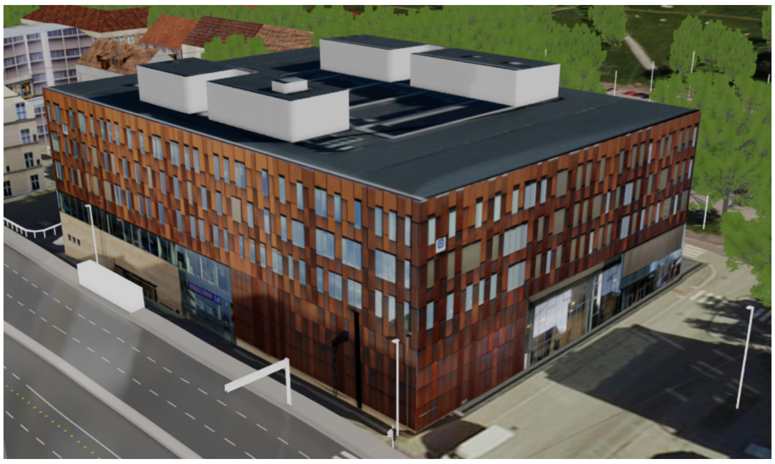
Figure 13.2: Before and after performing exposure and saturation correction on a stitched image textures.

13.2 Photogrammetric Wall Texturing

Figure 13.4 shows the results of projecting parts of a photogrammetry mesh onto a building mesh. The meshes in question represent the main building of Gløshaugen campus at NTNU, more specifically the building located at address Høgskoleringen 1. The figure shows both textured and untexture versions of the photogrammetry mesh, the latter for a clearer visualization of the geometry. Furthermore, Figure 13.5 shows the processed mesh with unprocessed vertices removed, again textured and untextured. Figure 13.6 shows the finished photogrammetry mesh georeferenced in Omniverse Create. Finally, Figure 13.7 compares closeups between the projective and the photogrammetric texturing methods.

Some timing measurements were also performed in order to given an answer to Research Question RQ4, specifically if we can use photogrammetry for real-time processing of textures. A video taken from the front-facing camera of NAPLab's car was used. Every 10th frame of the video was extracted and grouped into batches of 16 images each. Photogrammetry was performed on each of the batches separately. Figure 13.8 shows the running time for an arbitrary selection of batches, taken from batches that succeeded in outputting geometry.


(a) Building 1


(b) Building 2

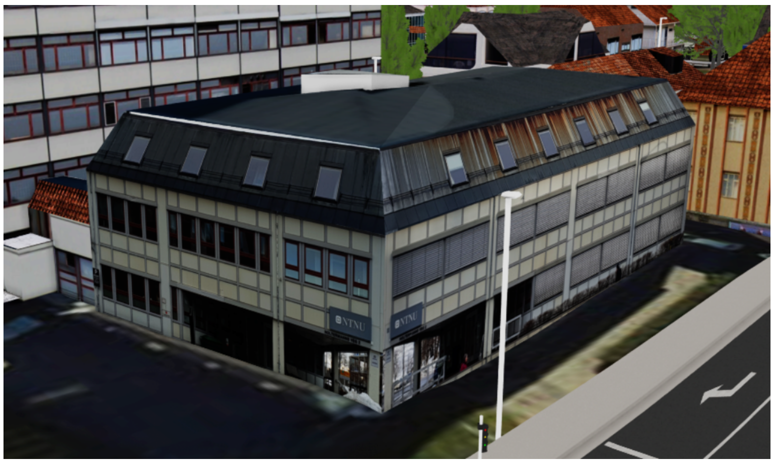
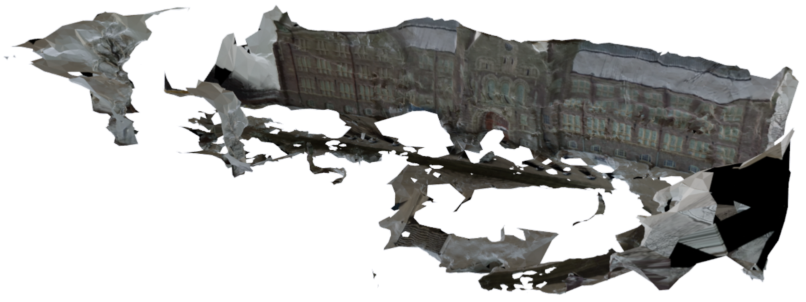
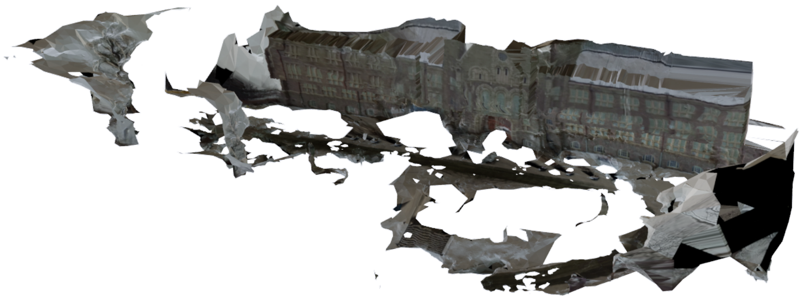

(c) Building 3

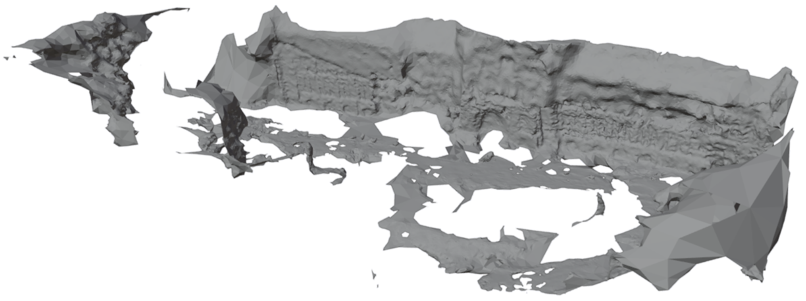
Figure 13.3: All three manually textured buildings shown in Omniverse Create (see Figure 10.5 and Table 10.1 for reference).



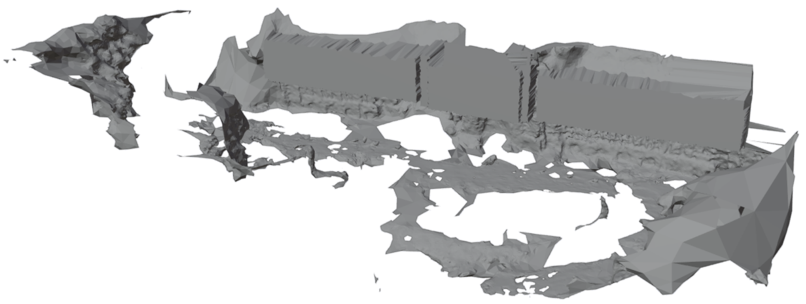
(a) Before, textured



(b) After, textured

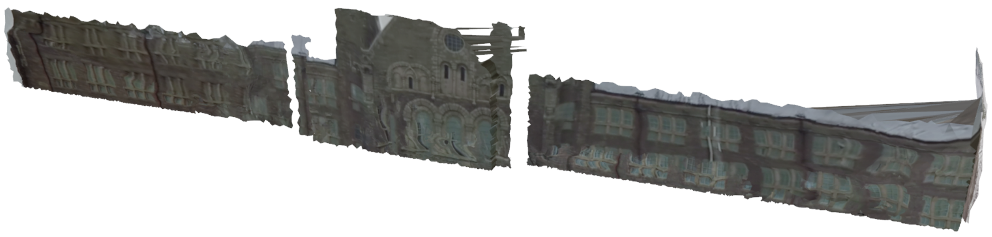


(c) Before, untextured



(d) After, untextured

Figure 13.4: The photogrammetry mesh before and after projecting to the building mesh, textured and untextured.


(a) Textured

(b) Untextured

Figure 13.5: The photogrammetry mesh after removing unprocessed vertices, textured and untextured.

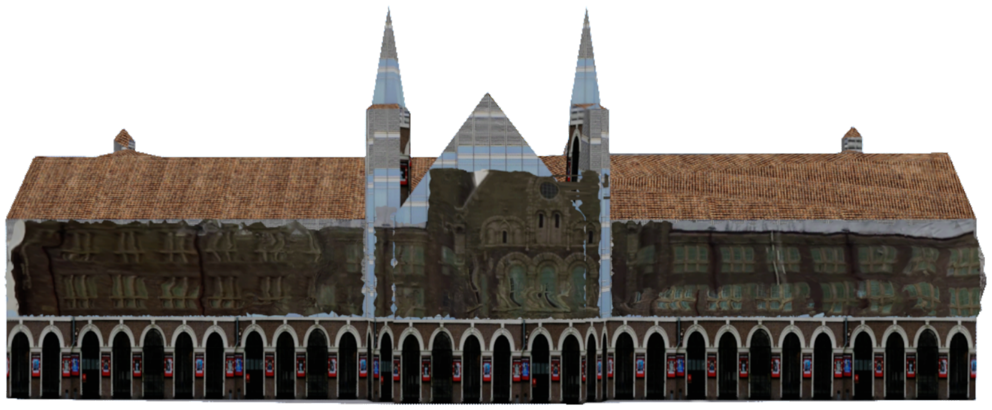
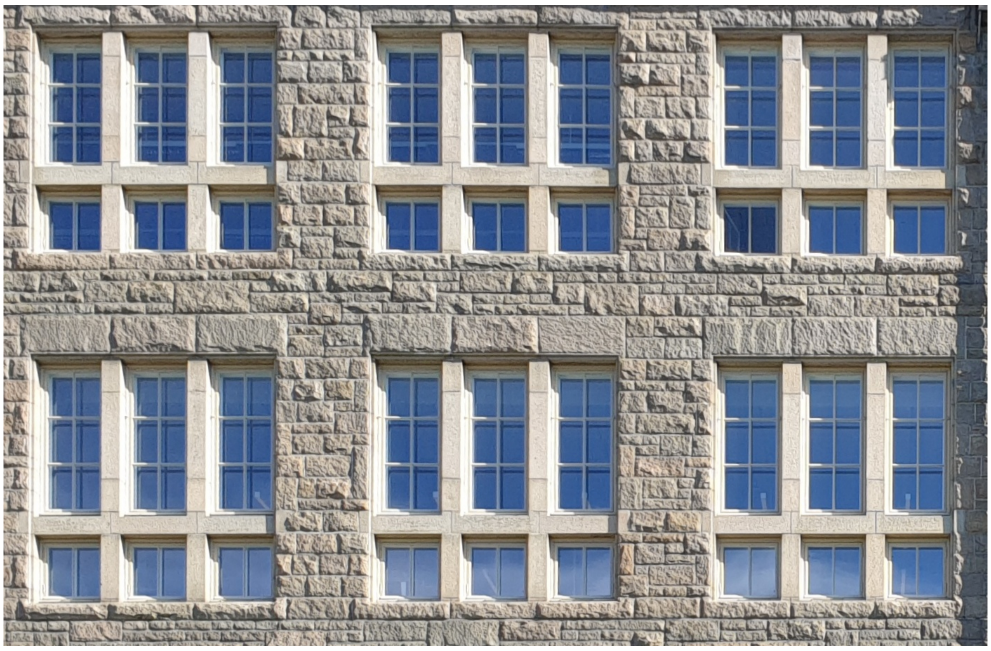
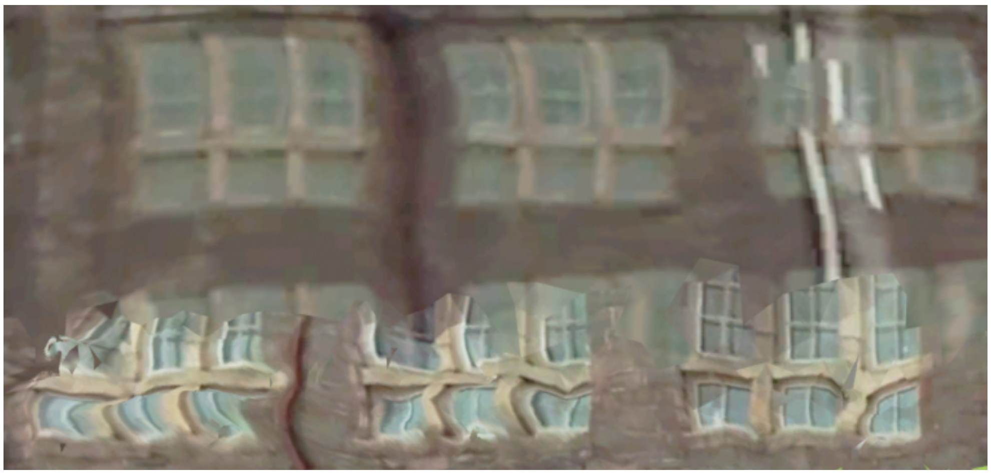


Figure 13.6: The processed photogrammetry mesh alongside the corresponding building mesh.



(a) Projective method



(b) Photogrammetric method

Figure 13.7: Comparison of closeup textures between the projective and the photogrammetric methods.

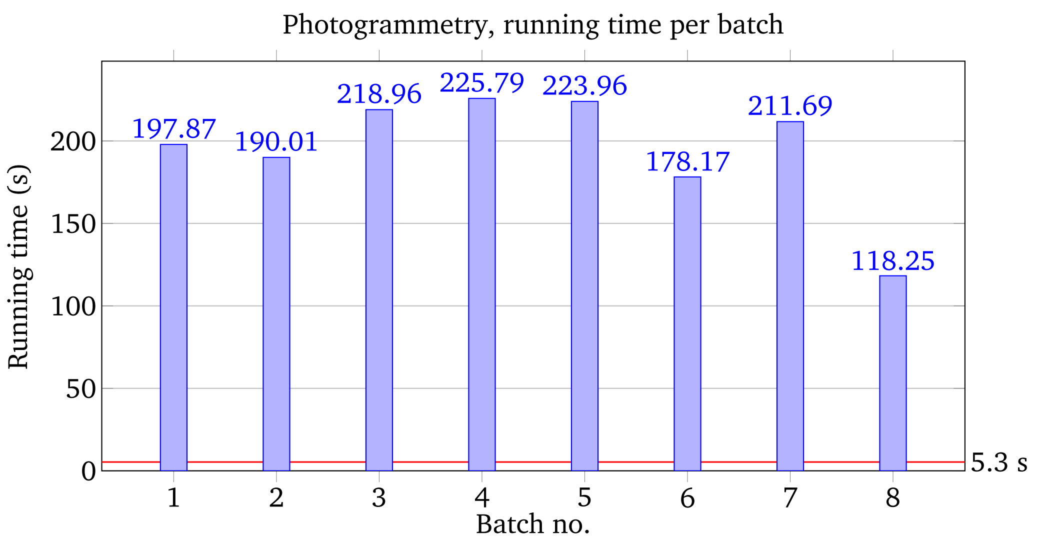


Figure 13.8: Graph showing the photogrammetry running time for an arbitrary selection of image batches. The red line shows the assumed upper limit for how long a batch should take to process, on average, in order for real-time processing to be possible (see Section 14.4 for more details).

13.3 Combined Results

A video displaying several aspects of the development was created. The video consists of an animation displaying two cars in Omniverse driving on the scene's RoadRunner road network. The cars' positions are updated in real-time with the connector developed in Chapter 7. Along their path, they pass two manually textured buildings from Chapter 10. The video can be found here: `https://www.youtube.com/watch?v=QouP4eRdQ78`

The same scenario was also recorded in VR using the Omniverse XR application. The video is evidence that real-time data can be received and the scene updated in real-time while simultaneously running in VR. It also illustrates the most common movement options available in the scene. The video is found here: `https://www.youtube.com/watch?v=8BxljpSVJo4`

Because of the presence of simulation metadata, the road network can be used for vehicle simulation. The network was imported into the CARLA vehicle simulator,[1] and a short video was created showing a simulation running. The video can be found here: `https://www.youtube.com/watch?v=FVghWQggE0g`

[1]`https://carla.org/`

Chapter 14

Discussion

This chapter will mainly present practical challenges encountered during development, as well as theoretical implications. At the end of the chapter, the research questions will be revisited and answered.

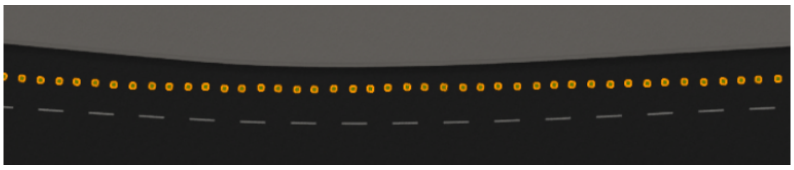
14.1 Position Data

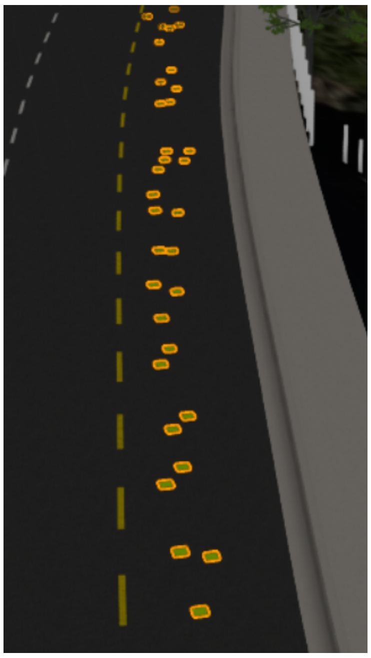
In the implementation of the position data connector introduced in Section 7, a number of precision issues in the data were uncovered. In areas where the line of sight between the car and navigation satellites is occluded, the GNSS readings are largely imprecise, causing jitter in the car model's travel path. Figure 14.1b shows examples of this (compare with Figure 14.1a for an example of accurate readings). Considering that the GNSS readings are used to orient the car as well, these imprecisions cause the vehicle model to rotate unpredictably.

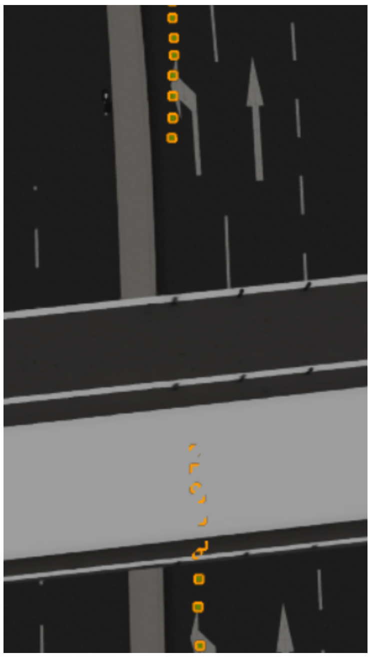
In some areas, the GNSS signals are missing completely, causing the GNSS readings to have a value of 0 in all spatial dimensions. In practice, this is encountered by visible gaps in the car's travel path, making the vehicle model virtually disappear from the scene. Examples of this are shown in Figure 14.1c.

In addition, errors in altitude readings were encountered, especially with the car staying still in the horizontal plane. The altitude readings slowly but steadily increased to the point of the car hovering several meters above ground. Examples are shown in Figure 14.1d. This issue can largely be avoided using a height map to set a vehicle model's altitude (see Chapter 7 for more details).

For the purpose of correct positioning relative to road lanes, GPS technology as used on NAPLab's car can be imprecise, at least if used in isolation. Ideally, readings should at least be precise to a few centimeters, while GPS technology can have an error of several meters [18]. This is especially true in dense, high-rise cities, where both aerial occlusion as well as reflective surfaces affect the quality of the signal [40]. This gap in accuracy can be closed using the Galileo GNSS, created by the European Space Agency. Galileo operates with a horizontal accuracy of less than 20 cm and a vertical accuracy of less than 40 cm [19]. Additionally, using 5G network signals or CPOS technology may further improve accuracy [20] [21].


(a) Example of an accurate GNSS signal. Readings are evenly spread, and changes in spread occur gradually when the vehicle drives normally without any sudden stops or accidents.


(b) Jitter in the GNSS readings caused by obstacles between the vehicle and GNSS satellites.


(c) Missing GNSS readings caused by obstacles between the vehicle and GNSS satellites.

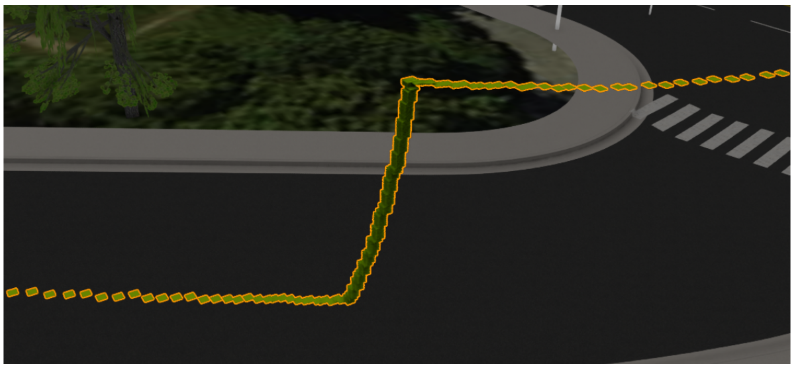

(d) Errors in altitude readings occurring while the vehicle stands still.

Figure 14.1: Examples from Omniverse showing inaccuracies in GNSS signals. Individual GNSS readings are indicated by green squares with orange borders.

14.2 Building Texture Quality

The topic of building texture quality requires special attention. The main reason for images and, by extension, building textures to be indistinguishable from reality is that they should be as recognizable as possible for neural networks. There exist several methods for validating the performance of an NN. However, comparing numerical evaluation methods was deemed out of scope for this thesis. Because neural networks are meant to mimic humans in their object recognition capabilities, the evaluation method used for texturing merely relies on qualitative and subjective object recognition capabilities to that of a human being. The following list of factors impacting the quality of a building texture is not exhaustive, but is rather meant to give a general sense of what aspects are important to assess the quality of building textures:

- **Resolution** — Resolution is deemed to be high enough if individual texels are indistinguishable from each other looking from any location on any road or pavement.
- **Proportions** — Features on the digital texture should be equally proportioned compared to the real-world wall—for example, equally sized windows should be equally sized in the texture.
- **Completeness** — The entire texture should be present on the corresponding wall, and all parts of the wall should be covered by the corresponding texture.
- **Parallel lines** — Lines that are parallel in real life should also be parallel in the texture. The opposite suggests that the texture has been erroneously warped in some way.

14.3 Projective Wall Texturing

Applying building textures is arguably the most difficult problem encountered during this thesis. Having accurately textured buildings is an integral part of a digital twin, especially one used for training neural networks with scene renders. It is important that the wall textures in the scene mimic the look of real-world walls accurately, and that no unnatural phenomena occur that can confuse the NNs and cause them to learn erroneous features.

The process of manual wall texturing proved to be a tedious and laborious one. However, it produces satisfactory results given that the camera used is of sufficient quality. In most cases, buildings will be distant enough from the car that the applied textures, and the lack of subtle geometric features like window sills and door knobs, are near indistinguishable from an actual building wall. However, the process unearthed many issues, many of which are addressed in the rest of this section.

14.3.1 Perspective Correction

Perspective correction is important in order to fulfill the requirement regarding parallel lines from Section 14.2. Based on the results in Figure 13.1, we observe that the size of the texture's diagonal artefact is proportional to the angle between the facing of the wall and the heading of the camera. This artefact happens because all texel values between the four vertices are linearly interpolated. This method of interpolation is called affine texture mapping. A common way to solve this by dividing the u and v coordinates with a depth value z [41]. However, the images taken are RGB images without depth information. One way to solve the problem is by supplying the RGB images with LiDAR readings; another is by using plane-to-plane homography as shown in Section 10.2.

However, usage of homography can bring its own set of problems. Figure 14.2 shows a corrected image with varying resolution. This can occur if the angle θ between the wall's normal vector $\vec{n}$ and the camera's orientation vector $\vec{c}$ is large (see Figure 14.3 for reference). The example used is the image from Figure 13.1i. The rightmost square shows a section of the wall having a satisfactory resolution. This section was located closest to the camera. The leftmost square shows a section with considerably lower resolution, located further away from the camera. This section of the wall does not fulfill the resolution requirement from Section 14.2. This result suggests that pixel density in the corrected image is inversely proportional to the distance between the camera and the corresponding point on the wall's surface. It also suggests that in order to achieve a uniform pixel density within the texture, all parts of the texture should be located at roughly the same distance from the camera. In other words, the camera should be oriented as parallel to the wall's normal vector as possible. Using side-facing vehicle cameras will mostly solve this problem, as they will be pointing approximately parallel to most walls located at the side of the road.

A high, uniform resolution is important in order to keep all parts of the wall recognizable for a neural network. If parts of a wall are severely impacted by resolution loss stemming from perspective correction, then the NN may wither fail to recognize the wall, or worse, label it incorrectly.

Objects protruding from the wall can also cause artefacts after applying homography. Figure 14.4 shows an example of lamps extending perpendicularly from a wall. Subfigure 14.4a constitutes a section from Figure 13.1c, and shows lamps extending from a wall in an image taken nearly perpendicularly to said wall. Subfigure 14.4b constitutes a section from Figure 13.1i, and shows the same lamps where the image has been taken at a large angle from the wall. In this case, the lamps are warped across large portions of the texture, giving the appearance of the lamps being rotated from its real-world perpendicular angle. This can be a source of confusion for neural networks, potentially causing false predictions.

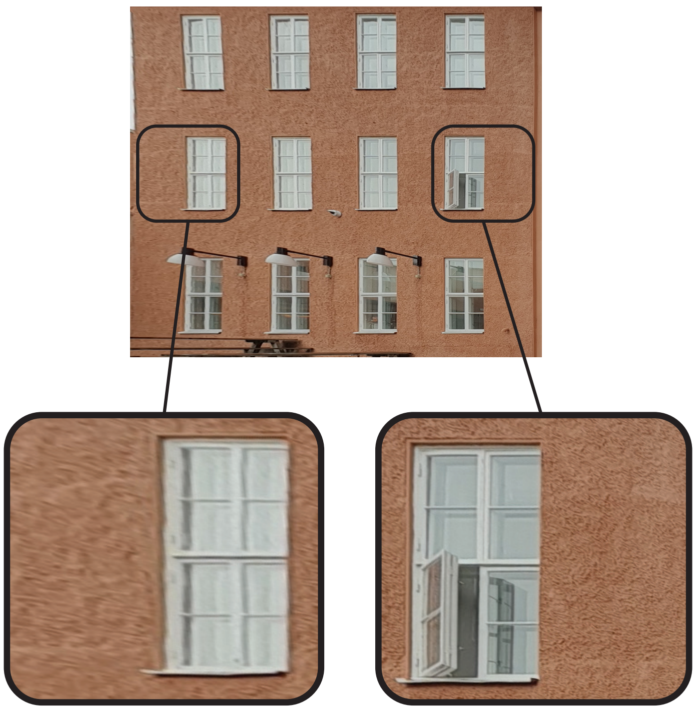


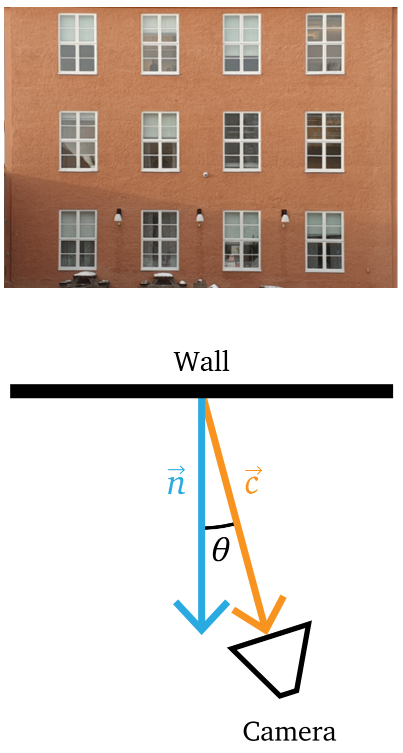
Figure 14.2: Illustration emphasizing the difference in resolution within an image that has been perspective corrected from an image taken at a sharp angle.

14.3.2 Obstructing Geometry

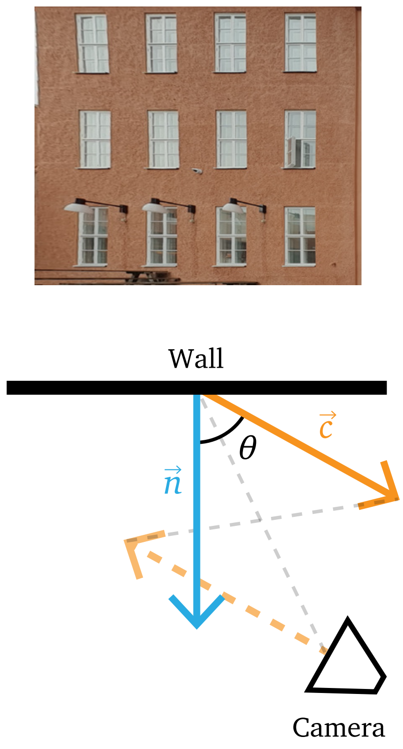
Often, the line of sight between the camera and the building walls can be obstructed. For example, Figure 14.5 shows a walking bridge obstructing the view of two different walls. This causes the bridge to become part of the texture that is applied to the building mesh. One possible way to remedy this is to sample the wall from different points of view. The effectiveness of this method depends on the distance between the obstacle and the wall—the larger the distance, the greater the effect of moving the camera. This has to do with the parallax effect, where faraway object will, if projected to a plane, move slower across the plane compared to closer objects. Figure 14.6 illustrates this phenomenon and shows how it solves the problem with obstructing geometry.

Obstacles can appear in many variants with varying degrees of problematicness. Stationary objects, like trees and other buildings, will obstruct walls over longer periods of time. Movable objects like cars, pedestrians, and smaller seasonal vegetation can pose less of a problem. Artefacts caused by these obstacles can be remedied simply by making another pass when they are absent.

From an NN standpoint, minimizing occlusion is particularly important. If there are many objects present within categories that a typical traffic NN is trained on (like people and cars), this can cause an autonomous vehicle to misunderstand the traffic situation, causing it to behave unpredictably.

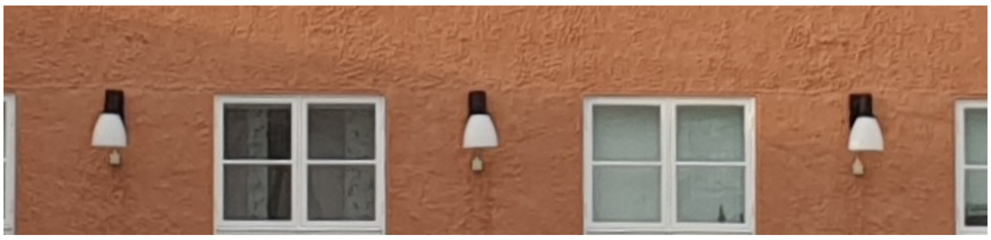


(a) Small θ, small distortions.

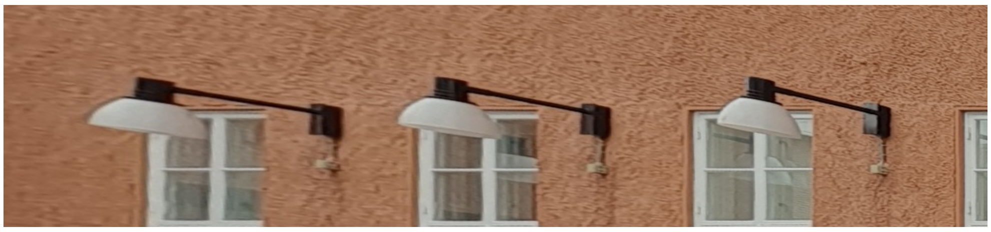


(b) Large θ, large distortions.

Figure 14.3: Illustration of how the angle θ between a wall's normal vector $\vec{n}$ and the camera's orientation vector $\vec{c}$ affects distortions arising from plane-to-plane homography.

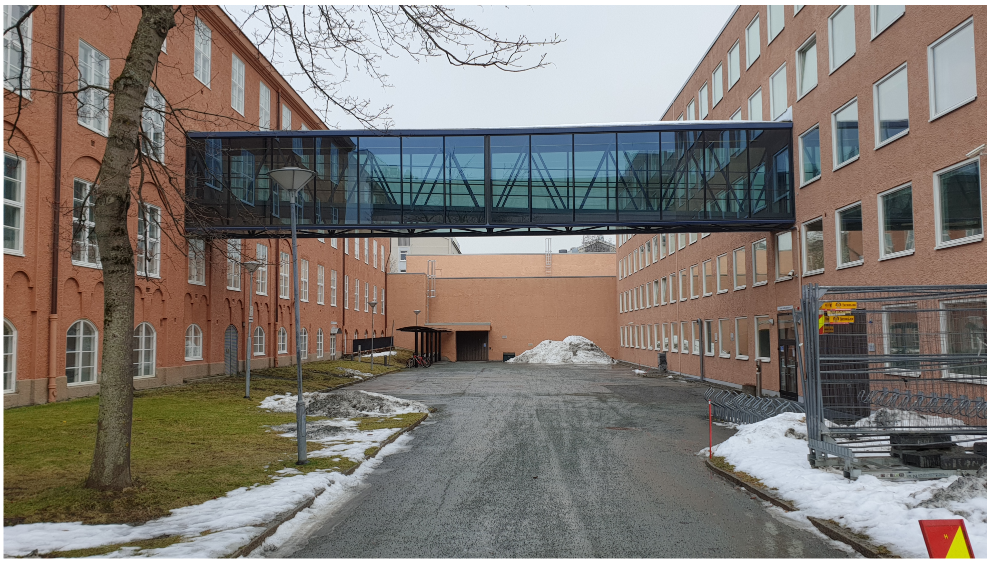


(a) Texture without protrusion distortion (taken from Figure 13.1c).



(b) Texture with protrusion distortion (taken from Figure 13.1i).

Figure 14.4: Illustration showing how objects protruding from walls can cause inaccuracies in photos taken from sharp angles from a wall.


(a) Untreated image with walking bridge.

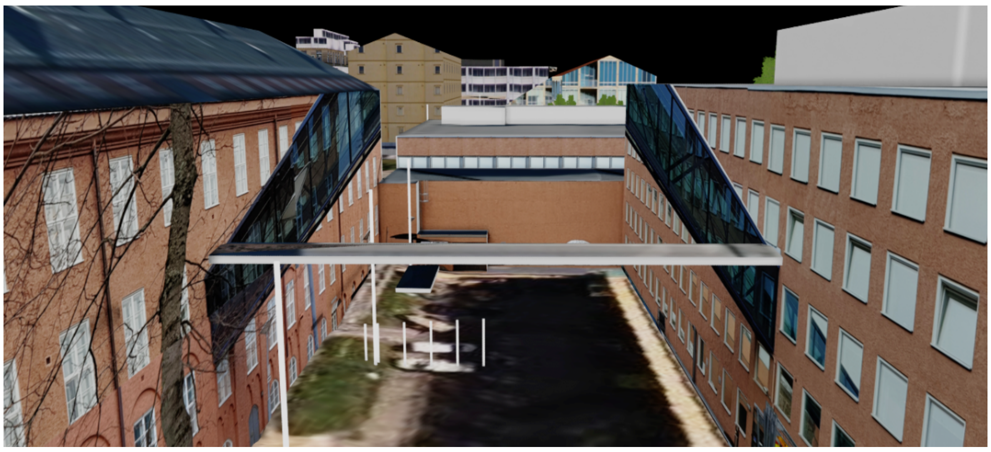

(b) Textures with obstructing walking bridge applied to building mesh.

Figure 14.5: Example of a walking bridge obstructing building walls, and the result of applying the corresponding textures to the building mesh.

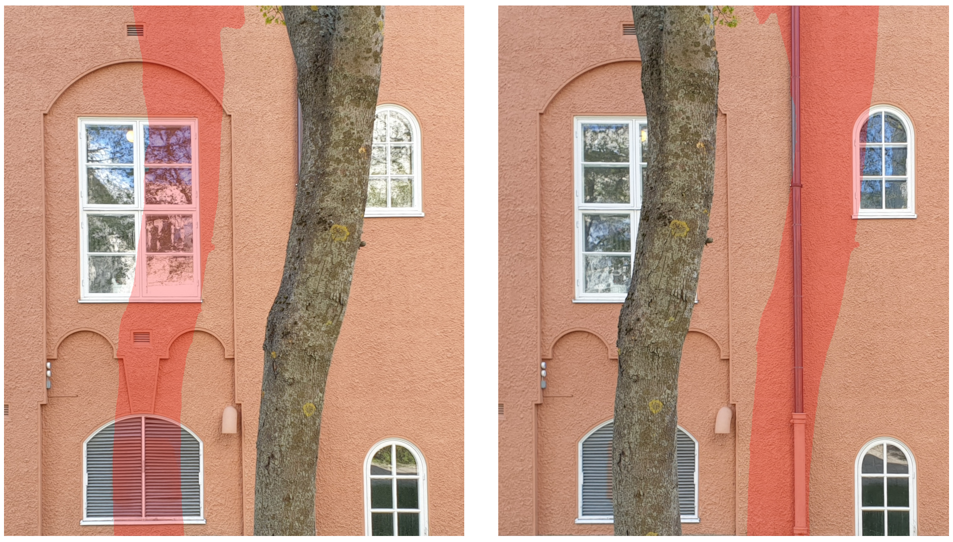

(a) (b)

Figure 14.6: Illustration of how the parallax effect can solve parts of the obstructing geometry problem. The red overlay in one subfigure represents areas revealed from the other subfigure by panning the camera.

14.3.3 Missing Geometry

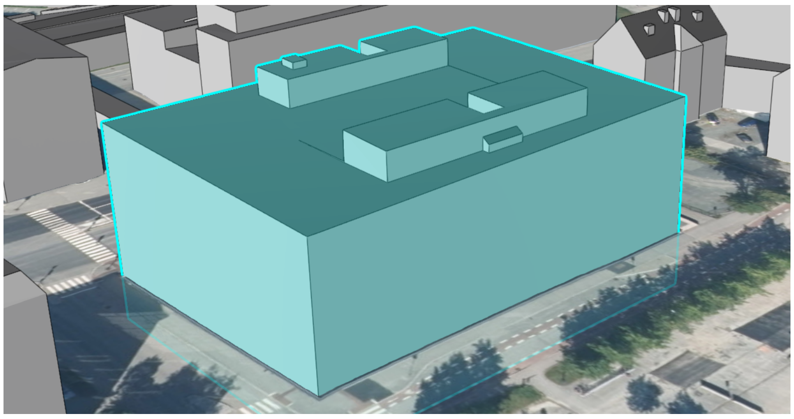
Sometimes, the available building meshes lack important geometric features. Figure 14.7 shows how textures for building corner—where the corner has an overhang not present in the 3D model—are applied to said 3D model. Not only does the resulting texture respond improperly to movement in terms of parallax; it also contains visual information that is not part of the building itself. For example, as shown in Figure 14.7d, both street signs, humans, vegetation, and other buildings are erroneously textured onto the 3D mesh.

14.3.4 Incremental Updates

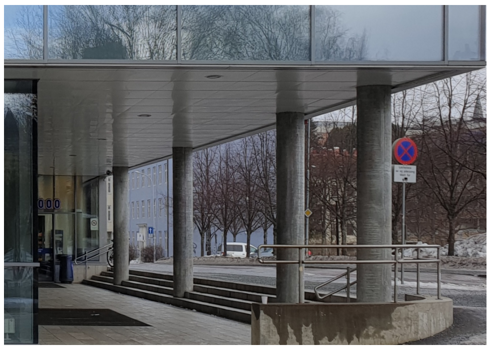
In the case of walls being too large to be textured from a single image, multiple images need to be combined. A solution is to use image stitching to expand the texture continuously as new video frames are taken. There are a number of existing software and algorithms that can be used for this purpose [42].

14.3.5 Incomplete Texture Coverage

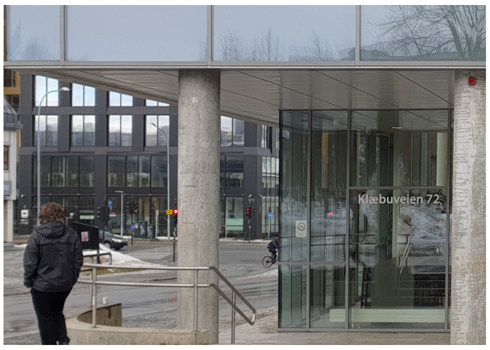
In many cases, buildings will be incompletely textured, both with manual and automatic approaches. Figure 14.8 shows an example of an untextured area using the manual approach in Chapter 10. Here, missing textures are shown as black areas. In automatic approaches, this phenomenon can frequently occur if only side-facing vehicle cameras are used. Such cameras will most often record



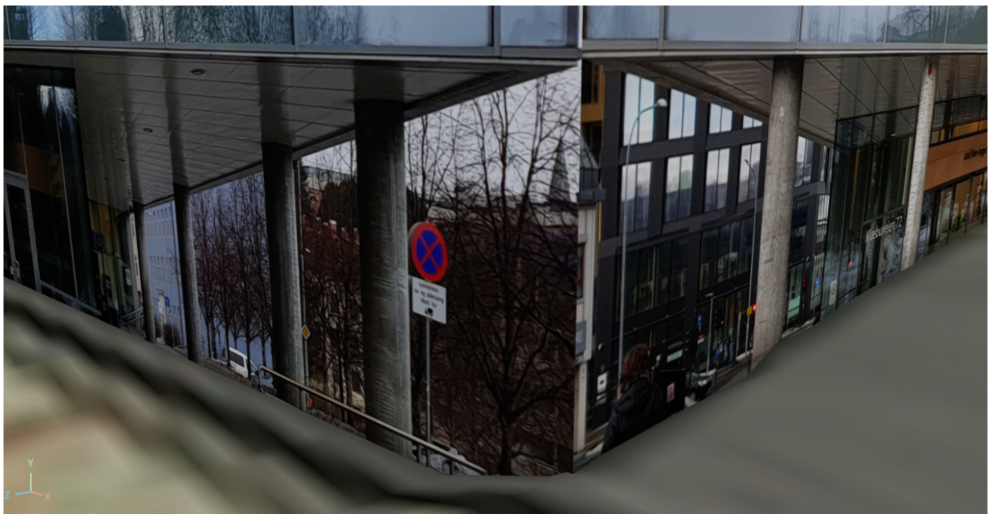
(a) Building mesh. The corner closest to the viewer is missing an overhang and pillars.



(b) Corner, looking north



(c) Corner, looking west



(d) textures applied to a building with missing geometry.

Figure 14.7: Example of a building missing large amounts of geometry compared to its real-life counterpart and the acquired image textures. The building shown is Building 2 from Table 10.1.

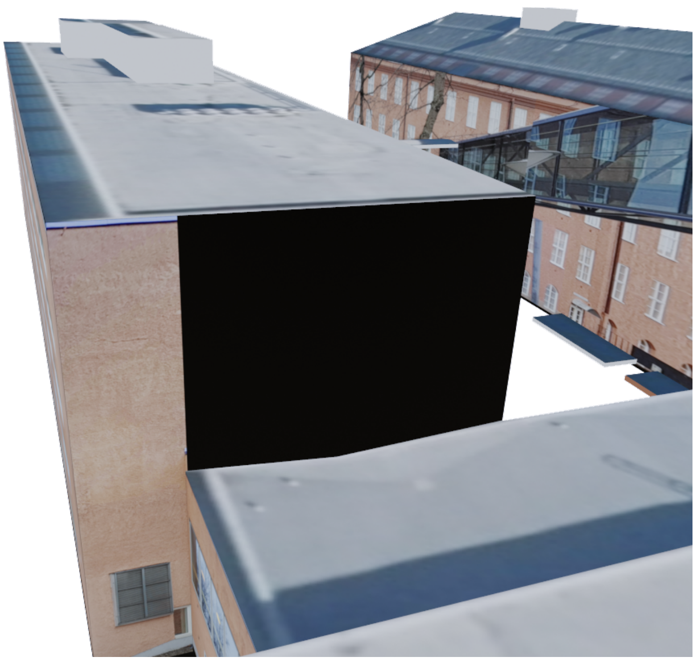


Figure 14.8: Example from the manual texturing process showing Gløshaugen's IT building (Building 1 from Table 10.1) with a missing part of its texture.

walls located close to the road. Only small sections of said walls will be recorded, meaning that large parts, especially on tall buildings, can be left untextured. One solution is using automatically generated textures, like those exported from CityEngine in Brekke's thesis [1].

14.3.6 Lens Distortion

Very often, because camera lenses are not able to capture its surroundings perfectly, there will be lens distortion present in the images. The most common way for lens distortion to manifest itself is by curving originally straight lines. This can cause curved texture segments to be applied to straight mesh edges, giving an erroneous texture mapping. This distortion can be reversed by applying mathematical distortion models to the distorted image. One of the most basic equations for distortion correction is the polynomial equation [43]

$$r_2 = r_1 \left(k_0 + k_1 r_1 + k_2 r_1^2 + \cdots + k_{n-1} r_1^{n-1} \right) \tag{14.1}$$

where r_1 is the original radius of a point along the image's radial direction, r_2 is the new radius after correction, and $k_0, \ldots, k_{n-1}$ is a series of n distortion coefficients.

14.3.7 Difficult Weather Conditions

Autonomous vehicles perform best in clear weather with bare roads. However, weather conditions can vary wildly, not only between seasons, but also on a day-to-day basis. Winters, for example, can be characterized by occasional heavy snowfall, reducing the visible range around the vehicle. This can lead to NNs not recognizing vehicles in front of it, which in turn can cause traffic accidents. This is also true for heavy rainfall during warmer conditions, which is predicted to become worse as the effects of climate change worsen. Also, snow-covered roads is a cause for road lanes being invisible to the cameras, meaning that vehicles may have difficulties navigating safely.

Special care must be taken in order to avoid dangerous scenarios. Neural networks should be trained with low-visibility images taken in difficult weather conditions. These images should be collected initially with a human driver controlling the vehicle. In addition, a digital twin with precise and accurate lane positionings can be used to guide the autonomous vehicles using GNSS signals, in case lane markings should be occluded.

14.3.8 Automation

The method and experiments performed in Chapter 10 were largely manual. In order to automate the process, there are a number of additional obstacles to overcome. Firstly, a method for determining which parts of an image belong to a wall needs to be designed. This can be done with an AI method called keypoint extraction to detect the corners of a wall. Technologies proposed by D. Jeong and Y. Kim, modified for use on street-level images, can be used [44] for this purpose. Once four keypoints are identified, plane-to-plane homography can be performed automatically, with special handling of images containing partial walls. Image stitching algorithms can optionally be used to compose a set of partial textures before performing keypoint extraction. Alternatively, GNSS coordinates—along with azimuth and accelerometer readings from the car—can be used for pose estimation. This, in turn, can be used to determine where the texture should be applied [25].

14.4 Photogrammetric Wall Texturing

As mentioned in Section 11, there were inaccuracies in the Y scaling component of the photogrammetry mesh after georeferencing. There are two hypotheses for why this occurs. The first stems from the fact that camera locations often lie very close to each other in the Y direction. A car moves predominantly in the horizontal plane, with only small increments of change in altitude. The resulting transformation may contain numerical imprecisions. The other hypothesis involves the altitude measurements received from the car's GNSS unit. These readings may themselves be inaccurate, considering that are measured with only two decimal digits. Both of these reasons may influence each other to create an inaccurate scaling.

Using a single monocular image sequence may also give too little data for the photogrammetry algorithms to produce accurate results. The output meshes and textures occasionally contain large amounts of distortion, as shown in Figure 13.7. This figure also illustrates the phenomenon where different parts of the texture have been sampled with differing distances between the camera and the building. The lower part of the texture has been sampled at closer range compared to the upper part, giving a non-uniform resolution within the texture. These result do not fulfill the requirements that textures should preserve parallel lines and proportions (see Section 14.2. If textures are sampled long-range, the method may also not fulfill the resolution requirement.

Photogrammetry is also prone to a range of different situations that commonly arise. For example, one single-colored wall may contain too few distinguishing features for the algorithms to compare with, especially if the image resolution is low. This may cause reconstruction to be inaccurate at best, or completely missing at worst. Inaccuracies can also arise where there are many similar or repeating features. Also, reflective surfaces can cause the algorithm to recognize areas of interest in the mirrored images, making it 'reconstruct' non-existing geometry.

Finally, the process may be too time-consuming for real-time processing. We assume that in order to achieve real-time processing, the time used for processing one batch of images is less than or equal the amount of time spanned by the frames in that batch. Using n for the total number of video frames per batch, f for the video framerate, and that every ith frame from the video is used, the total number of seconds s used to process the photos from each batch is

$$s = \frac{ni}{f} = \frac{16 \cdot 10}{30\,\mathrm{s}^{-1}} \approx 5.33\,\mathrm{s} \tag{14.2}$$

The condition we wish to fulfill is

$$s \leq p \tag{14.3}$$

where p is the average processing time per batch. As seen in Figure 13.8, all selected batches use a running time of almost two minutes or more, with an average running time of $p \approx 197$ seconds. This is considerably larger than the 5.3 seconds needed to achieve real-time processing.

14.5 Other Topics

14.5.1 Privacy

When it comes to digital twins in general, large amounts of data from a multitude of sources is often required. This can have negative implications concerning the privacy of individuals. This is especially true for digital twins in the medical domain, where patients' health conditions are subject to data collection [5].

In the case of a traffic-oriented digital twin, privacy is usually related to the possibility of identifying pedestrians and cyclists, and connecting them to a digital record of their location at any point in time. It is also related to vehicle license plates being identified, and similarly the possibility to locate these in space and time as well. One way of keeping the identities of individuals and vehicles hidden is to perform blurring on the relevant areas of the images recorded by cameras. This can be automated by utilizing NNs to segment the images, and subsequently blurring the segmented areas. Non-visual data should also be aggregated in order to decouple it from personal identities.

14.5.2 Upscaling

In order for a digital twin to be fully functional regarding predictions and accuracy, a vast amount of data is necessary. In order to give an accurate representation of traffic flow and congestion, one needs an accurate up-to-date record of the locations of vehicles, cyclists, and pedestrians. Connectors should be designed with special consideration of large data volumes—for example, they can be configured to collect updates from multiple sources in close temporal proximity, and update multiple in-scene objects in a single scene update. Connectors may also need to run on datacenter-type hard drives in order to handle the large amount of disk operations necessary to support large-scale multi-entity operations. Additionally, in order to run predictions when simulating changes in the road network, one needs an accurate and vast history of driving patterns, likely coupled with an artificial intelligence for learning said patterns.

For building textures to be accurate and high-resolution, a large amount of data needs to be collected. Both projective and photogrammetric wall texturing suffer from the fact that images from street-level actors only may not be sufficient. Ideally, camera-equipped Unmanned Aerial Vehicles can sweep the area with regular spatial and temporal intervals. The resulting images can then either be projected onto buildings using GNSS locations and pose estimation, or they can be used in a photogrammetry pipeline. Using projective texturing may be considerably faster, although building geometry needs to be updated in a separate process. Using photogrammetric texturing, although slower, will give updated building geometry in addition to textures.

14.6 Research Questions

RQ1: Can dynamic location data of a physical entity be transferred to a digital 3D scene in order to update it in real-time?

In Chapter 7, a script was created to update the Omniverse scene automatically with position data. Even though values were read from CSV files, the positions of the vehicles were updated in real-time as data was received.

RQ2: Can a three-dimensional model of a geographical area be acquired or created, especially with terrain, buildings, and roads; and to what degree can this be automated?

A road network model manually created in RoadRunner was used as part of this project's foundation. Some options exist for automating this (see Section 15.2). Using the Clip&Ship service from Geodata, 3D buildings and terrain models can be automatically acquired as shown in Chapter 8. In Chapter 9, a proof-of-concept code was developed for automatically applying roof textures to building meshes. Wall textures can be acquired manually using projective wall texturing (see Chapter 10), or automatically with photogrammetry (see Chapter 11). An automatic method using keypoint extraction was also briefly discussed in Section 14.3.8.

RQ3: Can any photo of a building be applied as a texture to a corresponding 3D object so that the building looks realistic enough to be near indistinguishable from reality?

In Chapter 10, we examined the viability of performing wall texturing manually. By taking photos with high enough resolution and performing perspective correction, most photos of a building wall can be applied to the corresponding 3D mesh. However, photos that are taken at a sharp angle compared to the facing direction of the wall might not be suitable for texturing.

RQ4: Can we use photos taken from a driving car to update building textures automatically in real-time?

In Chapter 11, a method was designed that uses photogrammetry and GNSS readings to automatically texture buildings. Section 13.2 shows that photos taken from a car can indeed be used for reconstructing a textured building mesh. However, as discussed in Section 14.4, the results often have undesirable distortions. Often, parallel lines become warped, proportions are not always preserved, and the resolution is often too low. The process may also be too time-consuming to be suitable for real-time processing.

RQ5: Can a digital twin be visualized in VR alongside real-time scene updates?

As shown in Chapter 12, the scene was successfully rendered using VR technology, while the position of multiple vehicle models were updated in real-time. For this thesis' simple setup, using SteamVR above CloudXR gave the more satisfying results and may be more suited for the purpose of digital twin visualization.

Chapter 15

Conclusion & Future Work

15.1 Conclusion

This thesis revolved around research and development into digital twins, and used an area around Gløshaugen in Trondheim, Norway as a case example. The main topics of interest were real-time scene updates, especially regarding the positioning of entities; wall and roof texturing; and virtual reality. The aspect of wall texturing was examined in particular depth. A manual method using hand-picked photographs, perspective correction, and careful stitching was conducted. An automatic method using photogrammetry was also explored.

Results show that real-time position updates, retrieval of building geometry, and roof texturing were relatively straightforward to accomplish. Conversely, wall texturing proved to be particularly difficult, and deserves further research and development. The projective method produces satisfactory results when performed manually, and acts as a solid foundation for exploring further possibilities regarding real-time automation. The photogrammetric method, although possible to automate without artificial intelligence, is prone to visual distortions, at least when input is limited to only one front-facing camera recording while driving. The process is also too slow to be used for real-time purposes.

VR was also briefly explored in this thesis. SteamVR provided the best navigation options using a VR headset. It was also proved that VR could be combined with a digital twin's real-time updates.

15.2 Future Work

One aspect in particular that deserves more thorough explorations is the simulation aspect. NVIDIA has developed a vehicle simulation software called DRIVE Sim.[1] Road networks developed with RoadRunner can be used alongside DRIVE Sim to provide a vehicle simulation environment within Omniverse, that in turn can be combined with the rest of the digital twin. The road network itself can

[1]`https://developer.nvidia.com/drive/drive-sim`

also be further developed, for example by settings speed limits and creating signs and traffic lights. Some automation can possibly be performed, for example by using data from OpenStreetMap[2] and Geonorge. Alternatively, aerial and street-level photography combined with AI and conventional algorithms could be used for detecting road features such as road dimensions, lane markings, signage, and traffic lights [45] [46] [47].

As of this thesis, the position data connector only reads prerecorded data points from a file. The first logical improvement would be for the connector to accept real-time data from a driving car. This could be achieved by configuring the vehicle to post its data to a web server, which in turn can do optional data processing and then forward the data to the connector. Another option could be to utilize the Robot Operating System (ROS)[3] that is already equipped on NAPLab's vehicle to collect and send the data.

Roof texturing can be easily improved by changing the method of segmenting the roof from the rest of the mesh. A roof can be interpreted as a set of faces that have an angle of less that 90° from the world's up axis. This means that roof texturing is only dependent on the geometry of the mesh, and not on the presence of pre-applied textures.

Photogrammetric texturing can also benefit from improvements. Although probably unsuitable for real-time processing, photogrammetry has the potential to create highly realistic and complete building models, especially when using camera-equipped UAVs [27] [48].

The VR implementation in this thesis, although functional, is minimal. It serves as a proof of concept, and can be greatly expanded. One example is implementing overlays that can display information like text and figures. Overlays could show the temperature at a specific location time. Alternatively, one could navigate a timeline showing both previously recorded and predicted temperatures in the near future. Warnings about traffic congestion or road work could appear where relevant. Construction workers could be informed about damages to buildings that have been identified using either manual inspection of collected 3D models or by trained artificial intelligences. Property developers could visualize new buildings, as well as move them and change their dimensions without performing any preliminary construction.

[2]`https://www.openstreetmap.org/`
[3]`https://www.ros.org/`

Bibliography

[1] R. S. Brekke, 'Creating models of the real world in universal scene description,' M.S. thesis, Norwegian University of Science and Technology, Jun. 2021.

[2] A. Knutsen, 'Developing a digital twin of a road network for use in vehicle simulation and VR visualization,' Norwegian University of Science and Technology, Tech. Rep., Dec. 2021.

[3] Merriam-Webster, *Digital Definition & Meaning*, `https://www.merriam-webster.com/dictionary/digital`, Accessed: 2022-02-24.

[4] Merriam-Webster, *Twin Definition & Meaning*, `https://www.merriam-webster.com/dictionary/twin`, Accessed: 2022-02-24.

[5] B. R. Berricelli, E. Casiraghi and D. Fogli, 'A Survey on Digital Twin: Definitions, Characteristics, Applications, and Design Implications,' *IEEE Access*, vol. 7, pp. 167 653–167 671, Nov. 2019. DOI: `10.1109/ACCESS.2019.2953499`.

[6] M. Singh, E. Fuenmayor, E. P. Hinchy, Y. Qiao, N. Murray and D. Devine, 'Digital Twin: Origin to Future,' *Applied System Innovation*, vol. 4, no. 36, May 2021. DOI: `10.3390/asi4020036`.

[7] W. Kritzinger, M. Karner, G. Traar, J. Henjes and W. Sihn, 'Digital Twin in manufacturing: A categorical literature review and classification,' *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018. DOI: `10.1016/j.ifacol.2018.08.474`.

[8] C. Cimino, E. Negri and L. Fumagalli, 'Review of digital twin applications in manufacturing,' *Computers in Industry*, vol. 113, Oct. 2019. DOI: `10.1016/j.compind.2019.103130`.

[9] S. Weyer, T. Meyer, M. Ohmer, D. Gorecky and D. Zühlke, 'Future Modeling and Simulation of CPS-based Factories: an Example from the Automotive Industry,' *IFAC-PapersOnLine*, vol. 49, no. 31, pp. 97–102, 2016. DOI: `10.1016/j.ifacol.2016.12.168`.

[10] F. Tao, H. Zhang, A. Liu and A. Y. C. Nee, 'Digital Twin in Industry: State-of-the-Art,' *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, Oct. 2018. DOI: `10.1109/TII.2018.2873186`.

[11] E. Negri, L. Fumagalli and M. Macchi, 'A Review of the Roles of Digital Twin in CPS-based Production Systems,' *Procedia Manufacturing*, vol. 11, pp. 939–948, Sep. 2017. DOI: 10.1016/j.promfg.2017.07.198.

[12] National Oceanic and Atmosphere Association (NOAA), *Is the Earth round?* https://oceanservice.noaa.gov/facts/earth-round.html, Accessed: 2022-05-02.

[13] DMA WGS 84 Development Committee, 'Department of Defence World Geodetic System 1984, Its Definition and Relationships with Local Geodetic Systems,' The Defense Mapping Agency, Systems Center (SG), 8613 Lee Highway, Fairfax, VA 22031-2138, Tech. Rep. AD-A280 358, Sep. 1991.

[14] European Commission, *About Galileo*, https://ec.europa.eu/defence-industry-space/eu-space-policy/galileo_en, Accessed: 2022-05-25.

[15] Roscosmos, *About GLONASS*, https://www.glonass-iac.ru/en/about_glonass/, Accessed: 2022-05-25.

[16] U.S. Space Force, *GPS Overview*, https://www.gps.gov/systems/gps/, Accessed: 2022-04-02.

[17] U.S. Federal Aviation Administration, *Satellite Navigation - GPS - How It Works*, https://www.faa.gov/about/office_org/headquarters_offices/ato/service_units/techops/navservices/gnss/gps/howitworks, Accessed: 2022-04-02.

[18] U.S. Space Force, *GPS Accuracy*, https://www.gps.gov/systems/gps/performance/accuracy/, Accessed: 2022-03-27.

[19] European Union Agency for the Space Programme, *Galileo High Accuracy Service (HAS) | European GNSS Service Centre*, https://www.gsc-europa.eu/galileo/services/galileo-high-accuracy-service-has, Accessed: 2022-05-05.

[20] L. Yin, Q. Ni and Z. Deng, 'A GNSS/5G Integrated Positioning Methodology in D2D Communication Networks,' *Journal on Selected Areas in Communications*, vol. 36, pp. 351–362, 2 Feb. 2018. DOI: 10.1109/JSAC.2018.2804223.

[21] Kartverket, *Guide to CPOS*, https://www.kartverket.no/en/on-land/posisjon/guide-to-cpos, Accessed: 2022-06-04.

[22] M. W. Grieves, 'Product lifecycle management: the new paradigm for enterprises,' *International Journal of Product Development*, vol. 2, no. 1-2, Apr. 2005. DOI: 10.1504/IJPD.2005.006669.

[23] NASA, *DRAFT Modeling, Simulation, Information Technology & Processing Roadmap*, https://www.nasa.gov/pdf/501321main_TA11-MSITP-DRAFT-Nov2010-A1.pdf, Nov. 2010.

[24] RWTH Aachen Campus, *Autonomous driving with 5G*, https://www.rwth-campus.com/en/news/interviews-en/5g-autonomous-driving/, Jul. 2021.

[25] D. Butalov, G. Häufel, J. Meidow, M. Pohl, P. Solbrig and P. Wernerus, 'Context-based automatic reconstruction and texturing of 3D urban terrain for quick-response tasks,' *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 93, pp. 157–170, Jul. 2014. DOI: 10.1016/j.isprsjprs.2014.02.016.

[26] Z. Kand, L. Zhang, S. Zlatanova and J. Li, 'An automatic mosaicking method for building facade texture mapping using a monocular close-range image sequence,' *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, pp. 282–293, 3 May 2010. DOI: 10.1016/j.isprsjprs.2009.11.003.

[27] M. Mohammadi, M. Rashidi, V. Mousavi, A. Karami, Y. Yu and B. Samali, 'Quality Evaluation of Digital Twins Generated Based on UAV Photogrammetry and TLS: Bridge Case Study,' *Remote Sensing*, vol. 13, no. 17, Sep. 2021. DOI: 10.3390/rs13173499.

[28] D. Nakath, M. She, Y. Song and K. Köser, 'An Optical Digital Twin for Underwater Photogrammetry,' *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 90, pp. 69–81, Mar. 2022. DOI: 10.1007/s41064-021-00190-9.

[29] F. Lafarge, X. Descombes, J. Zerubia and M. Pierrot-Deseilligny, 'An Automatic Building Reconstruction Method: A Structural Approach using High Resolution Satellite Images,' in *International Conference on Image Processing*, IEEE, Oct. 2006. DOI: 10.1109/ICIP.2006.312541.

[30] N. Haala and M. Kada, 'An update on automatic 3D building reconstruction,' *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 65, pp. 570–580, 6 Nov. 2010. DOI: 10.1016/j.isprsjprs.2010.09.006.

[31] C. Qiu, S. Zhou, Z. Liu, Q. Gao and J. Tan, 'Digital assembly technology based on augmented reality and digital twins: a review,' *Virtual Reality & Intelligent Hardware*, vol. 1, pp. 597–610, 6 Dec. 2019. DOI: 10.1016/j.vrih.2019.10.002.

[32] W. Wang, H. Guo, X. Li, S. Tang, Y. Li, L. Xie and Z. Lv, 'BIM Information Integration Based VR Modeling in Digital Twins in Industry 5.0,' *Journal of Industrial Information Integration*, vol. 28, Apr. 2022. DOI: 10.1016/j.jii.2022.100351.

[33] H. Laaki, Y. Miche and K. Tammi, 'Prototyping a Digital Twin for Real Time Remote Control Over Mobile Networks: Application of Remote Surgery,' vol. 7, IEEE, Feb. 2019, pp. 20 325–20 336. DOI: 10.1109/ACCESS.2019.2897018.

[34] Z. Lv, J. Guo, A. K. Singh and H. Lv, 'Digital Twins Based VR Simulation for Accident Prevention of Intelligent Vehicle,' in *Transactions on Vehicular Technology*, vol. 71, IEEE, Apr. 2022, pp. 3414–3428. DOI: 10.1109/TVT.2022.3152597.

[35] E. Yigitbas, K. Karakaya, I. Jovanovikj and G. Engels, 'Enhancing Human-in-the-Loop Adaptive Systems through Digital Twins and VR Interfaces,' in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, IEEE, May 2021, pp. 30–40. DOI: `10.1109/SEAMS51251.2021.00015`.

[36] F. Dembski, U. Wössner and M. Letzgus, 'The Digital Twin – Tackling Urban Challenges with Models, Spatial Analysis and Numerical Simulations in Immersive Virtual Environments,' in *Architecture in the Age of the 4th Industrial Revolution*, vol. 1, eCAADe; SIGraDi; FAUP, Sep. 2019, pp. 795–804. DOI: `10.5151/proceedings-ecaadesigradi2019_334`.

[37] F. Dembski, U. Wössner, M. Letzgus, M. Ruddat and C. Yamu, 'Urban Digital Twins for Smart Cities and Citizens: The Case Study of Herrenberg, Germany,' *Sustainability*, vol. 12, Mar. 2020. DOI: `10.3390/su12062307`.

[38] Norwegian University of Science and Technology, *NAPLab - NTNU*. [Online]. Available: `https://www.ntnu.edu/idi/naplab`.

[39] A. Soycan and M. Soycan, 'Perspective correction of building facade images for architectural applications,' *Engineering Science and Technology, an International Journal*, vol. 22, no. 3, pp. 697–705, Jun. 2019. DOI: `10.1016/j.jestch.2018.12.012`.

[40] E. Mok and G. Retscher, 'Location determination using WiFi fingerprinting versus WiFi trilateration,' *Journal of Location Based Services*, vol. 1, pp. 145–159, 2 Jun. 2007. DOI: `10.1080/17489720701781905`.

[41] P. S. Heckbert, 'Survey of Texture Mapping,' *Computer Graphics and Applications*, vol. 6, pp. 56–67, 11 Nov. 1986. DOI: `10.1109/MCG.1986.276672`.

[42] W. Lyu, Z. Zhou, L. Chen and Y. Zhou, 'A survey on image and video stitching,' *Virtual Reality & Intelligent Hardware*, vol. 1, pp. 55–83, 1 Feb. 2019. DOI: `10.3724/SP.J.2096-5796.2018.0008`.

[43] Z. Tang, R. G. von Gioi, P. Monasse and J.-M. Morel, 'A Precision Analysis of Camera Distortion Models,' *IEEE Transactions on Image Processing*, vol. 26, pp. 2694–2704, 6 Jul. 2017. DOI: `10.1109/TIP.2017.2686001`.

[44] D. Jeong and Y. Kim, 'Keypoint-based Deep Learning Approach for Building Footprint Extraction Using Aerial Images,' *Korean Journal of Remote Sensing*, vol. 37, no. 1, pp. 111–122, Feb. 2021. DOI: `10.7780/kjrs.2021.37.1.9`.

[45] P.-R. Chen, S.-Y. Lo, H.-M. Hang, S.-W. Chan and J.-J. Lin, 'Efficient road lane marking detection with deep learning,' in *2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, 2018, pp. 1–5. DOI: `10.1109/ICDSP.2018.8631673`.

[46] D. Neven, B. de Brabandere, S. Georgoulis, M. Proesmans and L. van Gool, 'Towards End-to-End Lane Detection: an Instance Segmentation Approach,' Jun. 2018. DOI: `10.1109/IVS.2018.8500547`.

[47] P. Fischer, S. M. Azimi, R. Roschlaub and T. Krauß, ‘Towards HD Maps from Aerial Imagery: Robust Lane Marking Segmentation Using Country-Scale Imagery,’ *International Journal of Geo-Information*, vol. 7, no. 12, Nov. 2018. DOI: `10.3390/ijgi7120458`.

[48] A. Gruen, X. Huang, R. Qin, T. Du, W. Fang, J. Boavida and A. Oliveira, ‘Joint processing of UAV imagery and terrestrial mobile mapping system data for very high resolution 3D city modeling,’ *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-1/W2, pp. 175–182, Aug. 2013. DOI: `10.5194/isprsarchives-XL-1-W2-175-2013`.

Part IV

Appendices

'But how will people get around? They will have to buy my latest invention! It's like a car, but it can drive on the surface of water! Behold! You like it? I call it the "Buoyancy Operated Aquatic Transport", or "BO-AT" for short.'

— Dr. Heinz Doofenshmirtz
PHINEAS AND FERB

Appendix A

Code

All the code used for this thesis is written in Python, and can be found at `https://github.com/al3xknutsen/tdt4900-master`. Note that the code will not run as-is—file paths have been anonymized, and certain asset files like height maps and orthophotos are not distributed due to copyright laws. Nonetheless, the repository contains package requirements and some simple instructions if running the code is deemed desirable.

Appendix B

Meta Quest 2 Controls

Figure B.1 gives a reference to the control bindings available for the controllers of the Meta Quest 2 VR headset.

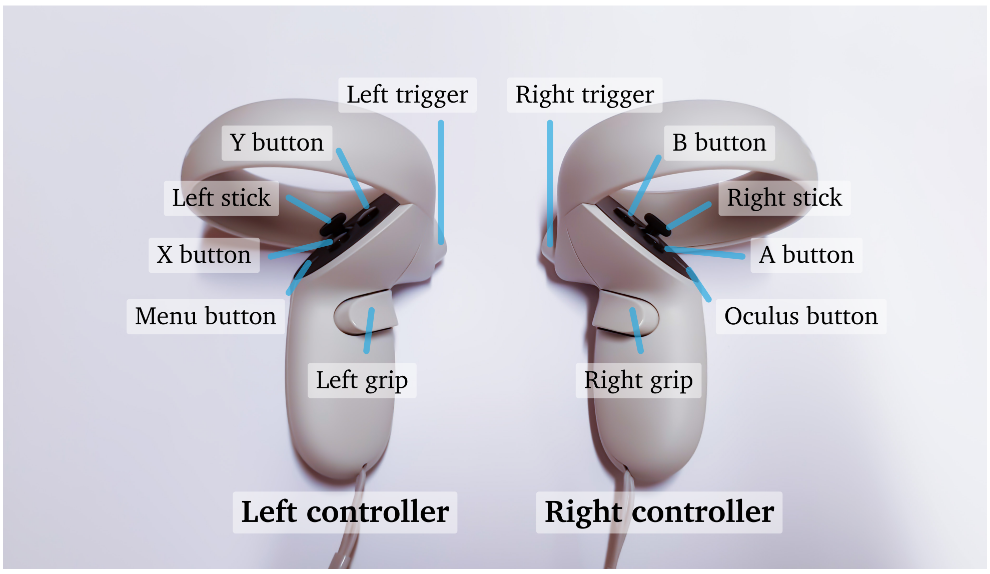


Figure B.1: Map of input controls and their names belonging to the controllers of the Meta Quest 2 VR headset.

Appendix C

3D Clip&Ship API Reference

This appendix contains a reference to the usage of the 3D Clip&Ship REST API. Table C.1 contains an attribute reference for acquiring access tokens to the service.[1] Likewise, Table C.2 contains a reference for the building export itself. Listing C.1 contains an example string used for retrieving access tokens. Furthermore, Listing C.2 contains an example of a geometry object used to define the borders of the geographical area to export. Finally, Listing C.3 shows an example string used for performing the building export.[2]

Code listing C.1: Example message sent to the Geodata Online API in order to receive an access token.

```
{"username": "olanormann",
 "password": "verysecurepassword123",
 "client": "requestip",
 "expiration": 1440,
 "f": "pjson"}
```

Code listing C.2: An example of a geometry object used for specifying the border of the Clip&Ship export. This object is used as a value for the `omrade` attribute. Text in SMALL CAPS are variables—EPSG is the EPSG code of the chosen coordinate system, and COORDS is a list of coordinate pairs in that coordinate system defining the outer boundary of the area to export.

```
{{"geometryType": "esriGeometryPolygon",
    "features": [{{
        "geometry": {{
            "rings": [COORDS],
            "spatialReference": {{
                "wkid": EPSG,
                "latestWkid": EPSG}}}}}}],
    "sr": {{
        "wkid": EPSG,
        "latestWkid": EPSG}}}}
```

[1]URL: `https://services.geodataonline.no/arcgis/tokens/generateToken`

[2]URL: `https://services.geodataonline.no/arcgis/rest/services/Geoeksport/3DClipAndShip/GPServer/ClipAndShip3D/submitJob`

Attribute name	Description	Data type	Possible values
username	Geodata Online username	String	—
password	Geodata Online password	String	—
client	What client the token should be valid for	String	• ip[1] • referer[2] • **requestip**[3]
ip[†]	Client IP address (use only if `client = ip`)	String	—
referer[†]	Client web URL (use only if `client = referer`)	String	—
expiration	Token validity (in minutes)	Integer	$\in [0, 525\,600]$
f	Response format	String	• **pjson**

[†] Only used if `client` $\neq$ `requestip`, i.e. for this thesis, they were ignored.
[1] Token should only be valid for one explicitly given IP address.
[2] Token should only be valid for one explicitly given URL.
[3] Token should only be valid for the IP address that requested the token.

Table C.1: Attribute reference for obtaining access tokens for the Clip&Ship service. All other API calls must have a valid token. Values marked in **bold** were used for this thesis.

Code listing C.3: Example message sent to the Clip&Ship API to export buildings.

```
{"omrade": {{"geometryType": "esriGeometryPolygon", "features": [{{"geometry": {{"
    rings": [[[270635.22491466044, 7040341.124050389], [270682.85000991065,
    7040212.800877076], [270612.73528634786, 7040183.035192545],
    [270559.1570541914, 7040316.650043108], [270635.22491466044,
    7040341.124050389]]], "spatialReference": {{"wkid": "25833", "latestWkid":
    "25833"}}}}}}], "sr": {{"wkid": "25833", "latestWkid": "25833"}}}},
 "innhold": ["3D Bygg med taktekstur"],
 "format": ["Wavefront OBJ"],
 "navn": "bygninger1",
 "projeksjon": "EPSG:25833 (UTM33N)",
 "epost": "emailaddr@emailprovider.extension",
 "Web_projeksjon": "Samme som leveranseprojeksjon",
 "min_vegetasjon_hoyde": 2, "f": "pjson", "env:outSR": "25833",
 "token": "fdeuphpg94qu2put2942v1q0kr"}
```

Attribute name	**Description**	**Data type**	**Possible values**
omrade	Outer bounds of the area to export	Geometry object	*see Listing C.2*
navn	Name of export	String	—
innhold	What type of data to export	List of strings	• FKB (bygg, vann, veg) • 3D Bygg • **3D Bygg med taktekstur** • 3D Bygg med taktekstur og veggtekstur • Overflateobjekter (gjerder, stolper, etc.) • Ortofoto • Vegetasjon (laserdata) • Tre/skog (generert fra laserdata) • Detaljert terrengmodell
format	Export's file format	List of strings	• ArcGIS Online • ArcGIS Pro • AutoCAD DWG • IFC • 3D-PDF • Collada • Sketchup • 3ds • CityGML • LandXML • **Wavefront OBJ** • ArcGIS Enterprise
epost	Destination e-mail address	String	—

Continued on next page.

(Continued)

Attribute name	Description	Data type	Possible values
projeksjon	Export's map projection	String	• EPSG:25832 (UTM 32N) • **EPSG:25833 (UTM 33N)** ... • EPSG:25836 (UTM 36N) • EPSG:3857 (Web Mercator) • Lokal UTM-sone • Lokal NTM-sone • EPSG:5105 (NTM sone 5) ... • EPSG:5130 (NTM sone 30) • EPSG:4326 (WGS 1984)
Web _projeksjon*	Map projection of web delivery	String	• EPSG:25833 (UTM 33N) • EPSG:3857 (Web Mercator) • **Samme som leveranseprojeksjon**
min_vegetasjon _hoyde*	Minimum altitude of vegetation data (in meters)	Integer	> 0
f	Response format	String	• **pjson**
env:outSR	EPSG number of the chosen `projeksjon` value	String	$\in \{3857, 4326\} \cup [5105, 5130] \cup [25832, 25836]$
token	Access token	String	—

* Attributes need a value in order for the export to succeed, but their values are irrelevant for this thesis.

Table C.2: Attribute reference for exporting data from Clip&Ship. Values marked in **bold** were used for this thesis.

Index

‘...and if you were to ask me
after all that we’ve been through:
“Still believe in magic?”
Oh, yes I do.
Yes, I do.
Yes, I do.
Yes, I do.
Of course I do.’

— Coldplay
‘Magic’
Ghost Stories