Multiple View Texture Mapping: A Rendering Approach Designed for Driving Simulation

William Clifford

A dissertation submitted for the degree of

Doctor of Philosophy

Department of Computer Science

Maynooth University

Head of Department: Dr. Joseph Timoney

Supervisors: Dr. Charles Markham

September 2023

Acknowledgements

This thesis would not have been possible if it were not for those closest to me. My mam, dad, and my siblings, Sean, Peter, Nicole and Vive, have always been there and I cannot thank them enough for that. Whether it was listening to me rant about testing code or driving me to the final viva because my car broke down, they were always ready and willing to help.

I would like to express my sincere gratitude to my supervisor Dr Charles Markham, for his advice, support, and patience throughout. Over the many years of this thesis I have enjoyed his enthusiasm for research and his work ethic. I hope to have the same affect on the students I supervise in the future.

My time at Maynooth University would have been less fruitful if it weren't for the characters in Maynooth's computer science department. My labmates over the years Louis, Toby, the two Robs, Marie, Liam, Sutirtha, Keith, James and Sarav, I thank you all. Whether we shared a coffee, went to the gym, proofed each others slides and research papers, or experienced geometry together. It was all so helpful and certainly funnest part of doing a PhD.

I would like to thank the examining committee Dr Paul Miller and Prof. John McDonald for reading this thesis. Your comments were invaluable and I am grateful to you both for encouraging me to increase the standard of my work.

Last but certainly not least, my girlfriend Katie and her daughter Hazel. They came into my life precisely when I needed the motivation to finish this thesis. The idea of putting a close to this chapter of my life and starting a new one with them was a great driving force. Plus

the fact Katie was not afraid to tell me when I was slacking off. I look forward to our bright future together and hope that I can be as supportive and encouraging to the two of you as you have been to me.

Abstract

Simulation provides a safe and controlled environment ideal for human testing [49, 142, 120]. Simulation of real environments has reached new heights in terms of photo-realism. Often, a team of professional graphical artists would have to be hired to compete with modern commercial simulators. Meanwhile, machine vision methods are currently being developed that attempt to automatically provide geometrically consistent and photo-realistic 3D models of real scenes [189, 139, 115, 19, 140, 111, 132]. Often the only requirement is a set of images of that scene. A road engineer wishing to simulate the environment of a real road for driving experiments could potentially use these tools.

This thesis develops a driving simulator that uses machine vision methods to reconstruct a real road automatically. A computer graphics method called projective texture mapping is applied to enhance the photo-realism of the 3D models[144, 43]. This essentially creates a virtual projector in the 3D environment to automatically assign image coordinates to a 3D model. These principles are demonstrated using custom shaders developed for an OpenGL rendering pipeline.

Projective texture mapping presents a list of challenges to overcome, these include reverse projection and projection onto surfaces not immediately in front of the projector [53]. A significant challenge was the removal of dynamic foreground objects. 3D reconstruction systems create 3D models based on static objects captured in images. Dynamic objects are rarely reconstructed. Projective texture mapping of images, including these dynamic objects, can result in visual artefacts. A workflow is developed to resolve this, resulting in videos and 3D reconstructions of streets with no moving vehicles on the scene.

The final simulator using 3D reconstruction and projective texture mapping is then developed. The rendering camera had a motion model introduced to enable human interaction. The final system is presented, experimentally tested, and future potential works are discussed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Driving simulation provides a safe, practical, and effective means of studying driving behaviour [55]. However, with a lack of fidelity, the correlation between real driving and simulation cannot be guaranteed. Although great strides have been made in computer graphics to create photo-realistic computer-generated models [49, 65, 16]. There exist ways to automate the generation of 3D models for an experimental setting that is often not taken advantage of [189, 184, 139].

This thesis describes a method for creating a photo-realistic driving simulation that uses 3D reconstruction to design the model the simulator renders. Figure 1.1 highlights an example of a model-based driving simulator compared to a photograph of a real road. There are differences in lighting, shadows, colour, and detail in the geometry of the road. A person viewing the left image would be aware they are observing a driving simulation, and a level of immersion may be taken away, in comparison to the real road.

Within the past ten years, the Irish-roads infrastructure has received significant investment [167]. Interest has increased in trying to assess the impact of distraction external to the vehicle such as road works, advertising, and roadside art, see Figure 1.2a,1.2b, 1.2c respectively. Transport Infrastructure Ireland has identified a need to quantify the level of driver distraction that would occur following road infrastructure and layout changes. During this time, a collaboration

Figure 1.1: Left: Asset based simulator. Right: desired level of photo-realism for simulation.

(a) Roadworks (b) Advertising

(c) Road art

Figure 1.2: Roadside possible sources of distraction.

between members of Maynooth University and the Technological University of Dublin has joined in a project to monitor driver response to measure driver distraction using driving simulators and biofeedback technologies. This has been a major motivating factor for this research.

Much work has been carried out on in-vehicle sources of distraction, such as mobile phone usage, radio tuning, and the use of driving assistance systems [107, 157, 194]. Less attention has been given to sources of distraction outside the vehicle. This is because the world within the vehicle is well constrained and, for this reason, easier to control and make repetitive measures [55]. The research

developed in this thesis aims to develop an additional set of tools to analyse driver responses and aid in detecting and understanding distractions external to the vehicle. It is predicted that reconstructing the outside world based on modern sensors and modelling tools should capture previously unseen instances of distraction.

It is possible to create realistic driving simulators using advanced computer graphics techniques and accurate physics given enough expertise and resources. Although these simulators have high performance, it can take significant time and effort to create them. A digital artist's or developer's impression of the road may not truly reflect the state of a real road. Using images to automate the generation of 3D models for a driving simulator enforces that the model is based on the scene's appearance. Driving simulators have been created using advanced machine vision and automated methods to create random environments for autonomous vehicles to train on [48, 147]. Automated methods that create environments for autonomous vehicles use learned approaches to construct their 3D environment [48, 47, 119, 138, 3]. These approaches are slightly askew from ours since it creates and estimates environments instead of focusing on developing simulations of real authentic to the real world environments. It only matters to the automated approaches that the environments are visually convincing and allow autonomous vehicles and robots to transfer knowledge to the real world. It is planned to use these 3D reconstruction tools to construct a 3D model, to be rendered in a driving simulator and to include methods for testing driver sensory behaviour, emphasizing eye tracking. Eye tracking is one of the most popular methods of measuring driver attention [84, 129, 90, 22, 103, 107].

There exists a perspective projection relationship within computer vision methods to reconstruct the 3D models and how the 3D models are rendered using computer graphics techniques. A virtual rendering camera is abstracted to a set of projective and rigid body transformations in most computer graphics programs. This virtual camera transforms 3D coordinates of objects in a virtual environment into screen coordinates that are presented on computer screens. These transformations can be applied in the opposite direction, to convert eye gaze coordinates to rays that can be compared against 3D objects. This process is further discussed in Appendix B.1. The field of eye tracking has previously

considered 3D eye tracking; however, there appear to be few works where the 3D models were reconstructed from photographic data.

A driving simulator of the above description did not exist prior to the undertaking of this project. To construct a driving simulator with these capabilities the following problems would have to be addressed:

1. A review of the current state of the art for driving simulation and testing, 3D reconstruction, and computer graphics techniques.

2. Experiments on real people to investigate important features to maintain within a driving simulator.

3. A workflow to automate 3D reconstruction of real roads and to deliver a photo-realistic rendering.

There exists, some of the early work carried out by Maynooth University and Technological University of Dublin including eye-tracking experiments on drivers within a photo-realistic driving simulator. These experiments used dashboard camera footage and allowed the participants to control video playback with an accelerator and brake pedals [81]. Ideally, providing steering in a video-based simulator would require a full 3D reconstruction of the road or clever use of specialized wide-angled cameras with computer vision techniques. The problem of 3D reconstruction is non-trivial and requires significant investigation for full development. A previous approach implemented by the group used 2.5D measurements (stereo camera pairs providing depth measurements) and reprojected the environment found from depth map estimation to simulate a 3D environment [14]. This resulted in a sparse representation witnessed by the viewing camera and led to visual artefacts, including non-continuous surfaces, while viewed from perspectives other than the viewing camera perspective. It is intended to continue this work and extend the research toward a robust solution, capable of observing reconstructions from perspectives with a transformation that is slightly off axis from the capturing camera, i.e. to enable lane changes or slight turning maneuvers.

1.2 Thesis Scope

This thesis presents a novel driving simulator that uses reconstructed 3D models of real roads and renders view-dependent texture mapping to increase photo-realistic renderings of a road. The principal contribution of this thesis is the synthesis of computer graphics, machine vision, and human testing technologies to produce a driving simulator as a photo-realistic experience. Driving simulator experiments are presented in this thesis to highlight features of driver attention and to gather requirements for the final video based driving simulator. The experiments include:

- A video based driving simulation where eye tracking is measured across participants and these participants are classified into groups of distracted and focused drivers varied over time. This work was presented at the European Conference of Eye Movement 2017 [34]. The analysis of this data is detailed in Section 3.1.

- An asset based driving simulation where participants were exposed to static and dynamic billboards while driving. Participants eye tracking was compared to discrete features on the road including the billboards, road signs, speedometer, and the center of the road. The findings of this experiment were presented at the International Conference of Driver Distraction and Inattention 2021 [35]. These eye tracking results were later matched to EEG measurements and presented in the journal of Sensors in 2021 [180]. The main findings in this paper included the demonstration of a fixation related potential within the brain that was triggered in relation to distractors. The contribution of the work done in this thesis towards that finding included the labelling of distractors, and the data processing and labelling of the eye tracking that enabled analysis of these potentials measured using EEG, see more detail in Section 3.2.

The video based driving simulator used projective texture mapping (PTM) to increase its photorealism. The use of PTM in combination with 3D reconstructed models, and localized camera poses will provide a method of:

- Assigning texture coordinates to vertices during render time using perspective projection. This removed the requirement to store UV-vertex information for every 3D point. The initial results of this were presented in Irish Machine Vision and Image Processing Conference 2017 [31]. This is detailed in Section 4.1.

- A view-dependent rendering of the 3D model. This was initially tested on a SLAM system called Elastic Fusion [184], see Appendix C.1. The estimated poses of the camera for each image relative to a 3D model were used during PTM. The formalization and implementation of this approach was presented in the Irish Machine Vision and Image Processing conference 2020 [33], see Section 4.2.

- Methods of inpainting/background subtraction using 2D template matching and later multiple-view texture mapping. The 2D methods were presented at the Irish Machine Vision and Image Processing Conference 2019 [32], see Section 5.2. The multiple view texture mapping approach has not yet been published. This is however detailed in Section 5.3

1.3 Thesis Structure

The representation of the road will be based off video sequences collected from camera mounted vehicles (ideally a dashboard camera). Based on what has been discussed in this chapter, to generate an environment capable of driving simulation, the sequences will be processed using modern machine vision and computer graphics techniques. These will be created with three main points in mind:

1. Photo-realism
2. Freedom of motion
3. Methods to measure driver attention

Chapter 2 reviews research in the area of driver distraction, and driving simulation to give context as to what is expected of the final driving simulation and

what is currently available. Machine vision and its function in reconstructing 3D assets and simulating environments using view synthesis will be discussed and considered for the requirements of the driving simulator.

The initial objective of a photorealistic simulator is achieved by showing real-world videos of the road to simulate the perspectives observed during video collection to participants. This was first approached by replaying frames of video which can be controlled only in terms of speed by adjusting the framerate of the video playback. This method accomplishes the first objective but takes away from the freedom of motion by not allowing a steerable environment. The construction of this environment was detailed in [81]. To measure attention on the road an analysis of eye-tracking data of participants within the simulator is detailed in Chapter 3. The remainder of this chapter considers an asset-based driving simulator and driver distraction experiments. Experimenting with both of these setups will aid in the gathering of further requirements for the driving simulator, as well as measuring driver attention.

To introduce free movement into the simulation while maintaining the photo-realism of video, PTM (PTM) is implemented. This is a method of mapping texture from images to 3D surfaces in a computer graphics environment by using a projective transformation [144, 53]. The implementation is detailed in Chapter 4. PTM requires an image to project and a 3D model to project onto. Several 3D proxies are considered in Chapter 4; including an open-ended box (to simulate the geometry of a tunnel), and a 3D model automatically reconstructed from SLAM (Simultaneous Localization and Mapping) [184] and SfM (Structure from Motion) [139] systems. The 3D geometry and the transformations of the capturing cameras are formalized to aid in the discussion of how best to combine the outputs of SLAM and SfM with PTM. The performance and applicability of each system and their use cases are compared and a final system is chosen as the optimal choice for this workflow.

Following the combination of PTM and SfM/SLAM, it was noticed that visual artifacts concerning unanticipated items shown in the image were projected but not modeled in the final geometry. This results in diminished photo-realism for the simulation. The solutions to these issues are detailed throughout Chapter 5. Initially, it is proposed to solve this problem entirely within the 2D image plane.

Semantic segmentation is used to localize vehicles within each image [199]. Following this the vehicles are removed from these images and replaced with pixel colours from other images within video sequences of the same road. This approach was tested on videos of the Moyglare Road, in Maynooth, Co. Kildare, Ireland. Following this approach, it will be extended to 3D, where PTM could be used to aid in the removal of vehicles from these videos. Multiple images would be projected onto the 3D model in all spaces, not including vehicles. This approach will be applied to videos from the KITTI dataset to allow for future benchmarking and comparison [63].

Chapter 6 reflects on the main contributions of this thesis, including the connection between PTM and 3D reconstruction. It reflects on the solutions to PTM challenges, including reverse projection, projection onto all 3D surfaces in front of the projector, and removing dynamic foreground objects from video. The chapter will end with a review of possible future work. The 3D reconstructed models will allow for eye-tracking data to be projected from the viewpoint of the viewer to the screen coordinates of their monitor, to the 3D model, and back to the images used to reconstruct the scene. This work opens the possibility of examining eye tracking under multiple contexts. If this is intended to be done in real-time, research into ray-picking methods may have to be considered. The vehicle motion models could also be extended to increase the simulator fidelity; Ackerman is proposed as a possible integration.

Chapter 2

From Review to Specification

2.1 Introduction

This chapter provides a comprehensive analysis of research on road safety, driving simulators, and driver distraction. The primary objective is to develop tools for measuring driver attention in response to external distractors on the road. The analysis includes driver and road safety reports, driving footage analysis, and driving simulators. The chapter discusses the definition of driver distraction and inattention, including internal and external distractions. It explores the challenges of replicating realistic 3D models of real roads in driving simulators and examines methods for automatically generating 3D models. Video-based data collection is emphasized, considering the use of easily accessible sensors and photo-realistic information. The chapter reviews video-based 3D reconstruction methods and their suitability for the driving simulator. It discusses the mathematical formalization of 3D models, camera models, and the pipeline for mapping 3D world space to 2D screen space. Different systems for 3D reconstruction using video data are analyzed, and their features are identified. The chapter also covers shading, texturing, and lighting of 3D models, as well as recent developments in view synthesis. The ability to aggregate visual fixations of different drivers over a single model from multiple perspectives is highlighted as an important feature of the video-based simulator. Finally, the chapter concludes with a review of eye-tracking applications related to driver distraction in a simulator.

2.2 Road Safety and Human Factors

Driver behaviour and simulation is an extensive research area with many textbooks, journals, and dedicated conferences [55, 126, 51]. It can be carried out using road safety and accidents reports or driver experimental studies; Sections 2.2.1 and 2.2.2, respectively [170, 103]. Reports can extract driver-specific and road-specific information contributing to road accidents. Accident studies inform the community of the factors associated with road accidents. These factors include the age of drivers, the time of day, consumption of alcohol, phone use, the geometry of the road, road markings, and road signage. Driver experimental studies that include monitoring driving performance allow researchers to examine distracting events. Simulators provide a safer environment to study individual drivers' responses to a distracting stimulus. The following sections develop these ideas in the context of the work done.

2.2.1 Road Incident Reports

Irish Road Engineers from Transport Infrastructure Ireland (TII) expressed interest in the effect of distraction from roadside advertising, art, and construction work on driver behaviour. This section reviews real-world driving experiments and road crash reports relating to these forms of distraction. These studies highlight the involvement of distraction in road crashes and near-crashes. The reports on real-world driving experiments are very detailed, but there is a noticeable lack of detail while describing external forms of distraction [84].

Often in driving research, a focus is put on examining distractions internal to the vehicle, an example of which includes the 100-car naturalistic study [84]. This major study was carried out for approximately 2,000,000 vehicle miles of driving and accumulated 43,000 hours of data over 12-13 months. In this study, 109 cars were fitted with cameras and other sensors for extended periods. Aside from the vehicle owner, other family members and friends occasionally drove these vehicles and accounted for an additional 132 drivers. The setting allowed drivers to interact with real roads (as opposed to a simulator). The drivers could be less concerned with being monitored by an experimenter in their vehicle and altering their driving in response. It allowed for the examination of driver

performance without the hindsight bias associated with road incident reports. Crashes in this study were defined as "any measurable dissipation of transfer of energy due to the contact of the subject vehicle with another vehicle or object". There were 82 recorded crashes and 761 near-crashes recorded throughout the study. Among other features, they examined when drivers looked away from the forward roadway by examining video data prior to crashes and near-crashes. They identified significant sources of distraction internal to the vehicle in search of causative sources of distraction (mobile phones, adjusting in-vehicle equipment, drinking, and eating). It was found that 80% of crashes and 65% of near-crashes were related to distractions internal to the vehicle. They attributed the remainder of crashes as either unknown or road infrastructure related. If internal sources of distraction caused 80% of crashes, it is possible that a proportion of the remaining 20% was related to external sources of distraction.

According to the National Crash Worthiness Data System, 29.4% of driver distraction-related crashes originate from outside people, objects, or events, which report similar figures to the naturalistic study [158]. These 20% to 35% of crashes are of interest to the work carried out in this thesis. In the National Crash Worthiness Data System, Stutts et al. highlighted each category of internal distraction and assigned a percentage to each category. However, for external factors, they were all categorised under a single heading receiving the highest percentage incidence, 29.4% [158]. The CRC handbook on driving simulation explains that most researchers may be deterred from researching this category of distraction because of the many factors contributing to external distraction from the vehicle [55]. In contrast, internal factors are better constrained. The CRC handbook also states that external distractors such as digital billboards have rarely been associated with road crashes, given the infrequency of reported events. The 100-car naturalistic study observed that more road crashes occurred than what would have been reported to the police (15 of the 82 crashes) [84]. Monitoring driver responses might be the best way of testing if these items on the road caused significant distractions to the driver to give cause for safety concerns, as not all crashes will be reported to the authorities.

Summary: While distraction external to the vehicle causes fewer reported accidents compared to internal sources, the number is still significant. A road

engineer also posed the problem of measuring external distractions relating to authorising roadside advertising, art, and maintenance. These factors informed the design of the simulator developed in this thesis. The simulator aims to allow a road engineer to collect video sequences from a section of the road of interest and then allow other drivers to drive the same section in the simulator. The road sequence might contain an existing object that is a potential distractor, or this might be added afterward by augmentation. The driving simulator would provide a safe means of testing and addressing these concerns. A 3D reconstruction of a road constrains the possible number of distractors on the road to the infrastructure itself. In a real-world environment, there are too many variables to consider while identifying external sources of distraction. Participant specific experiments address this domain gap, however safety can be a concern. Driving simulation is often used to address this issue at the expense of driving fidelity. Driving experimentation and simulation will be discussed further in the following sections. The system development within this thesis aims to reduce the complexity of identifying external distractors while also basing the simulator on a real-world road.

2.2.2 Review of Driving Experiments

The previous section reported driver performance from real camera footage and road accident reports [84, 158]. Distraction in these cases was only recorded if a crash or near-crash occurred. These studies noted a distraction event when a driver was witnessed or self-reported looking away from the forward roadway. In an experimental setting, it is possible to gauge the extent of driver distraction. These include several methods for monitoring the braking response in response to stimulus, speed, and progress along the road, EEG activity, and physiological factors such as galvanic skin response and heart rate [94, 55, 107].

Driving simulation meta data, including speed, lane position, braking distance from leading vehicles, and updates to vehicle controls can inform levels of attention and driving competence. Godthelp et al. tested the lane keeping ability of drivers while wearing a visual occlusion helmet; "an electromagnetically driven visual occlusion device mounted on a lightweight bicycle helmet" [66]. They found

drivers compensate steering behaviour based on vision. Compensatory steering (larger steering angles over time) was required when their vision was occluded and shown to them again. McKnight & McKnight examined the effects of cellular phone calls on the ability of drivers to respond to video driving sequences [107]. They found that older drivers had a significant impact on their ability to context switch and younger drivers were more affected based on the conversation intensity.

Perhaps the most used methods are based on measuring the focus of attention of the driver's eye while driving [90, 103, 157, 22, 127, 128, 73]. This section will discuss how attention is measured for drivers using eye gaze. It will begin by showing studies where human vision and driving are connected. Human vision, attention, and driving performance are connected; the question becomes, "what are drivers attending to on the road". Suppose a driver is visually distracted by an item on the road. In that case, that can be measured by querying the eye gaze of the driver and associating it with a semantic label (a high-level label of what an item is within a picture) of the item they are attending. This method will be used in experiments in this thesis. This will gauge when or if drivers are attending to distractions. It will measure levels of attention for the items on the road depending on the duration of visual attention caused by the distractor. Further details on the eye tracking process including collection, calibration, processing, and labelling can be found in Appendix AA.1.

An early example of driver attention and eye gaze as a measure of attention is discussed by Land et al. [90]. Land et al. used a head-mounted eye tracker on real drivers and a rigged Jaguar car with sensors for measuring the steering wheel angle and the vehicle's speed. Their study observed that as the steering wheel angle changed, the average eye-gaze direction followed into the bend of the road, corresponding to the angular change in the steering. Similar observations between eye gaze and tasks were found with other tasks [72]. This connection between eye-gaze and attention is essential in measuring distraction in driving. Other examples of observable behaviour include scanning strategies. Most drivers inspect the forward roadway while driving down straight roads to anticipate oncoming events/traffic. However, it has been noticed that experienced drivers tend to scan a wider section of the road compared to novices [22]. Visual scanning be-

haviour is a good predictor of the hazard awareness of drivers. Intuitively this makes sense, given that the drivers will have observed more of the road. This study further reinforces the observation that driving performance is connected to the visual perception of the environment. Eye-gaze measurement informs how likely it would be that a person has observed a given hazard or, for that matter, a distraction on the road. Recarte and Nunes tested cognitive workload effects of mental tasks to distract drivers [127]. This resulted in pupillary dilation and decreased horizontal and vertical scanning patterns. These effects were pronounced for mental visual tasks. Mind wandering affects glance patterns, and blink rates similar to cognitive distraction [73]. Experiments have also observed that driver eye gaze remained forward for more significant periods while they were cognitively distracted, using auditory tests [175] and handsfree systems [69]. Distraction can affect the ability of drivers to examine an environment, and the distraction does not need to be visual but visual distractions can further these effects.

Eye tracking studies in aviation have shown interest in what participants are observing in a natural environment (the class labels of the items the eyes are attending to) rather than how they are observing (variability/dispersion in their eye scanning strategies) [4]. This was done through an area of interest analysis for points of regard. Areas of interest are pre-defined boundaries usually categorised over an image plane of a viewing camera. In this case, the areas of interest included the controls used by the pilot to control the plane, see Figure 2.1. The point of regard is where the fovea (the direction the viewing part of the eye is pointed in) attention of some observer is looking with their eyes. Many eye-trackers calculate eye gaze by reflecting infrared light off the cornea and viewing it with an IR-sensitive camera [130, 117]. Area of interest analysis involves first highlighting an area on an image with labels that convey some semantic meaning. See Figure 2.2 for an example of a frontal view from a vehicle used in our simulator experiments in Chapter 3.

Screen coordinates corresponding to eye tracked points (points of regard) can then be compared to labelled regions of an image. The amount of time the gaze dwells within this region and the number of visits the areas of interest receives can be used as a measure of levels of attention/distraction. In the aviation study, they were concerned with measuring attention on the inside of the vehicle, which

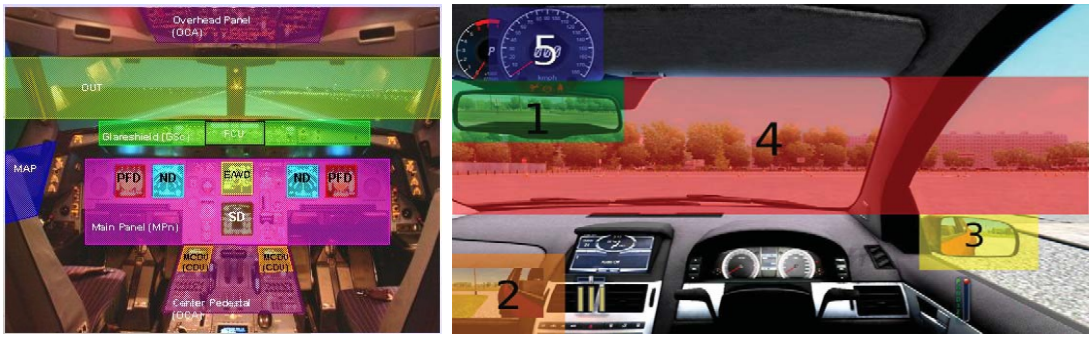


Figure 2.1: The image on the left is the labelled AOI's for the study on pilot attention during flight [4]. The image on the right is the AOI's used to measure the attention of drivers while doing dual-tasks with driving [103].

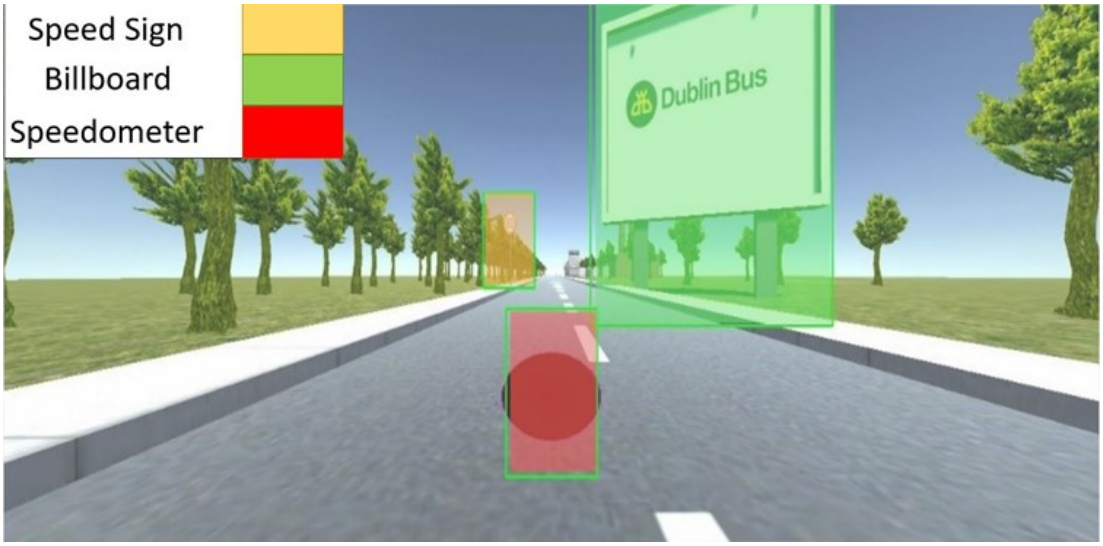


Figure 2.2: Driving simulation screenshot with highlighted areas of interest (AOI) from a billboard study (see chapter 3).

is static. Similar methods have been used more recently in driving research to measure attention within the vehicle while the participants were carrying out dual tasks [103]. In both studies, the area outside the vehicle was labelled under a single "outside", heading similar to the crash reports, see Figure 2.2. This is likely because this part of the scene was dynamic, and the task of labelling these features would be time-consuming. In this thesis, the areas outside the vehicle will be labelled, and the allocation of attention will be measured.

Summary: Traffic collision reports are informative and support the claim that visual driver distraction significantly contributes to road traffic accidents [170]. Driver distraction studies that directly observe drivers often report the sources of distraction inside the car [84, 157, 103]. While assigning labels items outside of this region, it is often a single label, "outside". With this single heading, external sources of distraction cause a higher incidence of collisions compared to all other categories of items within the vehicle. Further investigation into external sources of distraction is required to identify what factors outside the vehicle are involved. Testing on drivers can be done in the real world with camera-rigged vehicles where data is collected over long periods. This presents a serious investment for investigators. There are no guarantees that distracting events will occur in a natural driving setting without experimenter intervention, like in [84]. The likelihood increases over extended periods of time. If these events did occur, the next issue would be to investigate when this happened by annotating all the camera footage. Testing participants in a lab-based environment using a driving simulator is the safest and more practical method. Driving simulations keep the participant safe from any real harm and allow the experimenters to introduce distracting events within the driving simulation at times and locations of their choosing. It has been noted however that some driving simulators fail to replicate the exact environment most drivers experience, including vehicle controls, lighting conditions and the level of detail and complexity the real world has compared to a simulated environment [109]. The next section of this review will cover the various driving simulators available, and the novel features available in each. These features will be assessed for inclusion in the final driving simulator presented in this thesis. Previous studies have shown that measuring driver attention using driving performance and eye-tracking is possible. Eye tracking will be the primary

biosensor used in this thesis, and area of interest analysis will be used to detect and measure attention from participants. The driving simulator in this thesis will be built to measure human attention outside the vehicle.

2.3 Driving Simulators

Driving simulators are constructed to represent real-world driving without putting the driver themselves in the real environment. The CRC handbook states that driving simulation can often be safer and cost-effective compared to real driving experiments [55]. It allows repeatable measures to test drivers under specific circumstances, and experimenters have complete control of the driving environment. This section will start by describing the uses of driving simulation for the types of measurements available and the problems researchers are interested in. This section will then discuss historical driving simulations, highlighting their evolution, including practical video-based methods, analogue, and digital models, 3D models, and machine vision methods to simulate real roads. The history is provided as some of the earliest simulators use a moving camera over a model. These simulators are similar in concept to the method developed in this thesis.

2.3.1 Simulator Applications

Many driving simulator studies focus on testing environments unsafe for drivers in the real world. These environments include driver states such as fatigue, shifts of control from automation back to manual control, and inducing distractions to the driver during safety-critical moments. Everyday tasks used to test driver attention include braking behind leading vehicles that stop at pre-determined times or obeying speed limitations while attempting a secondary task like operating mobile phones or engaging with advanced driving assistance systems (ADAS) [61, 168]. These experiments require controlled environments using a driving simulator. The use of driving simulators allows driver ability and response testing without sacrificing safety [55]. It also enables the researchers to repeat rare hazardous events in real-life driving (passing distractions, crossing intersections, unexpected lead vehicles stopping).

Figure 2.3: Toyota's mechanical platform (Stewart platform) used to move a driving simulator cabin to introduce forces to the environment of the simulators operator and enhance fidelity.

Often road safety researchers and those interested in cognitive aspects of driver behaviour have employed third-party software for their driving simulators to measure the drivers' cognitive reactions [103]. Car companies often invest significant resources into their in-house simulators [20, 120, 49]. These simulators are often used to test specific criteria which would not be available to the researcher with a third-party simulator. Toyota developed their simulator to use haptic feedback in response to the simulator's motion to enhance user experience while driving inside a purpose-built cabin. This includes making the cabin vibrate, bounce, and rotate in response to the vehicle's environment simulating using a Stewart platform, see Figure 2.3 . This setup provided higher fidelity for the experience of driving.

Smaller scale research groups interested in the environment find ways of automatically reconstructing scenes. An example of this can be seen in a research group in Cyprus which used GIS (Geographic Information Systems) data from open street map to automatically construct their 3D models for their driving simulation [110]. Their system parsed GIS data, including positions and types of each feature, and rendered those objects in the corresponding positions within

the simulator, see Figure 2.4. Although this appears flexible, there are limitations, there is a limited range of assets, so features may appear to repeat, and some vital GIS information may be lost as there is no asset available to add it to the model. This repeated use of assets could take away from the immersion of the driving simulator. It is intended that within this thesis, a method will be used to automatically reconstruct 3D models, avoiding the requirement to create and use assets while measuring the attention of operators of the simulator using eye-tracking analysis.

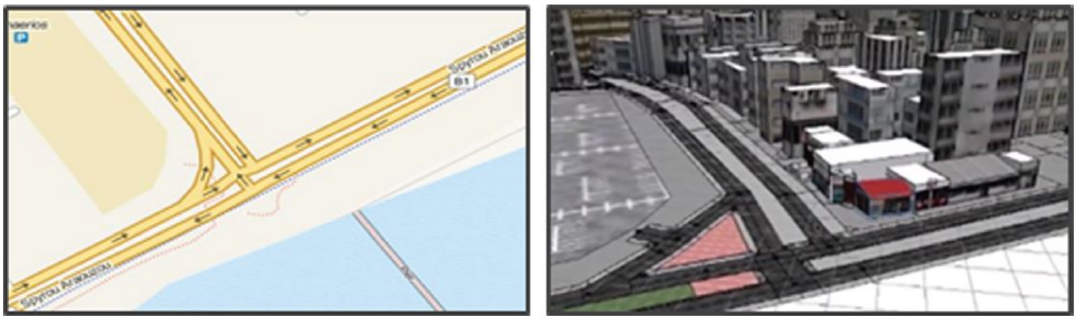


Figure 2.4: On the left is a map representation of Geographic Information Systems data and the right are the 3D models constructed from this data in [110].

2.3.2 Video-based Driving Simulation

A video-based simulation is an approach to automatically generating stimuli based on real data. This involves manipulating real videos of roads to provide the illusion that the participants are controlling a vehicle in the real world. This form of driving simulation dates back to the earliest driving simulators, over 60 years ago [77]. In this early study, a driving simulator was constructed using video film of a journey in a real vehicle where the camera was facing forward, the driver's view. Play-back for the driving simulator operators was presented on a projector. Steering was controlled by altering the pan of the projector(s), and speed was controlled by varying the playback speed; see Figure 2.5 for a concept drawing. This method provided high resolution (35mm film) with detail approaching real-world conditions, but it was described as arduous in its implementation [55].

Previous work in the author's research group developed methods to playback

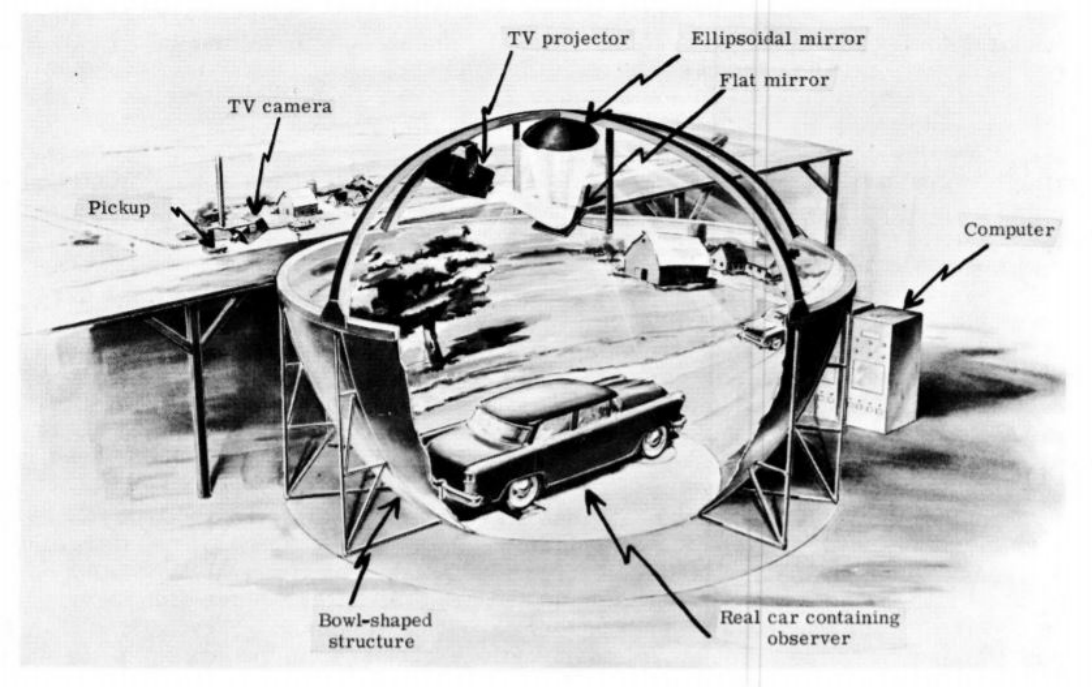


Figure 2.5: A concept drawing developed by Hutchinson describing a method of projecting real film footage streamed from some camera observing a representation of a road [77]. The operator being tested will be seated in the cabin of a car contained within a sphere that receives the broadcast projection of the road presented to the *TV camera*. There is a *computer* in the background used to compute vehicle dynamics and interpret this motion to the *TV camera*. This concept drawing can be found in a discussion section in [56].

video at a rate controlled by the accelerator and brake pedals. This was used in an eye-tracking study [81]. Later the generation of a model using GIS road data was also investigated [13]. This created a road-only model (with limited additional assets, mainly signs). A comparison between driving speeds between the video and model-based simulators was made. Since GPS data was available for the data acquisition vehicle, speeds were compared at some level with a real vehicle. Further work includes an element of steering developed by projecting video onto a simple model created using a stereo camera [14]. The approach used in this thesis is different. The methods used for projective texture mapping are built from a different foundation and are less likely to produce the distortions seen in earlier work. The generation of the models uses more recent SLAM/SfM-based reconstruction methods [184, 139]. This avoids the requirement for dedicated hardware to collect stereo/3D information. The work in this thesis develops methods for foreground object removal not available in earlier work.

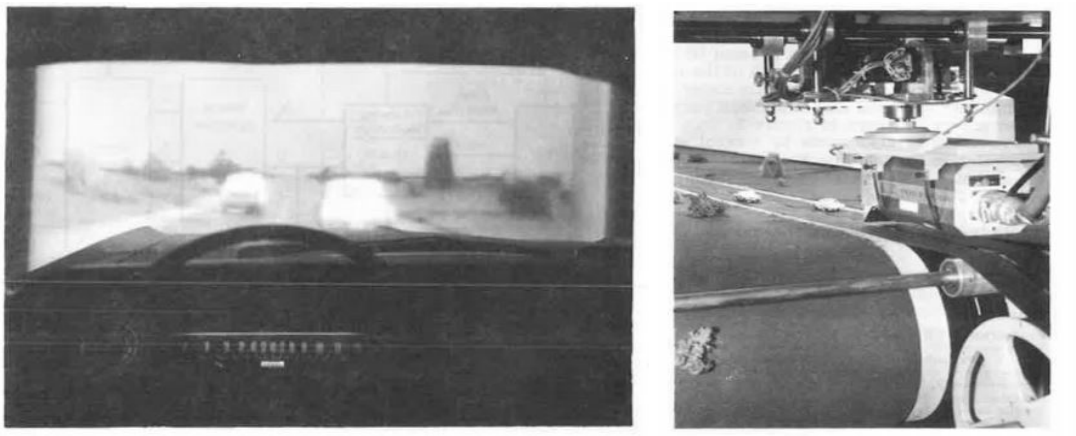


Figure 2.6: A scale model video based simulator [182]. The left image shows the drivers view and the right image shows the camera rig they operate and the belt system used to move the models.

Over ten years after the first video-based simulators, scale models were used to represent complex road geometry, including roadside objects like parked cars and trees [182]. These scale models were placed on a moving belt system in front of a camera; see the right image in Figure 2.6. The camera could maneuver (rotate) in response to the driver's steering input. The belt under the scale models for the road would vary its speed with respect to the accelerator and braking controls

given by the operator; see the left image for the driver's view and the right image for the camera rig and scale model in Figure 2.6. The steering and speed differential motion to simulate a real vehicle was computed on another computer, and it was possible to simulate the motion of a range of vehicles. The camera was connected to a screen presented to the operator while controlling the vehicle. This approach suffered from low-resolution lighting issues (unrealistic lighting for outdoor environments). These simulators were soon replaced by digital 3D models, which required less physical setup and often produced good resolution and lighting. However, the idea has merit and has many features in common with the simulator developed. It uses video of a real scene, and the driver can move the camera through the scene under their control.

2.3.3 Digital Driving Simulation

In the early 1970s, there was a transition from video-based to digital display simulators. This allowed researchers to design specific environments and increase the maneuverability of the simulation with less effort in terms of physical setup. Figure 2.7 shows an example of a 2D digital display driving simulation from [46]. It highlights the forward roadway from the perspective of a vehicle with well-labelled boundaries for the lanes and sides of the road. This 2D display study was used to test driver steering behaviour in response to increased speeds and variable curvature on the road. The simulated environment allowed human adaptability and driver limitations testing in response to the steerable environment. Although more complex digital driving simulations existed at the time and could produce more complex scenes, including obstacles and roadside signs/distractors [64], see Figure 2.8, they were limited in terms of run time. The serial processors of the time could not handle the number of objects they had to render and the vehicle's dynamic models. This resulted in delays in image generation. These display delay artifacts could alter participants' performance, as noted in a flight simulation study on pilot reactions to variable image generation delays [131]. They found flying formations challenging to complete at a low-frequency image presentation rate. The pilots require a high frame rate for accurate control of the aircraft and quick responses to errors. It was considered likely that these issues would carry

over to driving simulation for delays over 100 milliseconds per frame update [55].

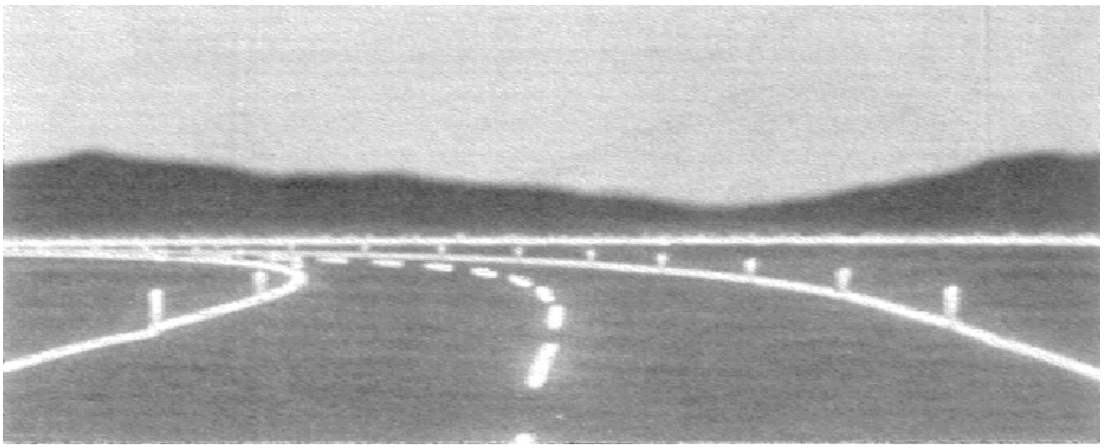


Figure 2.7: An example of an analog display driving simulator [46] that enabled steerable simulation experiments.

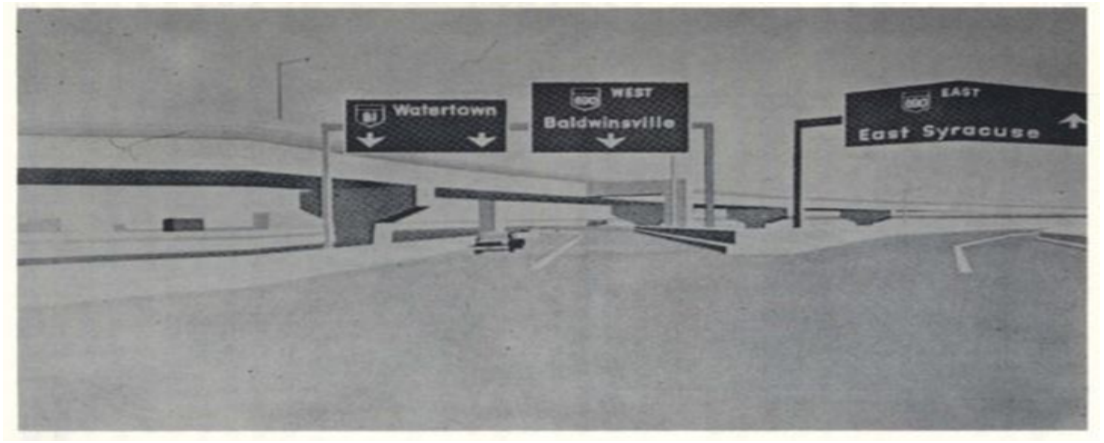


Figure 2.8: Early digital display driving simulators presented by Gilliland in a discussion on the applications of computer generated driving simulators [64].

Over the last 30 years, graphics cards and graphical processing units have allowed faster serial and parallel processing of these 3D environments [38]. This, in turn, resulted in driving simulation seeing significant development in digital displays, which have provided higher frame rates with better resolution. 3D scenes could be rendered in greater detail, including shadow calculations and light reflection models [124, 178]. An example of these displays is the Daimler-Benz

driving simulator [49]; see Figure 2.9[1]. This example high- lights the capability of driving simulators to represent more photo-realistic and complex scenes. It also allows experimenters to test multiple lighting conditions for studying the effects of night-time against daytime driving. Following the introduction of texturing and light models, very few simulators appeared to focus on the photo-realism of driving simulation. Testing on driver response to different rendering techniques would focus on night driving, adaptive headlights, fog, rain, and drivers' reaction to the high dynamic range [181, 18, 123]. Most of the literature on state-of-the-art driving simulators focuses on advancing haptic response and vehicle dynamics; these include the University of Iowa National Advanced Driving Simulator, Toyota's simulator, and the University of Leeds driving simulator [20, 142, 37, 120]. In these experiments, the focus has been on how the driver controls the vehicle regarding lane keeping, speed control, and response to safety-critical events, like braking on time. The simulators mentioned use an orbital cabin on a Stewart platform. The cabin contains real vehicles, and the road is presented with projectors, similar to Figures 2.3 and 2.10. Although not reflected in driving simulator research, computer graphics techniques have continued to develop, including physical-based rendering, photometric lights, planar reflections, and ray-traced shadows, which have been implemented in game engines such as Unreal Engine 4 [52]. Gaming driving simulators tend to embrace these rendering features and are, for the most part, racing simulators [143, 78, 173]. These advances can also be seen in realistic world simulations such as Airsim, built on top of Unreal Engine 4 [147], see Figure 2.11. Airsim was built to extend the 3D environments used to train machine learning algorithms, given their dependence on large amounts of photo-realistic data. They provide dynamic vehicle models as well as realistic physical environments. Airsim can also return multiple sensory measurements from the perspective of some set of viewing cameras, such as depth maps, semantically labelled images, and colour images.

The increased capability of personal computers through graphics cards and multi-core processors enabled more development of digital environments for researchers and enthusiasts. Initially, it was required to write shaders to instruct

[1]These images would be in colour for the actual simulator but the published research paper they were taken from was in black and white.

Figure 2.9: The Daimler-Benz driving simulator using lighting models to simulate real world light in different time of day settings. Left to right it shows a standard overcast day, bright light, and dark nighttime driving scenes [49].

Figure 2.10: University of Leeds Driving Simulator. The left image is the motion cueing cabin used for haptic feedback during simulation. The right image is the inside of that cabin showing the projection of the simulation on the walls.

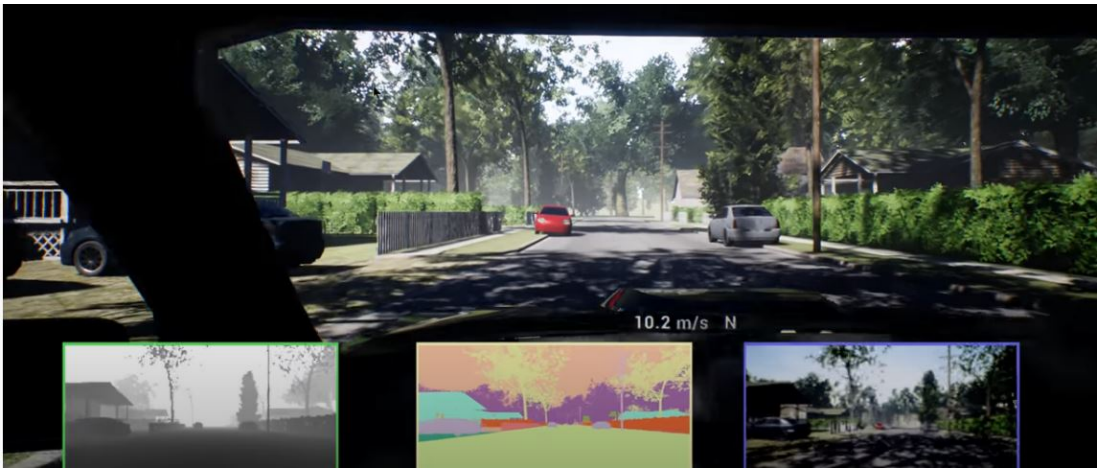


Figure 2.11: An example of Airsim driving simulation the main screen shows the drivers perspective, the sub-screens across the base show a depth image, a sematically labelled image, and a colour image in that respective order left to right, all from the perspective of the front of the vehicle [147].

graphics cards on how to interpret 3D data for rendering, often through APIs like OpenGL [145]. More recently, game engines like Unity, Unreal Engine, and OpenSceneGraph have provided a development environment that reduces the knowledge required of the rendering pipeline for the software developer [164, 52, 154]. A study using unreal engine and unity as a stimulus for their driving simulator found that drivers' gaze behaviour varies in response to different lighting models presented to drivers [109]. The experiment involved examining drivers' gaze and vehicle operation behaviour while drivers used augmented reality instructions (animated arrows for direction on the road). Driver attention was measured during a safety-critical event where a pedestrian crossed the road in front of the driver's vehicle. They found that the distribution of the driver's attention shifted significantly to the augmented reality instructions from the pedestrian while in the high graphical fidelity simulation compared to the lower fidelity graphical simulation. This finding showed that graphic fidelity factors were significant while measuring driver attention in simulators.

2.3.4 Machine Vision and Driving Simulation

The number of driving simulators created from machine vision methods appears to be a niche area. As a result, this section is shorter. Computer graphics and vision methods have allowed videos of the real world to be transformed into navigable 3D environments. Driving simulators have been developed using image mosaics from omnidirectional cameras using image projections over manually reconstructed 3D scenes [163, 118]. In their approach, it was possible to compute a texture-mapped solution to rendering real roads with their 3D model and an estimation of the set of camera poses and parameters (field of view, skew, radial distortions). The texture mapping technique splits a panoramic image into vertical sections (they refer to this as slits). These slits were then oriented in front of the rendering camera depending on the relative transformation between the rendering camera and the 3D model. The slits were textured using a set of local images sampled to construct mosaics. The slits used in the final rendering did not have the constraint of being from a single image. As a result, new renderings of the 3D model could be produced. The method applied in this example assumed that the capturing camera travelled in a straight trajectory. The assumptions on forward camera motion and a predefined 3D model are no longer required for modern machine vision methods while localizing a camera [171, 85, 189, 116, 139]. A projective model associated with a camera used to record dashboard footage can be matched to other images of the same scene to make a depth image. The approach in [14] was to project images into a single sample depth view made from an image pair. Although this method provided 3D geometry to steer around while viewing from the perspective of the capturing camera, when the user required a trajectory that deviated slightly from the capturing camera, holes in the depth image were exposed, leading to visual artefacts.

Recently data driven approaches to simulated environments have been developed [3, 190]. Companies have gotten involved in creating photo-realistic driving simulation including Parallel Domains and Facebook Reality Labs [119, 138]. Their main motivations with these simulations is to allow robots and autonomous vehicles to transfer learned skills from their synthetic environments to the real world. A roads engineer however would be interested in constructing a simulation

as close to a real road, as possible. Some of these data driven approaches appear to require a multitude of sensors, not typically made readily accessible. For each of these reasons these approaches have not been applied within this thesis.

2.3.5 Summary

Driving simulation has been used in safety, psychology, and engineering research for many years. It offers a safe alternative to real-world driving, where repeatable measures can be taken, and the environment can be controlled. Real-world information has been used to automatically construct simulated environments since the 1950s, with video-based driving simulation [77]. Digital computers were used to replace video-based driving simulators to allow experimenters more control over the environment and to introduce higher degrees of mobility. Digital simulation continued to advance as computing power increased, allowing for 3D and dynamic lighting environments. This allowed for the introduction of complex scenes, including dynamic traffic and pedestrians. In the driving research community, there appears to be less emphasis on keeping up to date with modern methods for creating photo-realistic environments for driving simulators. The focus appears to be on creating accurate physics models and high-fidelity haptic feedback. This may be because there are less noticeable differences in driver performance for increases in photo-realism compared to the standards set by car companies in the 2000s [20, 142]. However, recent studies have shown that human attention, through eye gaze, is influenced by the level of photo-realism [109]. Photo-realism in simulations has been expanded in the gaming industry, machine vision, and robotics research communities. Photo-realism in driving simulation is of interest within this thesis because the original simulator requirements focused on human-driver attention on real roads. Attention will be measured using human vision, so it is desired to get the driver's eye gaze behaviour within the simulator as close as possible to the real world. Machine vision methods have previously been used to automate the reconstruction of 3D environments for driving simulation. The work in this thesis expands on the premise that 3D reconstruction can be automated and enhances steering in video-based simulations. This thesis provides a solution to projecting image data onto reconstructed surfaces. This is

done by projecting the images onto a dense volume rather than a discrete depth map. A dense model with known relations between itself and the capturing camera extrinsic parameters allows for new forms of analysis in driving simulation. Interactions between users of the simulator and the 3D model can be mapped back to one of the 2D images used for the reconstruction. Since reconstructions will be used for the simulator, augmentation of distractors will also be possible.

2.4 3D Scenes and Reconstruction

Driving simulation can benefit from modern computer vision and graphics techniques for 3D reconstruction. The methods of photogrammetry and machine vision used for 3D reconstruction will be introduced and reviewed. A 3D reconstructed environment will provide a good representation of real roads for driving simulation, reducing the time and effort required to manually construct 3D assets using 3D design software. Building the pipeline for 3D reconstruction takes more time than designing a single road. However, the cumulative time of designing many roads will exceed the design of the 3D reconstruction pipeline. This section will mathematically formalize how a 3D scene and a camera model are composed to describe the chosen approach better. This formalization will be used throughout the thesis for computer graphics and machine vision sections. The camera model will be beneficial as perspective projection techniques (using a camera model) will be used to enhance the texturing of the 3D reconstructed models later in the thesis.

2.4.1 3D Reconstruction

3D reconstruction from image sequences forms an essential component of the video-based driving simulator. The following section reviews existing methods and discusses their relevance to the problem solved. Specifically open source implementations have been focused on given these solutions are readily available for testing and validation from the wider computer vision community. Reconstruction of scenes using colour camera images has been an area of interest for those in photogrammetry, robotics, and machine vision communities for many

years [98, 42]. The most popular algorithms for 3D reconstruction from camera images include structure-from-motion (SfM) and visual simultaneous localization and mapping (V-SLAM). These algorithms take images as inputs and produce a set of localized poses for each image and a relative 3D structure representing the geometry the cameras observed in each image. Basic SfM and V-SLAM pipelines will be outlined to highlight the benefits of the different types of systems. A detailed discussion of the main compoenets of SfM can be found in Appendix AA.2.

Structure-from-Motion: Longuet-Higgins appears to be among the first examples in computer vision to present a method for creating 3D structures from two views [98]. They presented an approach for estimating camera poses using 8-point correspondences from two views to estimate the camera's epipolar geometry. Since then, full pipelines have been developed to create structure from many views [152, 189, 160, 58, 115, 139]. These systems contain two main components: correspondence search and incremental reconstruction. Given a set of images of the same scene, correspondence search focuses on identifying unique visual features between images that can be detected from different perspectives. The output is a scene graph, highlighting robust features that connect the images. The identified features are filtered to ensure the 3D reconstruction optimizations and triangulations in the incremental reconstruction are not corrupted by outliers. The correspondence search uses feature extraction, matching, and geometric verification. Incremental reconstruction focuses on taking the matched features from the scene graph and building a 3D geometry. This is done using the densely overlapped images first and adding additional images to the set. Additional images are used to expand the 3D geometry and optimize the existing triangulated points towards a good solution. The output is a set of poses for each image and a set of 3D feature points connecting the poses. It does this in the following order: initialization, image registration, triangulation, and bundle adjustment. The steps in structure-from-motion will explain why some reconstruction tools were chosen over others.

Structure-from-motion has previously been used to reconstruct extensive image collections of tourist attractions initially found on the Internet using publicly available software [152, 189]; see Figure 2.12. This acted as a demonstration of

how SfM could be applied robustly applied to real world datasets. SfM would later be expanded to city-wide reconstructions, including reconstruction of popular cities for tourism, including Rome [57, 1, 188]. A significant advantage of SfM is that the reconstruction process is usually performed offline. The advantage here is that more effort can be put into forming the scene graph based on feature matching. This increases the number of iterations of RANSAC for selecting features to compute geometric verification and image registration. Searching for two views for initialization of incremental reconstruction also has increased runtime. Bundle Adjustment can often fall into incorrect local minima when not initialized with a pair of images containing features that overlap with many other images. Modern approaches to SfM find a good initial pair of images with a large set of shared features between themselves and all/most other images in the dataset. This increases the likelihood bundle adjustment will optimize towards a good solution for camera poses and 3D surfaces.

Figure 2.12: The left image is a collection of unstructured photographs of Notre-Dame, and the middle image is a reconstruction of the 3D surfaces and the viewpoints. The right image is the new way of browsing photos using Photo Tourism [152].

There are multiple implementations of the structure-from-motion pipeline. The most popular and publicly available algorithms include COLMAP [139], Theia [160], OpenMVG [115], visualSFM [189], Bundler [152], and MVE (Multi-view reconstruction environment) [58]. Each of these has been evaluated by [8] to show the best performing algorithm for sparse 3D reconstruction, pose estimation, and dense 3D reconstruction. This review found that COLMAP produced the best results on average, but comparatively, the results are quite close between itself and OpenMVG. This is the case for most comparisons involving both SfM

pipelines [155]. A thorough examination of 3D reconstruction pipelines was done by [136], where they compared dense 3D reconstructions using COLMAP and OpenMVG. These were combined with multiple view stereopsis (MVS) pipelines that constructed dense meshes from the sparse point clouds given by SfM [59]. The MVS pipelines used were those that were developed in combination with the COLMAP application [140] and OpenMVS [19]. These 3D meshes were constructed using aerial photographs of Dublin city centre and compared against ground truth LIDAR scans from the DublinCity dataset [202]. They reported measurements of precision, recall, and an F-score (a combination of the previous two scores). The full COLMAP pipeline had higher precision compared to OpenMVG. However, the OpenMVG and OpenMVS combination acquired a higher recall and F-score measurement.

Reasons for performance difference: Bundler is one of the earliest examples of SfM running successfully on extensive image collections [152]. Most other pipelines have either been built on this or heavily influenced by it. VisualSFM is a GUI application for 3D reconstruction, and it focuses on increasing processing speed through multi-core parallelism for feature detection, matching, and bundle adjustment. VisualSfM integrates PMVS (Patch-based Multi-view Stereo)/CMVS (Clustering Views for Multi-view Stereo) to extend the outputs of SfM (a set of camera poses and a sparse 3D model of image features). This extension reconstructs a dense 3D structure of visible objects in the scene observed in the set of images [60]. Only static structures should be reconstructed, automatically ignoring non-rigid objects. VisualSfM gained popularity as it was the first accessible 3D reconstruction pipeline from image collection to 3D models. MVE has been built with a focus on reconstructing detailed 3D objects with a set of images taken using multiple resolutions [58]. OpenMVG and Theia are both end-to-end structure-from-motion libraries capable of including many implementations of the different stages of SfM [115, 160]. However, OpenMVG has multiple published works focusing on their improvements to the SfM pipeline.

OpenMVG has used:

- **Adaptive thresholds** (a contrario) to improve the accuracy of the final 3D model [114, 112]. The thresholds adapted in this paper were image feature thresholds for RANSAC used for projective model estimation during image

registration for both image-to-image and image-to-world correspondences and pose estimation.

- A method to find 10% more (than Bundler and other existing methods at the time) multi-view tracks based on two-view image correspondences [113].

COLMAP was explicitly developed to be robust to outliers with each improvement it introduced to the SfM pipeline. It does this by editing the following steps [139]:

- **Geometric Verification**: The scene graph is augmented to include information on the transformations between corresponding images. The number of inliers for each type of projective transformation (homography, essential, and fundamental matrix) was recorded. Based on the number of inliers, the best of these transformations can be chosen to describe their geometric relationship best. A homography can be omitted to distinguish panoramic and planar scenes from others by using a threshold based on the median triangulation angle of the inliers.

- **Next best view selection**: to improve accuracy selecting new views for incremental reconstruction is based on the number of image features and their distribution across the image plane. Preference is shown for images with a uniform distribution of matched features currently triangulated in the 3D geometry. The uniformity of the features is computed by splitting the image plane into a grid and categorizing a cell as empty or full. If a point becomes visible inside an empty cell, it is re-assigned to full, and some weight updates the distribution score of the image. This approach is extended to multiple resolutions to reduce the influence of scale.

- **Robust triangulation**: Matching techniques favour image pairs with a similar appearance which usually means that the image pair will have a small baseline. Feature tracks are formed from concatenating two-view correspondences to form larger baselines between images. RANSAC is used to deal with the high number of outliers in feature tracks. Triangulated points are not calculated from image pairs labelled as panoramic to avoid

degeneracy. The triangulated points should display sufficient triangulation angle and positive depths. New triangulated points conform to the model if the depths are positive and the reprojection error is below some threshold, t.

Visual Simultaneous Localization and Mapping: SLAM has been an active research area for many years [116, 184, 106, 79, 85]. It involves generating a map using sensor measurements and simultaneously localizing the sensors/robot within an environment, often in real time. It has not always been associated with camera sensors. However, over the last 20 years, visual-SLAM (V-SLAM) has been a central research area in robotics and computer vision. These systems are capable of producing 3D reconstructions, specifically on video sequences. They were considered a possible means of reconstructing real roads and localizing images relative to the final model. The features of visual SLAM will be discussed to bring context to why it was used in development throughout this thesis.

Like SfM, SLAM contains three main modules: initialization, tracking, and mapping. Initialization defines the global coordinate system through stereo view reconstruction with a known baseline using the 8-point algorithm [98] or localization relative to a known model in the real world with distinguishable image features [42]. Tracking is done by localizing the camera relative to the initial set of image features and growing the set of features through iterations of feature recognition, matching, and triangulation. 2D-3D feature correspondences are used to track the camera moving in the environment using algorithms like PnP (perspective-n-point) [54]. The 3D structure is estimated in the mapping thread when new, not yet mapped points are observed and triangulated [71].

Additional modules have been added to create stable and accurate 3D models and camera pose tracking. These include relocalization and global map optimization. Relocalization is a process used to find the location of a robot or sensor when tracking has been lost. In practice, the visibility of features can vary due to changes in lighting or motion blurring. Relocalization will allow the tracking algorithm to query a large portion of the map against the features in the image rather than local features typically used during tracking. Generally, there is a cumulative error in the map due to inaccuracies in the camera itself (camera parameters), the detail the camera is capable of capturing (the resolution), and the

features being matched against (feature mismatch). Global optimization helps suppress this error. The map is refined to enhance the consistency of the whole map. An example of this includes loop closures. A loop closure occurs when some map segment is revisited, and a drift measurement can be computed. See Figure 2.13 for an example of drift and loop closure. The drift is the distance between where the sensor/robot believes it is due to estimation against the position of where it is located. When the sensor revisits a known location, a measurement of this drift can be computed, and the error throughout its current map estimation can be suppressed. Pose graph optimization and bundle adjustment are other methods used to suppress the map's error and estimated camera poses [171, 88].

Initially, visual-SLAM systems considered the probabilistic optimization or refinement of measurements and predictions using filter-based approaches (Kalman and particle filters). These filters increased the certainty of a robot agent's location and orientation over whatever map representation they used (2D grid-based distances from landmarks, 2D topological maps, or continuous 3D detailed virtual representations of a map) [166]. The earliest example of a monocular V-SLAM system was MonoSLAM [41, 42]. For this system, a map was initialized using an object with known geometry. This known geometry enabled a camera to estimate its pose within the real 3D world. A state vector was used to contain the camera motion and the set of 3D positions from visual feature points triangulated in the images captured by the camera being localized. As the camera moved and observed more of the scene, more feature points were added to this vector. This system used an extended Kalman filter to refine the 3D structure and camera pose contained in the state vector. An issue with this approach was that the dimensionality of the state vector increased as the number of points increased. Eventually, it would get too large to compute 3D features and camera transformations in real-time—the lack of scalability limits this algorithm's applicability to building models for the video-based driving simulator.

The computational costs of localizing the camera against a growing state vector were later solved by putting the localization and mapping into two separate threads. This was done in Parallel Tracking and Mapping (PTAM) [85]. This allowed the camera to be localized in real-time in one thread while the mapping

Figure 2.13: New college sequence, top-down view of 3D reconstruction using ORBSLAM [116]. The top map is the reconstruction before the loop closure match, drawn in blue, the trajectory in green, and the local tracking map in red. The bottom map is the reconstruction following the loop closure.

thread used bundle adjustment to compute an accurate estimate of the 3D models [171]. Many v-SLAM systems today still use this multi-threaded technique. PTAM also introduced key-framing, where selected images containing matching features with a large disparity from another image in the key-frame set would be used for local tracking. The large disparity enabled accurate triangulation. The problem with PTAM, however, is that scale ambiguity (the scale of the coordinate system is unknown) causes scale drift. There are no consistently known distances between frames in a monocular system (a stereo system would have a fixed, known distance between the sensors) [85]. Cameras can be optimized over 7 degrees of freedom for their poses in a pose graph to correct this [156]. ORB-SLAM uses this method along with bundle adjustment and loop-closure detection see Figure 2.13 [116]. ORB-SLAM is one of the few complete implementations of feature-based monocular SLAM. It has since been extended to work with other sensor setups such as stereo and RGBD cameras.

Summary: Bundler was an early implementation of SfM for large datasets, and it has often been treated as baseline performance for many of the other algorithms/pipelines mentioned. Therefore it often displays lower performance in comparison to the other methods mentioned. VisualSfM focused on the speed of reconstruction from Bundler and creating dense models from the SfM output. Its contributions were significant. However, compared to other systems, it has lower accuracy (reprojection error) and point/mesh density. MVE focused on improving reconstructions from images over multiple resolutions, a specific use case that does not generalize as well as some of the other algorithms mentioned. Theia provided an SfM pipeline using multiple algorithms available for SfM; however, their paper does not mention their contributions to increase robustness or 3D point recall. It is unlikely that Theia could compete with other state-of-the-art SfM algorithms. It re-implemented the available algorithms of the time with few novel contributions. This left OpenMVG and COLMAP. Each of these has contributed approaches to increasing robustness and 3D point density. OpenMVG focuses on accuracy over robustness. COLMAP does not make this trade-off. Ruano et al. [136] found that the scenes from COLMAP were sparser (lower recall) while also having lower levels of error (higher precision). COLMAP would be the pipeline of choice for instances where reprojection error over a 3D model

should be low (to reduce visual artefacts). OpenMVG would be preferable if a denser model were required with less concern for the average reprojection error (good accuracy but lower precision). For this reason, COLMAP was used to reconstruct real-world 3D models in the video-based driving simulator developed in this thesis.

Visual SLAM has been shown to perform well on the task of camera localization, sparse and dense mapping [85, 116, 79, 184, 40, 106]. It is often associated with sequential data, like the video sequences used for data collection in this thesis. However, dense 3D models have often been associated with active cameras [79, 184, 40, 106]. This presents an issue for applying SLAM as the solution required for this thesis given standard/accessible equipment has been identified as a requirement for this research. Meanwhile, monocular visual SLAM produces sparse point clouds, like SfM [116]. Monocular SLAM approaches could be used in combination with SfM and MVS. SfM typically struggles to find a good pair of images to begin mapping or localization. However SLAM appears to be more robust and time efficient. This performance difference could be attributed to a mixture of factors including the assumptions that can be made with SLAM. SLAM can assume measurements are made in close temporal proximity and are often closer locally. SfM does not make this assumption. Most SfM implementations assume all images are uniformly equal in probability of being related until all image features have been compared. In this thesis, Combining SLAM and SfM solutions has been applied to overcome instances where COLMAP's SfM components failed to converge to a stable set of 3D correspondences. ORBSLAM was instead used to seed initial poses for COLMAP's SfM reconstruction. This is discussed more in Section D.3.1.

2.4.2 Formalization of 3D Scenes

An artefact of this thesis included the manipulation of standard RGB camera models and projective transformations from world to image space and image to world space. Photo-realistic renderings could be produced using projective texture mapping (PTM) on 3D models [144, 43, 53]. PTM would then be connected to 3D reconstructions and their corresponding camera models. Later, eye track-

ing signals will be measured, and the screen coordinates mapping from the image plane to geometric model rendered in the virtual environments. This section will present a formalization of the mathematics involved in these processes.

In a basic graphics pipeline, three main components are used for rendering; these are 3D points (vertices), illumination sources, and cameras. A 3D object can be composed of vertices where each vertex has a position $\mathbf{p} \in \mathbb{R}^3$, a colour $\mathbf{c} \in \mathbb{N}^3$, and normals $\mathbf{n} \in \mathbb{R}^3$. A 3D model typically comprises many vertices grouped by some rendering system. Usually, vertices are rendered together in groups of 3 vertices to form triangles, otherwise known as faces. Many of these triangles used for one model are referred to as a triangle mesh. The space between vertices will be filled with some surface while being rendered from one side of the face. The colour of each vertex is interpolated over the surface of the face. Therefore, the higher the resolution of the model (more 3D positions), the more colour information can be stored for the model's surface, assuming they are using vertex colours. Normals can be computed by finding the cross product of the vectors between the vertices on the face. The normal is used for lighting calculations. This is discussed later.

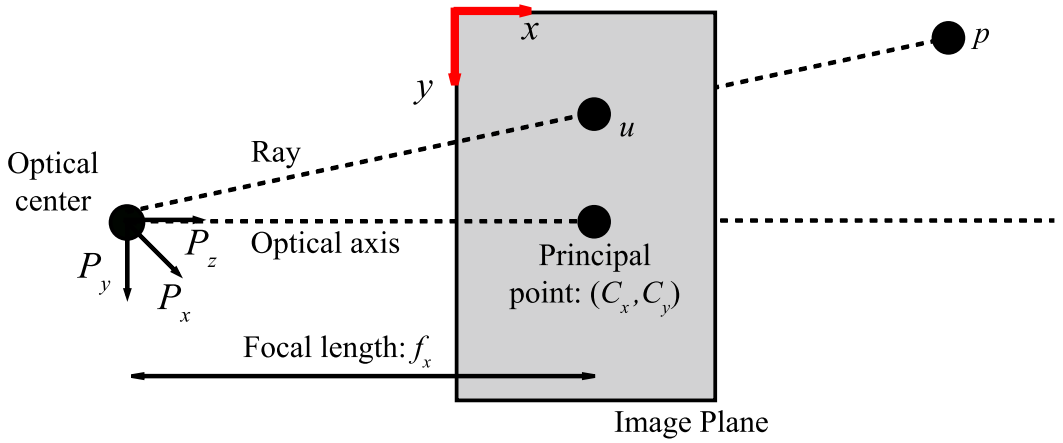


Figure 2.14: Pin-hole camera model highlighting how the point p is observed as a 2D image coordinate u. Also highlighted is the principal point (C_x, C_y) and the focal length f_x

Rendering 3D points with the description above requires transformations from the 3D representation to a 2D viewing plane, see Figure 2.14. Camera models are used to assist in this transformation. This is true for interpreting 3D points in the real world (a computer vision perspective) and 3D points in a graphics pipeline for rendering to a viewport. The standard pin-hole camera model is a good approximation for how cameras can transform 3D points into 2D. In this thesis it will be assumed that the 3D points are in homogeneous form, with an extra dimension w added as a scaling factor, e.g., $\mathbf{p} \in \mathbb{R}^3$ is now $\mathbf{p} \in \mathbb{R}^4$ by appending a 1 to the 3-vector. Let the image space domain be defined as $\Omega \subset \mathbb{N}^2$. The colour image C of colour pixels $\mathbf{c} : \Omega \to \mathbb{N}^3$.

The pin-hole camera model assumes a single point of projection, which all rays, within some viewing frustum, pass through. The distance of the image plane to the pinhole is known as the focal length (f_x, f_y). The pinhole model considers the principal point (c_x, c_y) of the image plane as the point on the image plane where the angle of the rays of light entering the pinhole is 0. A 3D point in line with the camera center and the direction the camera's forward vector is pointing in will be observable at this point on the image plane. This principal point corresponds to the origin of the image plane coordinates. There exists a skew s that identifies the relative alignment between the x and y-axis of the image plane. These axes are not always at a perfect right angle.

In the real world, there are also lens distortion parameters to consider, radial distortion, such as positive radial distortion ("barrel") and negative radial distortion ("pincushion") distortions. However, it will be assumed from this point that images have been undistorted using standard calibration techniques [196]. Camera parameters can be composed into a matrix $\mathbf{K}$, known as the camera intrinsic parameters, Equation 2.1. This matrix can aid in the a transformation for homogeneous points from $\mathbb{P}_3 \to \mathbb{P}_2$, assuming that distortions have been removed from the image plane. It scales the projection onto the image plane in the ideal scenario where f_x and f_y are equal to 1. For a more thorough examination of these parameters, consider [70]. The perspective projection itself can be calculated by adding an additional row and column to this matrix and adding additional clipping parameters for the z-dimension of the viewing frustum [162, 150]. In later chapters, 3D reconstruction tools will be used; see Chapter 4. The video camera

used to collect data will be calibrated. This will mean that the video could be undistorted to remove camera lens distortion and allow the use of the standard camera models for 3D reconstruction and projective texture mapping.

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The camera can then have its pose within a world coordinate system. The above camera intrinsic parameters alone assume the camera center and rotation are aligned with the global coordinate system. To allow for a camera that can have varied positions, consider a rotation matrix $\mathbf{R} \in \mathbf{SO}(3)$ and a translation $\mathbf{t} \in \mathbb{R}^3$. These are all considered in the global reference frame. These two components are then stored in a matrix $\mathbf{P}$ of the form Equation 2.2. These are known as the camera's extrinsic parameters. In Figure 2.14, the optical center's relative position to the world coordinate system is analogous to the translation vector $\mathbf{t}$, and the rotation between the optical axis and the world's coordinate system is analogous to the rotation matrix $\mathbf{R}$.

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbf{SE}(3) \tag{2.2}$$

To compute these projections, the points used must be in homogeneous form. These matrices will allow for transformations between image and world space and vice versa. Vertices in the map at position $\mathbf{p} = [x, y, z]^\intercal$, can be mapped to an image point $\mathbf{u} \in \Omega$ for a given camera, Equation 2.3. Where $\pi(\mathbf{p}) = (x/w, y/w)^\intercal$ is the dehomogenisation of the point. This point $\mathbf{u}$ can be seen in Figure 2.14 where a ray connected to the optical center and the point p intersects the image plane.

$$\mathbf{u} = \pi(\mathbf{KPp}) \tag{2.3}$$

The matrices in Equation 2.3 define a generalization of 3D points $\mathbf{p}$ and poses $\mathbf{P}$, with intrinsic parameters $\mathbf{K}$ and how they relate to image plane coordinates $\mathbf{u}$. Each of these transformations can be visualized in Figure 2.15. This relationship

can be used for rendering image-based representations of the real world after 3D point retrieval. It allows for transformations of 2D image coordinates used for eye tracking to be transformed between the rendering camera $\mathbf{P}_R$ (what the viewer will see) and some image of the real-world $\mathbf{P}_T$, sharing the point $\mathbf{p}$ from a shared model $\mathcal{M}$. The ability to map points between 2D views of the same scene will be used in reconstructing 3D models, projective texture mapping, and eye-tracking work presented in later chapters, see Chapters 4 and 5.

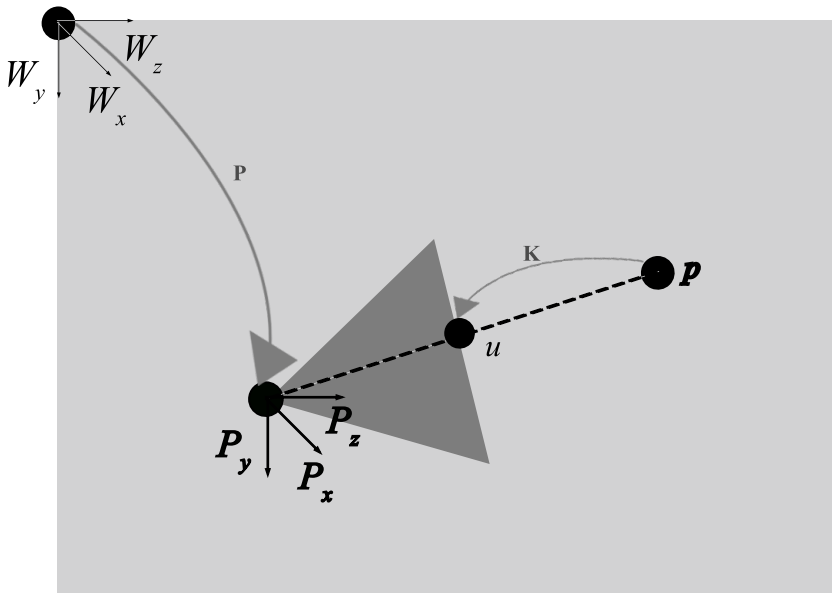


Figure 2.15: Transformations from the world coordinate system $[W_x, W_y, W_z]$ to the camera coordinate system $[P_x, P_y, P_z]$, and the projection of the 3D point p onto the image plane of the camera as the 2D point u.

2.4.3 Modelling Colour on geometry

Texturing will be used to enhance the photo-realism of the final 3D models. Textures are used in place of complex colour and lighting models. Using colour and lighting can aid in the perception of depth. Without colour and lighting, 3D models appear as uniform blobs on a screen, and their edges are difficult to distinguish. Colour and lighting provide cues for depth, such as shadows and varying intensity of light/radiance across the 3D surface, see Figure 2.16. This section discusses the interpretation of colour in computer graphics and 3D reconstructed models. It mentions the assumptions used to build the 3D model and presents the problems with how those colour representations decrease photo-realism. The section finishes with a discussion on what techniques are currently being used to address this problem in 3D view synthesis.

The human eye can distinguish different spectra and interpret them as colour.

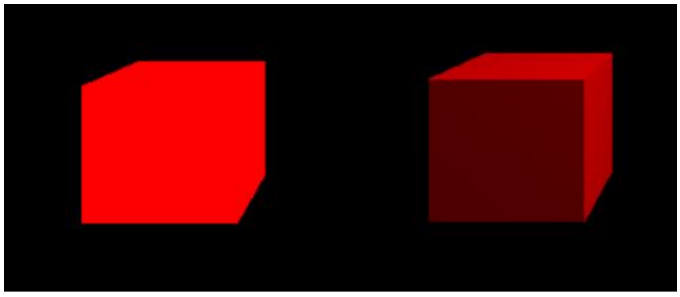


Figure 2.16: Example of a cube with uniform colour and no light on the left. Example of a cube with colour and lighting on the right (diffuse reflection).

It does this using cone cells located at the back of the eye over the retina. There are 6 million cones, with three types that can detect red, blue, and green colours. The correct combination of red, green, and blue light can represent most colours. Typically, in a graphics or computer vision system, three values associated with red, green, and blue represent most other colours (there are others, but an exhaustive list may be going off-topic). Lighting can affect the final appearance of these colours. Depending on the type of surface material, the light can cause ambient, diffuse, and specular reflections. A Blinn Phong model is a simple example to demonstrate 3D graphics environments [124]. There is also BSDF (Bidirectional scattering distribution function), ray tracing, and physical-based rendering, among others [15, 65, 17]. For simplicity, Blinn Phong will be used for this argument. Ambient light is the light emanating from the object without considering a light source. Diffuse lighting is considered when the light ray hits a point/surface of an object, and equal illumination of the point can be witnessed from all possible viewing directions. Specular light, however, varies its illumination based on the observer's viewing direction. See Figure 2.17 for examples of how these can be visualized. With diffuse and specular light, there is a reflection vector in relation to the light direction vector. Figure 2.17 Left to right are examples of light illumination for ambient, diffuse, and specular lighting, respectively. The rectangle is some surface on a model, and the circle on the surface is where the illumination is concerned. The outward-facing arrows from this point are the radiance of light from that point. The diffuse and specular lighting receives light from some source and radiates light differently in response to it. Radiance

is equal for all directions from the diffuse lighting, and radiance is higher in the direction the surface reflects the light for specular lighting.

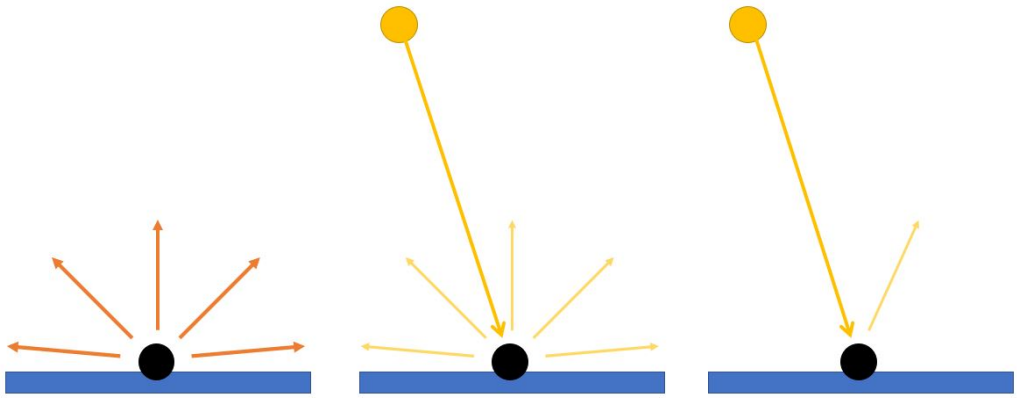


Figure 2.17: Left to right are examples of light illumination for ambient, diffuse, and specular lighting respectively. The rectangle is some surface on a model, the circle on the surface is where the illumination is concerned. The outward facing arrows from this point is the radiance of light from that point. The diffuse and specular lighting receive light from some source and radiate light differently in response to it. Radiance is equal for all directions from the diffuse lighting. Radiance is higher in the direction the surface reflects the light for specular lighting.

Lambertian reflectance is a special case of diffuse reflection in which the material can reflect light in all directions with equal probability (no homogeneity in reflectance). A point appears with equal brightness from all points of view. To accomplish this, the angle between the normal of the point and the light source vector is calculated. Intensities of colour depend on how this angle varies; see Equation 2.4. Here k_d is the contribution of diffuse light to the intensity of the point as determined by the material properties of the point. θ is the angular difference between the normal of the point and the light source angle to that point. I_d is the overall illumination of the point based on the diffuse material and light properties.

$$I_d = k_d \cos(\theta) \tag{2.4}$$

$$I_s = k_s \cos^n(\alpha) \tag{2.5}$$

Specular reflection considers the angle between the reflected light direction vector and the viewing direction vector of the observer. While this angle is fine, it provides greater illumination than the equivalent diffuse source, resulting in a highlight on the surface of rendered objects. In Equation 2.5, α is the angular difference between the reflection vector of light and the view vector. k_s is the contribution of specular lighting according to the point's material properties, and n is a shininess parameter to describe light distribution when reflected off the object's surface. It is the overall illumination of the point based on the specular material, light, and viewing angle properties.

Often the assumption made in 3D modelling is that surfaces are Lambertian. In the real world, there are many cases where this is not true. It is a fair assumption to allow for the recovery of some 3D points and has often proved successful in producing geometrically accurate 3D models [102]. During reconstruction, 3D surfaces' colour and depth are often a weighted average of observations. This can lead to a loss of detail, and hence, a loss in photo-realism [39]. These methods reduce noise in the reconstructed geometry but can lead to over-smoothing of the vertex colours. This approach is employed in many state-of-the-art systems [184, 79]. However, some methods have been developed for estimating light sources in the scene and, in turn, simulating specular reflections [185]. Some techniques reduce over smoothing by optimizing local camera pose photometrically by matching surface colours (the rendering of the 3D model) to image colours (the equivalent image used for 3D reconstruction) [200]. These optimizations have since been expanded upon in [7, 76]. Most static solutions do not change their appearance with changing views of the rendering camera. They are based on the Lambertian surface assumption used for the reconstruction.

Light field estimation has been a popular approach to view-dependent rendering on 3D surfaces. However, these methods have since been dominated by data-driven approaches to learning the view-dependent effects of a model. A pop-

ular method introduced recently was NeRF (neural radiance fields) [111]. This approach used many training images of the same 3D model, reconstructed from the same set of views using COLMAP [139], to estimate the projection of image colours onto points on the model ray-traced back to some query camera pose. The pose to be estimated is provided as input of the form location $p \in \mathbb{R}^3$ and the pose's viewing direction (ϕ, θ). This approach relates to the lighting concepts described above and a neural network/perceptron. The approach estimates the light vectors radiating from the 3D points without prior knowledge of the light sources. The network architecture was a multi-layer perceptron. Following training with colour images and depth maps traced from projections of all the views of a single 3D model, the perceptron could estimate each pixel of a queried pose relative to this 3D model. It provides a corresponding colour and depth value per pixel of the estimated viewing camera (RGB and σ). This provided promising results, including reflections, shadows, and other physical-based rendering lighting effects. However, in later comparisons, it was found that NeRF would only perform well when the synthesized images were within proximity of the training dataset [195]. Other competitive approaches to view synthesis allow for much larger deviations from the original training set camera path but still use similar premises on training with light vectors [132, 133]. The reasons for their performance boost include factors such as an original over-parameterization used in NeRF (using an input vector of degree 5 instead of 4) and a more considerable emphasis on training with image features rather than the raw colour information itself.

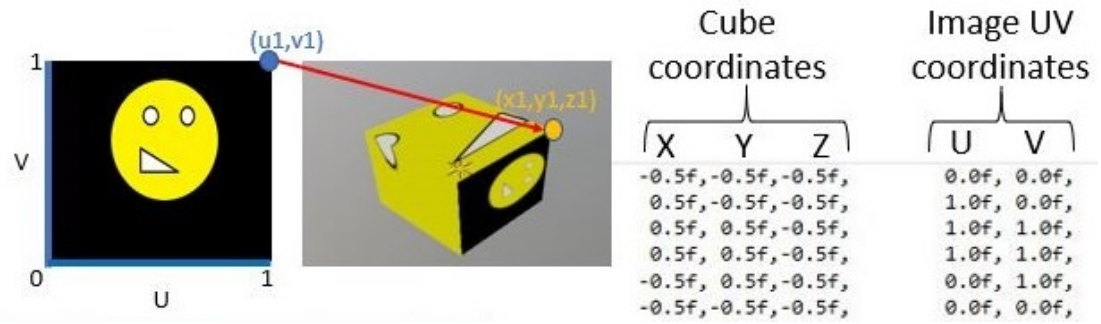
Summary: Colour and light can be challenging to simulate through physical properties. Although the properties of light are currently well known, computers (processors and graphics cards) can sometimes struggle with the computation necessary to calculate how light should reflect off each surface from the source until entry to the viewing camera. Unreal engine boasts the ability to host multiple ray tracing functionalities. However, it often requires limits on the light distance and bounce (number of times to track the reflectance of light on surfaces) [101]. NeRF used data-driven approach has been taken to estimate how surfaces should be rendered when observed from a smaller set of perspectives (a subset of all possible images of that surface). Although the available methods for scene recon-

struction and view synthesis have reached a high level of photo-realism, they can be challenging to include in a real-time system. Data driver approaches, including NeRF [111], often take several hours to compute an appropriate rendering for a set path of perspectives/views. The approach taken in this thesis expands the use of projective texture mapping, a rendering approach to 3D models that will be discussed more in the next section. This approach is taken because it allows for a high levels of photo-realism with the ability to render graphics at a high framerate. However, this will come at the expense of some visual artefacts and only being able to select from a discrete set of images for texturing rather than having the continuous lighting model provided by NeRF-like models.

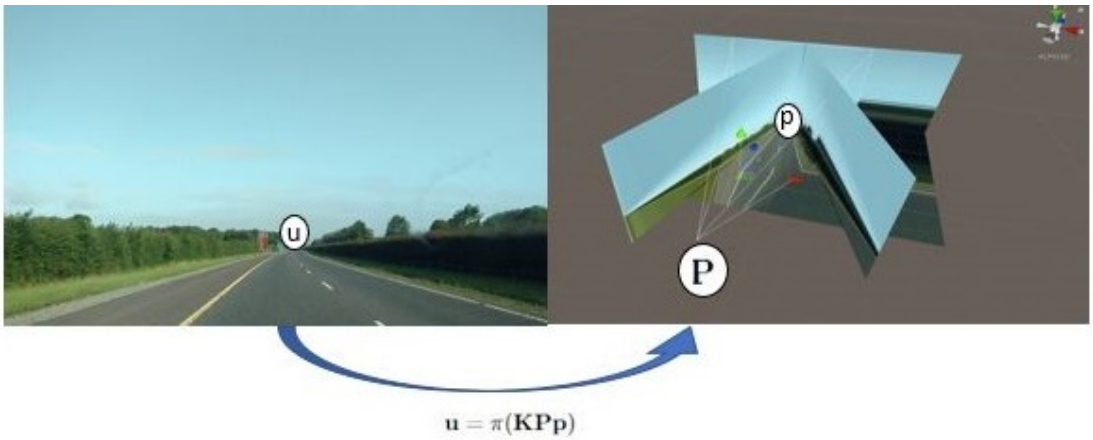
2.4.4 Introduction to Projective Texture Mapping

In general, texture mapping involves matching the vertices of an object to texels (pixels) of an image. This process can be difficult to calculate for intricate 3D objects. An example of a 2D image texture on a 3D object can be seen in Figure 2.18a. 2D image coordinates are matched to the 3D coordinates of a cube. Projective texture mapping allows a 2D image to be projected onto a surface, defined by the vertices within a viewing frustum, described by a projection matrix [144, 53]. This can be seen in Figure 2.18b where a 2D image coordinate $\mathbf{u}$ is mapped to a 3D model at point $\mathbf{p}$. In the 3D model, the bottom plane represents the road. The plane behind the model is a surface on which objects appearing at a distance can be rendered. The two intersecting planes assist with capturing (roughly) the geometry of the road boundary and hedgerows. Other models could simulate vanishing points on the road (parallel planes). However, this model is presented to show an approximation of the projection while limiting the Z-coordinates (distance or depth from the virtual projector) required for the model. This uses a pose $\mathbf{P}$ and a projection matrix $\mathbf{K}$ to simulate a projector. If perspective projection were used to map textures onto a set of surfaces that approximate the geometry captured in a real image, then it would be possible to render the expected perspective changes seen in the real world while rendering in this fashion.

These equations are similar/identical to Equation 2.3 used for rendering mod-



(a) Manual Mapping



(b) Projective mapping

Figure 2.18: Texturing examples: (a): Manual texture mapping from 3D coordinates to (u, v) image coordinates. (b): projective texturing onto planes from **P**.

els in 3D. PTM uses the same transformations discussed previously to map 3D points to any 2D image, instead of the 2D image plane for a rendering camera. These equations can be used for both image plane calculations during rendering and texturing of 3D objects. When colour information is required by the renderer, instead of using an attached colour value within the vertex, an additional lookup is done where some projector (using a specific transformation and camera parameters) is compared to the 3D point and a mapping is calculated between the point and an associated 2D image.

To formally clarify, the rendering camera's extrinsic and intrinsic parameters will be referred to as $\mathbf{P}_R$ and $\mathbf{K}_R$ respectively. Any instance of a projector's extrinsic and intrinsic parameters used for PTM will be referred to as $\mathbf{P}_T$ and $\mathbf{K}_T$ respectively, see Figure 2.19. While rendering 3D objects in this way the texture is connected to the geometric relationship between the 3D surfaces and some matrix representing a projector. Rendering 3D surfaces not encompassed by this projector's projection will have to receive colour information in some other way.

PTM expanded from the basic projection of colour information on arbitrary surfaces, as was previously highlighted by Everitt and Segal for projection and shadow mapping [53, 144]. Later view-dependent texture mapping was coined. View-dependent texture mapping allows for a range of dynamically rendered surfaces that are view-dependent, acknowledging that real world surfaces can often not be Lambertian in nature [43]. This uses projective texture mapping as a method of placing an image or image mosaic onto many surfaces and updating which texture is used based on the transformation of the rendering camera. The extrinsic parameters and image used for texturing are required for this procedure. View-dependent texture mapping has been used in 3D modelling for many years. There is often an emphasis on how to select the appropriate view/image and methods to reduce noise caused by the re-projection error [43, 174, 179]. Chen et al., has published a series of papers in the area of view-dependent texture mapping and has highlighted its real-time and dynamic capabilities while applied to 3D reconstructed models as well as its use in virtual reality [25, 23, 24, 27, 26, 28] [1].

[1] These papers were published in parallel to the time that works in this thesis were published.

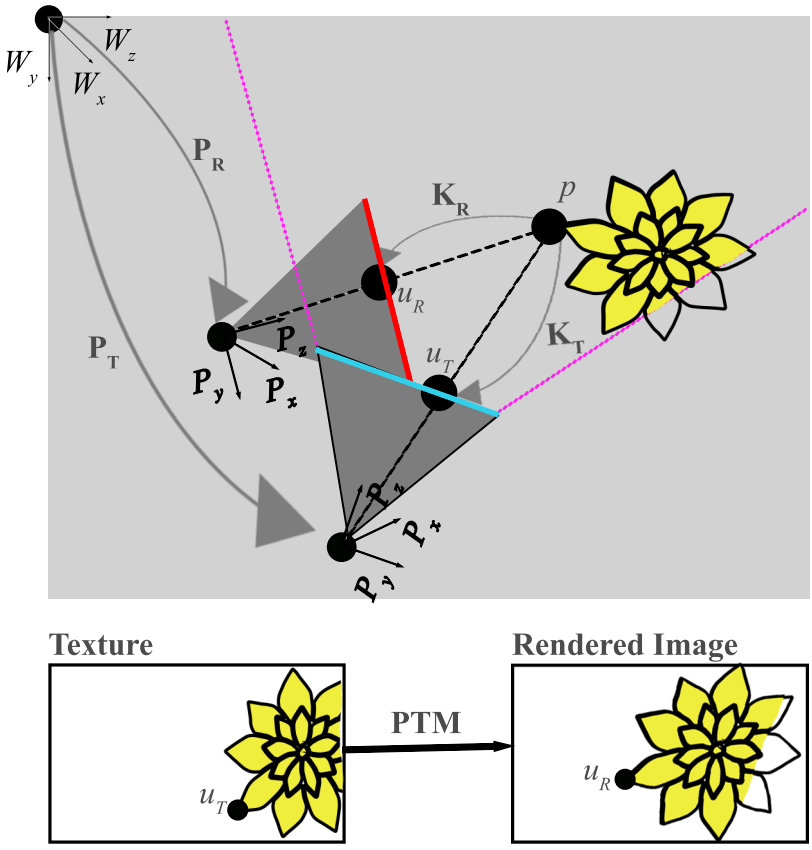


Figure 2.19: Transformations from the world coordinate system $[\mathbf{W_x}, \mathbf{W_y}, \mathbf{W_z}]$ to the rendering camera coordinate system $[\mathbf{P_x}, \mathbf{P_y}, \mathbf{P_z}]$, indicated through the transformation $\mathbf{P_R}$. The projection of the 3D point p onto the image plane of the camera as the 2D point u_R with the transformation $\mathbf{K_R}$. Similarly there are transformations from the world coordinate system to the projector's coordinate system with $\mathbf{P_T}$ and the projection of the point p onto the image plane of the projector as a 2D point u_T using the transformation $\mathbf{K_T}$. Below, the texture on the left image is mapped onto the surfaces (the flower) in front of the projector. The rendered image shows a partially textured object.

This thesis implements and demonstrates PTM in combination with 3D reconstruction techniques including SLAM and SfM. While other researchers previously focused on selecting appropriate views for view-dependent texture mapping, this thesis will focus on the robust application of PTM to real environments. This will include solving the following challenges:

1. Reverse projection: when using PTM surfaces in front and behind the projector and textured. While rendering an image that can see surfaces in both of these directions, this produces visual artefacts.

2. Projection onto surfaces that are not front-facing to the projector.

3. The removal of foreground objects captured in the image used for texturing but not present in the final 3D model.

Challenges one and two have previously been identified by Everitt [53] in their white paper and there have been various solutions presented in the past. The third challenge was identified in the course of the research presented in this thesis. This is discussed further in the next section on background subtraction and inpainting. The solutions to challenges one and two are presented in Chapter 4. The solutions to challenge three is addressed in Chapter 5.

2.4.5 Background Subtraction/Inpainting for Foreground and Background Objects

Inpainting is the process of fixing damaged images through a filling process using information from neighbouring pixels [6, 165]. Background subtraction, however, is the process of highlighting dynamic foreground objects in a scene, given a static capturing camera. The static capturing camera is not always a requirement but most approaches consider very little movement. Background subtraction algorithms often measure the rate of change of pixel intensities over time and set a threshold based on a rate of change commonly associated with moving objects. These two techniques can be paired to aid in removing foreground objects from video, see Figure 2.20. The left image contains the road with cars present. The centre image highlights cars and labels the regions as φ_0, φ_1. The right image

shows that these regions are replaced with the appropriate background colours. It is desired to reconstruct real roads for use in a driving simulator. Projective texture mapping (PTM) takes images and projects their appearance onto a 3D surface. The 3D surface should contain the same geometry as the items captured in the image for photo-realistic PTM. 3D reconstruction algorithms, however, often only reconstruct static objects. Dynamic objects entering the image will not match the 3D model during projective texture mapping, resulting in visual artefacts. For this reason, the above methods (inpainting and background subtraction) will aid in synthesizing photo-realistic views of real roads by aiding in the removal of foreground objects such as vehicles and pedestrians.

Figure 2.20: Inpainting example of the Moyglare Road.

Early inpainting algorithms were only capable of repairing small regions of images [6, 165]. There are some issues these early inpainting algorithms do not satisfy. Over more extensive regions of an image, these algorithms perform poorly, often resulting in visual artefacts. These artefacts include blurring and image bleeding due to the inpainted area occupying an image boundary (a part of the image with significant changes in image colour). Averaging the neighbouring patch pixels would result in the colours at the boundary flooding into the patch. Recently methods of inpainting have evolved, most notably using Convolutional Neural Networks (CNNs). It is possible to train CNNs to develop a contextual understanding of an image and inpaint based on the context, like humans [121]. The results of this form of inpainting are impressive. However, in some cases, inpainted regions appear blurred. More recently, improvements have been made by [177]; these appear to provide more compelling inpainted images.

To solve image inpainting in videos, early research papers focused on optical flow/fluid dynamics to aid in the prediction of missing regions within videos [6].

This would be a rational choice given that most videos travelling in a consistent trajectory would contain surfaces and features that could be tracked frame to frame. The inpainting could be done based on visual consistency between video frames rather than solely relying on neighbouring pixels within a single frame. More recently, deep networks have been used to predict the propagation of pixels, and flow completion [192, 75]. Some of these networks have achieved impressive results on the DAVIS [125] and YouTubeVOS [191] datasets.

Background subtraction would be among the methods of choice for enabling the labelling of regions to be inpainted. Background modelling is typically used to separate foreground from background objects for inpainting. For a taxonomy of how these models are initialized, see [11]. Typically, these methods assume a static camera. The background remains static, allowing a probabilistic model to be computed based on the assumption that foreground objects will move and the background will not [9, 92, 91]. The best results among these have been seen in neural network approaches [104], as compared by [11].

Given that many of the background subtraction algorithms rely on static cameras or cameras with slight movement, it is likely that another approach may be required for labelling moving/dynamic objects within the scene. Semantic segmentation has been a popular method for labelling extensive image collections according to the pixel's relative semantic meaning. For example, all the pixels that occupy the parts of a car are labelled blue, and those that occupy the road are grey [1]. Since 2012 [87, 74, 151] deep neural networks have high accuracy with these types of tasks, especially in instances where large datasets have been provided [44, 63, 96]. This may be a viable option for labelling vehicles and pedestrians, given the large number of frames that require labelling and the dynamic motion the camera will undergo.

Summary: Many popular video inpainting methods rely on time and 2D/image plane space. The solution proposed in this thesis attempts to fuse the observations from many images recorded at different times and organise them spatially in a 3D point cloud. The DVI paper attempts to solve the problem of removing transient traffic objects using LIDAR-based measurements of streets together with a fused 3D point cloud [95]. A similar idea to our proposed solution was

[1] This is for visualization.

published after we used multiple videos for inpainting in the 2D case [32, 95]. It was also slightly before we expanded our solution to the 3D case, see Chapter 5. However, we used a 3D model built using only images with a structure from motion algorithm. Background subtraction has been used over the years to separate foreground and background objects by measuring large deviations of pixel intensities from the mean. This would be useful with a static or panning camera. The solution required for this thesis's use case must be robust to significant changes in the camera position. For this reason, semantic segmentation was considered given its high accuracy and the relation between particular dynamically moving objects often within the scene (other vehicles and pedestrians). For best results in a single task, whether it be segmentation, inpainting, or background subtraction, it appears that deep neural networks are dominating the field. This may be because of their ability to connect image features and associate them with objects to develop a contextual awareness of the scene. Deep neural networks will likely, if not already, dominate in terms of video [21]. The connection between these individual functions and a good understanding of camera projective geometry should allow a system to approximate a photo-realistic scene and manipulate it to the point that the scene only contains features that the system designer selects. Reconstruction algorithms allow foreground object removal, which is quite common. Examples can be seen in systems such as COLMAP [139]. They purposely mask parts of their images for scene reconstruction, so they do not use unwanted objects for their camera localization during feature extraction. This allows the algorithm to localize based only on the background.

2.5 Concluding remarks

This chapter began with a review of driving safety reports, crash reports [170, 158, 167] and driver attention experiments [84, 175, 69, 103, 107]. It was found that eye-tracking was a key factor in determining allocation of driver attention [90, 103, 157, 22, 127, 128, 73]. Experiments carried out on real roads have been informative but can require long term trials to gather enough data [84]. While inducing distraction on real roads, this can pose a potential danger to the drivers [66]. Driving simulation appears to be a safe alternative that gives the

experimenter full control over the environment [157, 107, 103, 20]. It has been gathered that user experiments should be completed in driving simulators and attention should be measured using eye-tracking. This research is reflected in Chapter 3.

A review of driving simulations highlighted their evolution, from analogue systems using projectors and video footage of real roads [77], to cameras over a scale model of a road [182]. Simulators were shifted to digital applications by the 1970's [46, 64]. Although these lacked photo-realism in comparison to their analogue counterparts, over time, this would be compensated for by including light reflection, and advanced 3D modelling [124, 65, 16, 49, 78, 52]. More recently, developments in the area of machine vision and computer graphics have made it possible to merge real world measurements into digital simulators. This can be done using 3D reconstruction [139, 184, 40, 19, 112]. The geometry of these models are convincing but their texture and colour can be less convincing. View-dependent texture and view synthesis is explored to resolve this challenge [43, 111, 195, 132]. It was decided that view-dependent textures mapping using projective texture mapping would be pursued due to its connection to real measurements and its real-time capabilities [53, 144, 43]. The 3D reconstruction process and implementation of projective texture mapping are covered in Chapter 4.

The most concerning challenge with projective texture mapping is explored. The ability to remove dynamic foreground objects from images. This is a predicted issue given structure from motion, and multiple view stereopsis techniques reconstruct geometry for static objects [139, 189, 59, 19]. Artefacts will be produced if images containing moving objects are projected onto these 3D models. It has been decided that dynamic objects will be labelled based on semantic labels highlighted using trained neural networks [74, 151, 87]. The filling procedure used over the regions previously containing the semantic labels will select images near the query image and search over image space for the corresponding area not containing those semantic labels. This was similar to the work done by [95]. The inpainting methods inspired by all the above research are covered in Chapter 5.

Chapter 3

Requirements Gathering Experiments

This chapter reviews eye tracking data obtained from a driver distraction experiment using a video-based driving simulator. The chapter also describes an experiment carried out using a Unity based driving simulator [164]. These experiments assisted with identifying the requirements for the final driving simulator developed. The two experimental approaches, video based and generic 3D models, were compared. This comparison highlighted the benefits and drawbacks of each approach. The video-based driving simulator controls, available to the participants, were limited to acceleration and braking. The system used video playback of a dashboard camera footage to the participants to simulate driving. Eye tracking was recorded for each participant while interacting with the simulator. An automatic method for identifying locations of interest for the drivers was developed. This exploratory experiment highlighted the features required for future experiments. This included image labelling, eye tracking analysis, data processing, and temporal synchronization between video and eye-tracking. Following this pilot study, an experiment was carried out using Unity as the presentation method for a driving simulation. The aim of the experiment was to detect the effect of external distractors from the car (static vs dynamic billboards). The system recorded the driver's reactions to distractors, including their speed control of the vehicle and their eye movement. In this experiment areas of interest over the

images representing the driver's perspective were hand labelled. This allowed a focus to be placed on testing individuals attention to particular events. The first experiment focused on the group behaviour to the events on the road.

3.1 Video-Based Simulation

The data for this experiment was originally collected for the master's thesis by Kaneswaren [81]. The data was re-processed and analysed here for initial requirements gathering for the simulator described in this thesis. Sixty-eight people participated in an experiment in which dashcam footage was replayed to them. The participants were active observers as they could control the speed at which frames of video were played in front of them, using acceleration and braking pedals. The roads used for testing were the L2007 road for 3640 frames of video, see Figure 3.1 (Straffan Rd, Greenfield, Maynooth, Co. Kildare, Ireland), and a segment of the M3 motorway for 10,000 frames of video, see Figure 3.2 (Co. Kildare, Ireland). Although the two roads state the duration of each journey is seven and three minutes long respectively, this may vary depending on how fast the participant chose to drive in the simulation.

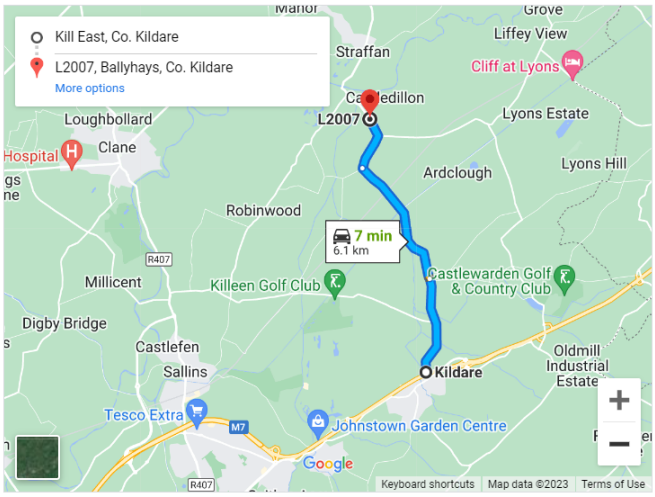


Figure 3.1: Northbound L2007 route from the M7 used for rural driving in video based driving simulator - Screenshot from Google Maps [67]

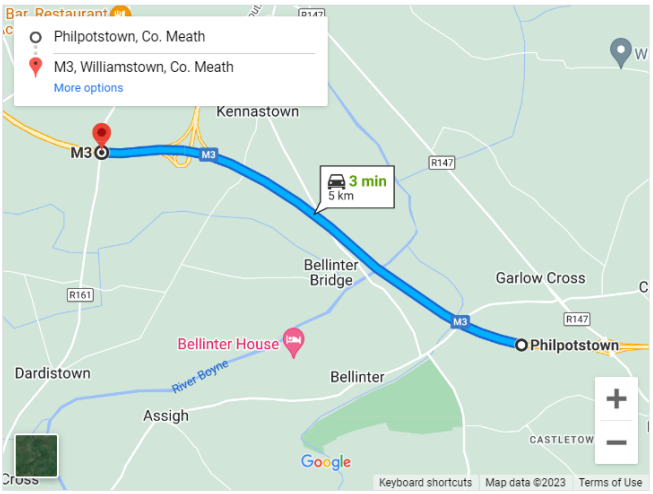


Figure 3.2: M3 route from exit 7 to exit 8 used for motorway driving in video based driving simulator - Screenshot from Google Maps [67]

The eye tracking data was recorded on a Tobii 1750 eye tracker, which was built into a 43cm screen. This screen was accompanied by two other 43cm screens to the left and right of the eye tracker, each with resolutions of 1280×1024 pixels, see Figure 3.3. The three screens were used to give the driver a wider field of view and add to the immersion. The eye tracking was recorded at a rate of 30Hz. The video frames were timestamped and recorded with timestamped eye-gaze coordinates. These were later synchronized offline, and eye-gaze was matched to the relevant frames of video. This enabled cross-participant analysis. Given each frame of video and the corresponding eye gaze positions; an algorithm was developed to automatically identify areas of interest. A temporal comparison, highlighted areas where attention was taken away from the road or dashboard. The identification of object competing for attention on the side of the road were identified. This pilot study showed that participants could be tested while participating in video-based driving simulation and highlighted the requirement for increased immersion with this setup.

3.1.1 Method

Participants were seated approximately 60cm from the three monitors and were asked to observe a calibration video before beginning the experiment. This cal-

Figure 3.3: Tobii 1750 eye tracking system using three monitors.

ibration included a short video sequence of a calibration dot moving across the screen in a zigzag figure of eight pattern. This calibration was used to calibrate the eye tracker and synchronize the eye tracking signal to the frames of video being played back in front of the participants. The video then proceeded to show the R406 regional road video sequence. Following the presentation of the R406 regional road, there was a second calibration zigzag eight pattern, and finally, the M3 motorway sequence was followed by a final calibration sequence. Participants controlled the speed of playback and the apparent speed of the vehicle by adjusting the accelerator and braking pedals. There was no steering control available to the participants, but there was a wheel they could hold during the experiment. Participants were thanked for their time, and data were stored for later analysis.

3.1.2 Data Processing & Analysis

Eye gaze data was matched to video data using timestamps and a third-party time synchronizing software developed by [81]. Video data and eye-gaze data were each saved to their files. The video data included the timestamps and frame number from the video used for this experiment. The eye-gaze data contained the x and y coordinates of the eye on the desktop screens, as well as the global timestamps to synchronize to the video data. There was a third file containing the participant control input for the simulation. This contained the speed at the frames of video being played back and the adjustment to the accelerator controls. The pre-processing carried out by Kaneswaren spared the requirement

to synchronize frames of video to eye gaze during pre-processing [81].

Given raw eye gaze was synchronized to the frames of video presented across all participants a single frame of the video had multiple participant gazes associated with it. An assumption was made that attention could be measured across 68 participants for each frame. A k-means algorithm was used to localize areas within each frame of the video which gathered the most attention across the participants. An initial seed of three cluster coordinates was selected. The first location was in the place of the speedometer, this feature was at a fixed location. The second location was the centre of the screen close to the focus of expansion for driving forward. It was predicted this location would correspond to the centre of the road. The third location was the left side of the road, it was predicted items found at this location may be less related to driving and potential distractors. However, this third focus also detected important events such as reading signs. The positions of the eye gaze for every participant were then plotted alongside these cluster centres. An iterative algorithm where the nearest cluster (per Euclidean distance across the image plane) was assigned to each gaze. When all gazes were assigned to a cluster, a new cluster centre was calculated based on the mean of the coordinates for each gaze within a cluster. These steps were repeated until no new clusters were assigned to each of the gazes (convergence) or the algorithm was repeated some set number of times (10,000). An example of the three clusters can be seen in Figure 3.4. The blue cluster was seeded with distractors to the side of the road, green was related to the centre of the road, and red was associated with the seed over the speedometer. In instances where the final mean of a cluster moved closer to some initial seed point, the labels were swapped to improve consistency across frames of video.

To filter the outliers from the gaze data, an eigenvalue decomposition was used to identify which gaze points were found within two standard deviations [1] of the centre of the cluster. An introduction to the eigenvalue decomposition used can be found in [161]. This approach was taken to omit outliers which have been labelled within the cluster. This variance and co-variance of the eye gaze positions

[1]Using a single standard deviation (65% of the data) omitted too much data and a qualitative assessment highlighted that many of these points were indeed part of the cluster. Using three standard deviations did not add any significant differences to the number of points retained within the cluster.

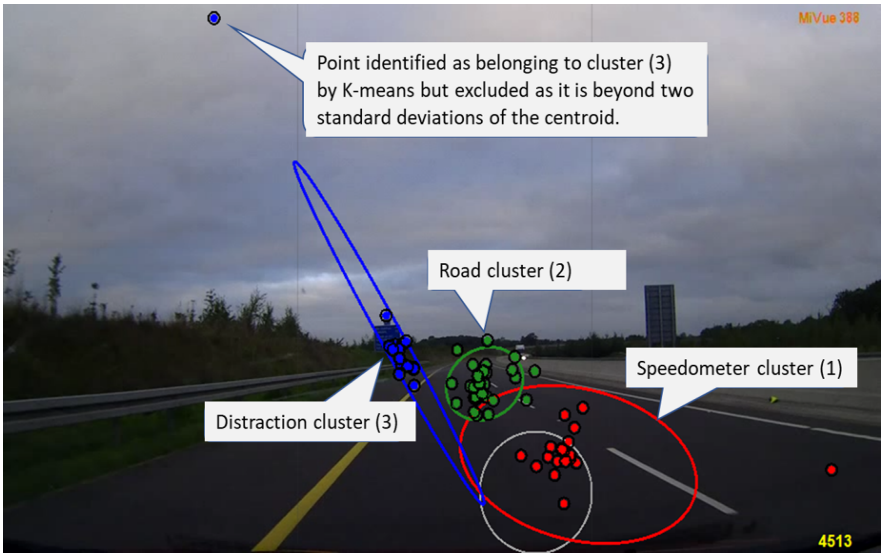


Figure 3.4: Eye gazes split into clusters (red, green, and blue points). The large white circle right of the middle of the image indicates where the speedometer was rendered. The large circles surrounding the eye gazes indicates the decomposed eigenvalues or variance of each cluster.

were calculated using the x/y-cartesian coordinates of the image presented to the participants, see Figure 3.4. These variances and covariances were stored in a covariance matrix, C. The covariance matrix could then be related to a set of eigenvectors, $\vec{v}$, to determine the directions in which the set of points varied the greatest. The degree to which the eigenvector varied was represented by the eigenvalues (there is an implicit relationship between the variance and an eigenvalue). This relationship between all of the above is shown in Equation 3.1.

$$\begin{aligned} C\vec{v} &= \lambda\vec{v} \\ (\lambda I - C)\vec{v} &= 0 \end{aligned} \tag{3.1}$$

Following on from Equation 3.1, and assuming $C \in \mathbb{R}^{2\times 2}$, and the eigenvectors, $\vec{v}$, are non-zero, $(\lambda I - C)$ must be singular. The determinant of a singular matrix is equal to zero. This provides Equation 3.2 and shows how the eigenvalues can be retrieved from a quadratic equation. C_{xx} is the variance of the data in the

x direction (horizontally), C_{yy} is the variance in the y direction (vertically), C_{xy} and C_{yx} are the covariances. Substituting the eigenvalues back into Equation 3.1 gives the corresponding eigenvectors.

$$\det(\lambda I - C) = 0$$
$$\lambda^2 - \lambda(C_{xx} + C_{yy}) + (C_{xx} \times C_{yy}) - (C_{xy} \times C_{xy}) = 0 \quad (3.2)$$

It was predicted that increases in the number of gazes for each cluster may be an indicator of increased visual attention across participants for a particular item on the road. It was also predicted the "distraction cluster" would show more non-driving related distractors, see Figure 3.4. This was because it was seeded to an area away from the centre of the forward roadway and driving controls.

3.1.3 Results

For each cluster, the number of participant eye gazes was recorded for each frame of the video, see Table 3.1. From this data, it was found that participants focused more on the speedometer on the rural roads when compared to the motorway. Using an independent samples z-test between the two treatments it was found that the difference was significant, $p < 0.05$. This was also true for the centre of the road, although the effect size was less significant following the same z-test between treatments, it still reported a $p < 0.05$. While examining the "distractor cluster" it appeared the averages were similar between routes, giving a $p > 0.05$ highlighting the similarity between the two distributions.

There were some distinct peaks corresponding to attention-diverting objects. These peaks were further investigated for the top six contributors for each sequence. For the motorway sequence, the distractor cluster was always associated with road signs, see Figure 3.5. For the rural road, the distractor cluster brought attention to more interesting features, see Figure 3.6. These features included bends in the road Figure 3.6(b), and forks in the road Figure 3.6(c,f). During over-taking scenarios where some participants observed the lane they are in, and

Road Type	Cluster type	Average	Standard Error	Standard Deviation
Rural	Speedometer	6.46	±0.06	3.87
Motorway	Speedometer	3.38	±0.03	3.41
Rural	Center	41.6	±0.15	9.14
Motorway	Center	39.68	±0.11	10.88
Rural	Distractor	2.89	±0.08	4.88
Motorway	Distractor	2.86	±0.05	5.15

Table 3.1: Number of participants associated with each type of cluster for the rural and motorway videos. Each row reports a mean, standard error, and the standard deviation, for the eye gaze count of each participant within a cluster measured over the sequence of frames from each video.

others looked to where they are going, see Figure 3.6(d). Finally, there were instances where clusters split because of approaching vehicles, some participants looked at the vehicle and others observed the gap they were driving into between the vehicle and the edge of the road. The beginning of the divergence between these two groups is shown Figure 3.6(e).

Summary: In this pilot study an algorithm was developed to automatically highlight areas of interest based on eye-gaze behaviour. All participant eye-gaze data for two video sequences were synchronized such that eye-gaze for allotted frames of video could be matched between participants. This pre-processing step to match video and eye-gaze was completed within the work done by Kaneswaren [81]. The clustering algorithm was implemented for this thesis to aid in the analysis of eye-gaze data in a video-based driving simulation. The technique for identifying clusters of participants in each frame of video was K-means and it was filtered using eigenvalue decomposition with a threshold based on the distance of an eye-gaze sample from the cluster centre along the two major eigenvectors of the cluster. There were significant differences in the number of participants observing the speedometer and the centre of the road depending on the type of road they were observing, rural or motorway. The number of participants that observed objects away from either the speedometer or the centre of the road, appeared to be consistent between video sequences. Following an investigation into when this distractor cluster reached a local maximum, it appeared that most peaks corresponded to participants observing road signs on the motorway sequence. In

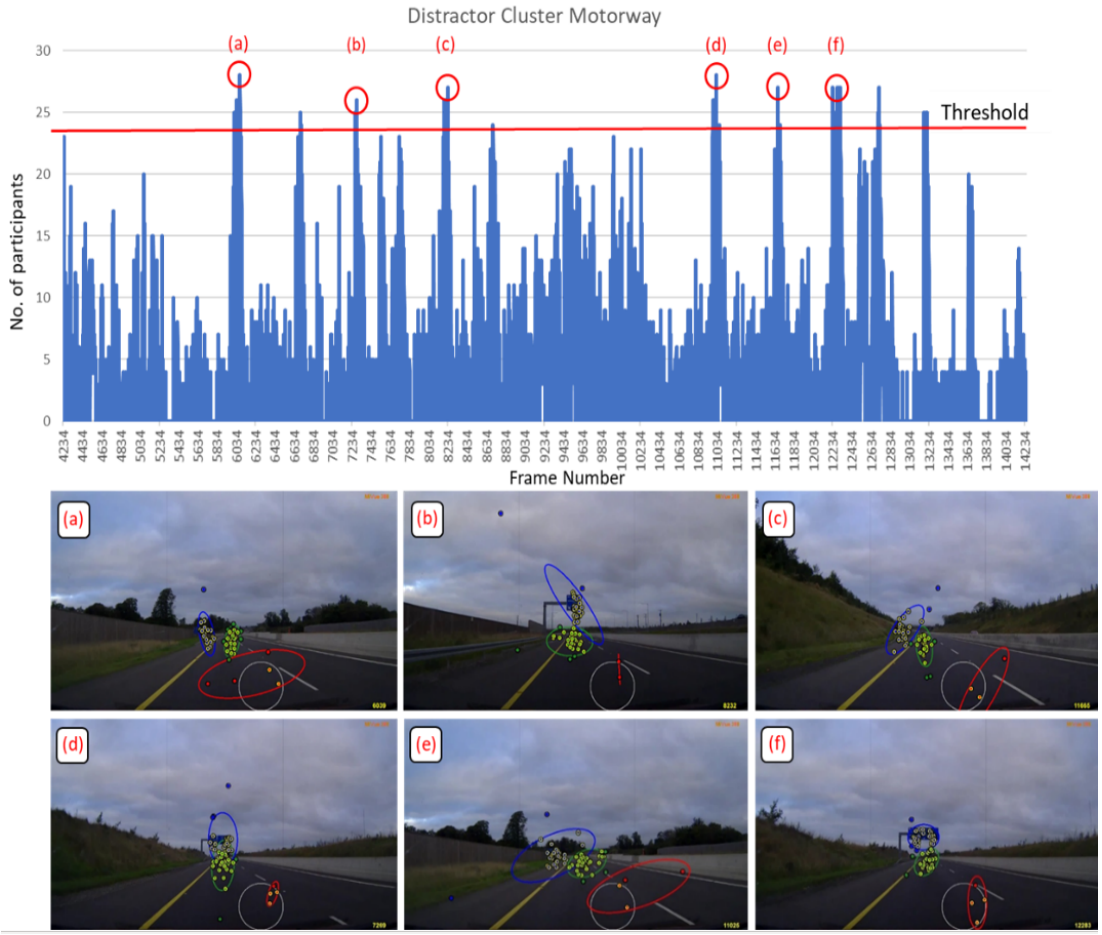


Figure 3.5: Graph (above) highlights the number of participants associated with the distractor cluster. Six frames from the motorway video, (a)-(f), were selected to highlight instances where more than 24 participants appeared to be distracted. These six frames (below) are presented below and the 3 clusters are highlighted, as shown previously in Figure 3.4.

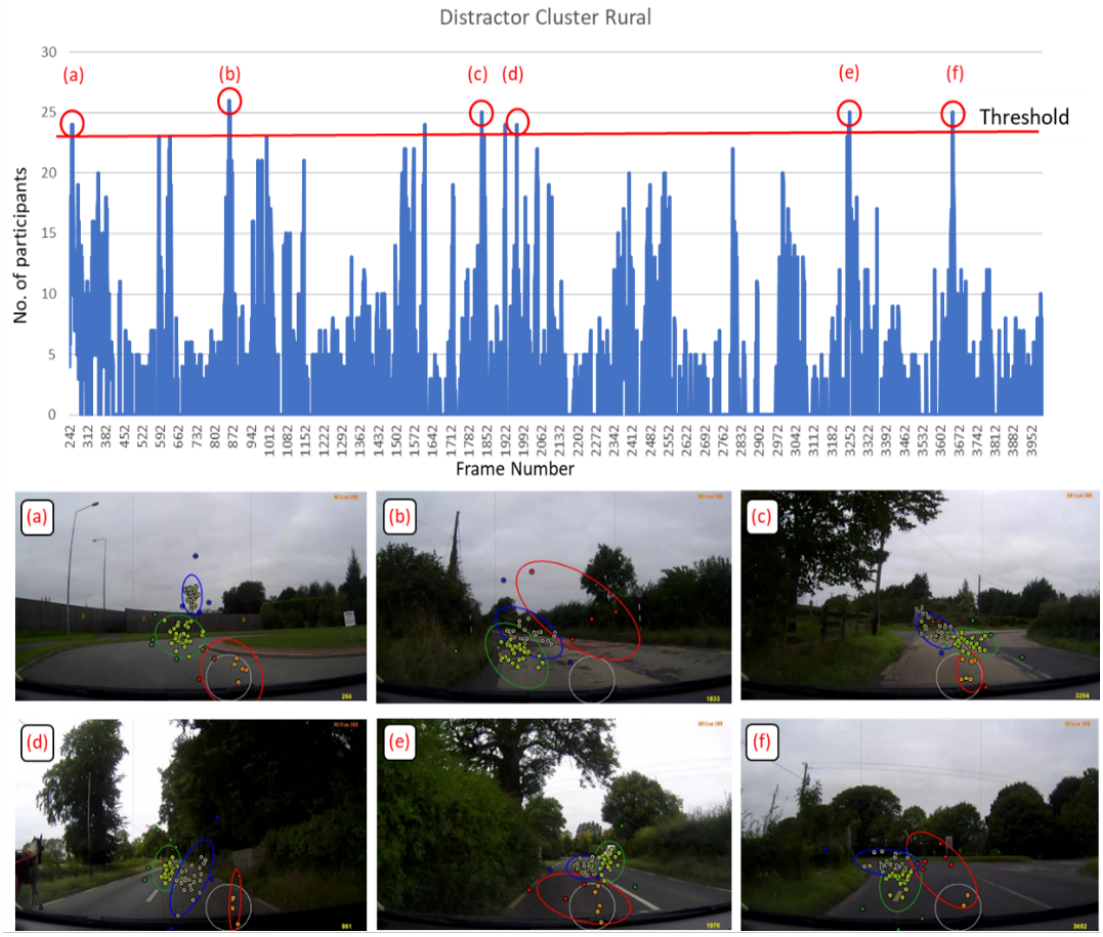


Figure 3.6: Graph (above) highlights the number of participants associated with the distractor cluster. Six frames from the rural video, (a)-(f), were selected to highlight instances where more than 24 participants appeared to be distracted. These six frames (below) are presented below and the 3 clusters are highlighted, as shown previously in Figure 3.4.

the rural sequences, with a more diverse environment, interesting features could be observed. This included forks in the road, varying viewing tactics during vehicle overtaking, and distinct groups who pay more attention to passing vehicles while others look into the gaps they are driving into. Based on this study it was shown that observations of driver behaviour in response to the video datasets could be made using human eye gaze. Observations of attention and distraction for roadside items require a full fixation analysis and possibly labelled frames of video to highlight areas of interest [4, 103, 146]. This study highlighted that participants would engage with video-based simulation. Participants vary their attention across the three monitors based on the environmental cues observed outside the vehicle. Video-based simulation allows for the testing of participants' observation on real roads while also ensuring the safety associated with simulation. A similar study on real roads would expose drivers to the dangers of the distraction. This study had reduced fidelity due to absence of steering associated with real driving.

3.2 Asset-Based Simulation

The previous section examined driver behaviour in a video-based driving simulation. It highlighted important factors to consider for a video-based simulator. This included the requirement to synchronize eye-tracking to the frames of video, and the need to identify areas of interest in the eye-tracking data. However, the study did not consider standard practices in driver distraction or eye tracking experiments. Driver distraction experiments often introduce a distraction stimulus and test the effect of this distractor on the driver. Eye tracking experiments often classify eye-gaze into fixations and saccades, to aid in the measurement of attention, see Appendix A.1 for details on these types of eye movement and how they are acquired. Raw eye gaze was used in the previous study. An assumption was made that groups of drivers would be likely to attend to the same object at one time within a cluster. The study was exploratory and relied on an existing dataset. To gauge the requirements for a driver distraction and/or an eye-tracking experiment an asset-based (using 3D models specifically designed for a computer game or simulation) driving simulator was developed. This was

used to bring the focus onto data collection, testing, and processing. The previous study highlighted the practices required to incorporate video-based driving simulation in a driving experiment. The development of the experimental design, setup, and analysis was done in collaboration with the author and colleagues from the Technological University of Dublin.

This study focused on a method to measure external sources of distraction while driving. This experiment used an asset-based driving simulator with basic 3D depth and lighting developed using the Unity game engine [164]. To measure the attention of participants eye tracking was monitored. It was hypothesised that increased levels of attention would be found in the presence of distractors, and varying levels of distraction might also be measured based on the level of animation of the distractors. Driving performance was examined throughout the experiment, including driving speed and the location and orientation of the participant's vehicle within the virtual world.

3.2.1 Experimental Design

A driving simulator was customised and then used within the gaming development tool Unity [164]. The driving simulation consisted of a straight road and a virtual vehicle the participant/driver could control within this environment. The left side of the road intermittently contained fifty speed limit signs, randomly placed. The speed limits varied between 40km/h, 60km/h, and 80 km/h. Along the right side of the road, there were 50 digital billboards (distractors) placed opposite the speed sign. The billboards and speed signs were randomly sorted but were presented to all participants in the same order. Half of the billboards were dynamic (changing image) and the other half were static. The billboard consisted of a combination of advertising and road messages. Dynamic billboard animations were activated based on the distance of the participant from the billboard, see the transition between top and bottom images on the billboard in Figure 3.7. The same billboard changed its content from a Volvic to a KitKat advertisement based on the position of the vehicle, see left images and the length of the line δ measuring the distance of the vehicle to the billboard. The simulator controls included a Logitech G27 racing wheel and pedals for the participants to

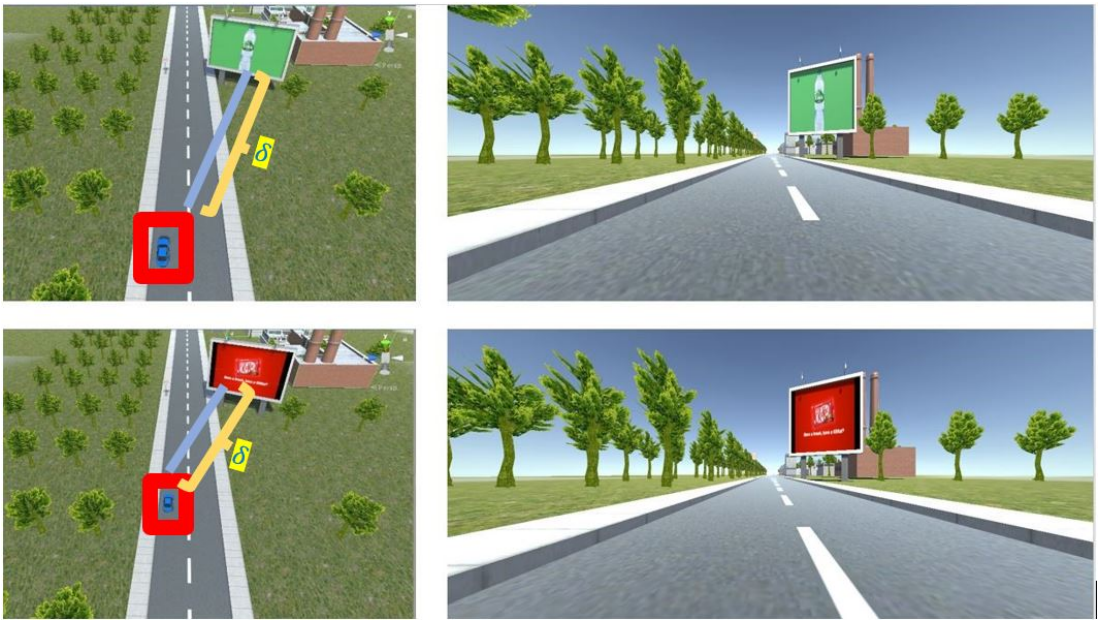


Figure 3.7: An example of a billboard animation given the proximity of the car to the billboard. δ highlights the distance between the vehicle and the billboard.

control the car in the simulator. The steering in the simulator was disabled as the drive was along a straight road and the steering was originally very sensitive to user input. Participants had control of speed and braking using the pedals. The participants were told to obey the rules of the road and observe speed limits.

3.2.2 Data Collection

Participants were seated at 100 cm from the triple head monitor (three 1024 x 768 monitors connected to a Matrox TripleHead2Go converter interface to a computer act as a single 3072×768 monitor) and 70cm from the eye tracker. The dimensions of the screen's resolution and size, and the distance of the participant from the screen were used to calculate the number of degrees the eyes would move about their centre of rotation, see Appendix A.1.3. Eye movement data were recorded with the Tobii X-120 eye tracking system. The sampling rate of this system was 120Hz. Participants had their eye-tracking calibrated and validated before each experiment, and the error was ensured to be kept below the 1.5° of error. Calibration was carried out across three monitors, but the calibration

software treated the three monitors as one large screen. Nine calibration dots were randomly shown at extreme positions across the three monitors.

Initially, it was planned to record the image participants observed for each perspective that each participant was exposed to during the experiment. The eye tracking could then be projected on to the image for analysis. However, the run-time performance of the simulator would not allow for this. This was due to the large amount of data being written to disk continuously. Instead, the pose and position of the rendering camera were recorded such that an image could be rendered after the experiment was complete to allow for matching between eye-gaze and the image the participant was observing. The position was represented in 3D coordinates using three values and the pose was represented using a quaternion represented by 4 values. These were all written to a separate row within a CSV file for each new sample and recorded at a rate of 15Hz. The simulation also recorded events when billboards triggered their animations.

All the sensor measurements and simulation meta-data was synchronized using the lab streaming layer (LSL) [86]. LSL is a system for the unified collection of time series measurements specifically for research-oriented experiments. The eye tracker was connected to the system using a customised python script. The driving simulator meta-data, including camera pose and position and billboard markers, was written to a CSV file locally together with LSL timestamps collected through an LSL interface.

3.2.3 Participants

Eleven healthy adults volunteered for the experiment. The volunteers were post-graduate students and university staff, of which they were all right-handed and there were eight males and three females. The age range was between 22 and 55 years with a mean of 32.3 years. Participants had normal or corrected vision. This experiment was approved by the Maynooth University ethics committee to carry out the experiments on their premises. Written informed consent was obtained before the experiment. After completion of the experiment, it was found that ten of the participant datasets could be used for further analysis. One participant was omitted due to a loss of eye tracking found during the post-experiment

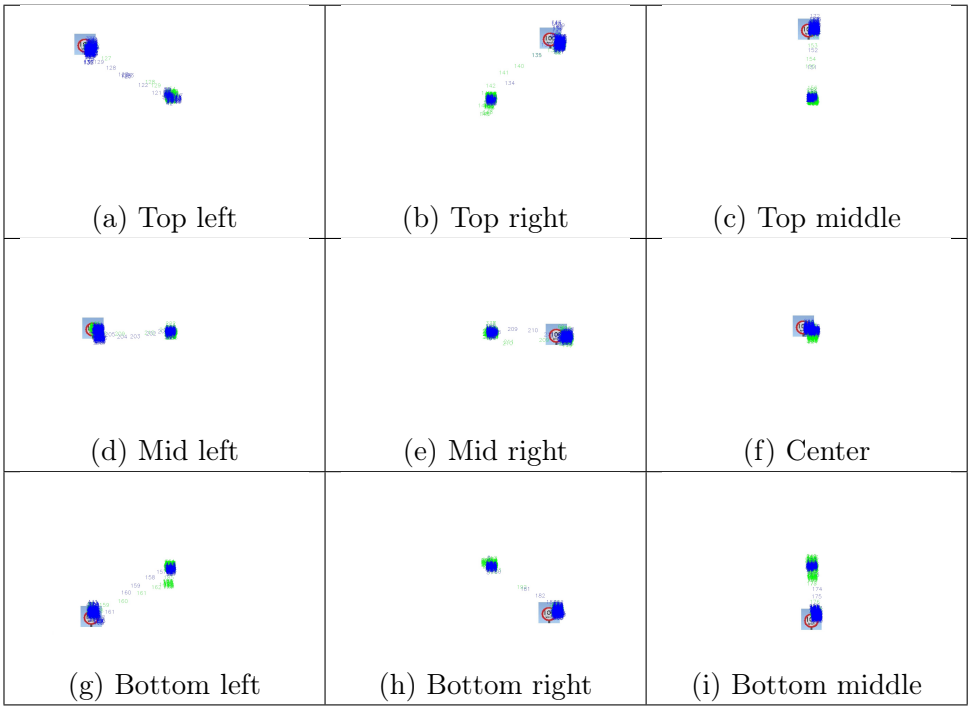
validation.

3.2.4 Data pre-processing

Eye tracking signals contain a variety of artefacts due to noise present during data collection. For example, eye tracking can suffer from loss of tracking or misclassified eye-gaze coordinates caused by head movement. The following section describes the processing of the eye gaze data before analysis.

Before any pre-processing of eye tracking data was carried out, the validation data was queried to ensure the eye-gaze was correctly synchronized to the simulation and the coordinates of the eye-gaze were aligned to the screen coordinates. The validation screen contained nine images of speed signs randomly appearing across different locations on a screen with a white background. The road signs accounted for approximately $\pm 1.5^{\circ}$ of ocular range while presented on the screen. The eye gaze coordinates were plotted over these validation images to verify that the time and space synchronization of the eye-tracking was correct. Figure 3.8 shows the nine validation images with a participant's eye gaze plotted on top of each image. The eye gaze was plotted for one second leading to the image being presented and is highlighted in blue text, and the eye gaze following its presentation is highlighted in green text. The text is a number relating to the n^{th} eye gaze sample. Eye tracking was reviewed following each experiment. If the eye gaze did not overlap with the appearance of the roadsign in these images then that participants data was removed from the final analysis due to tracking loss. This was successfully carried out for all but two of the participants eye-tracking. These participants eye gaze did not overlap with the road signs during the validation.

For the remaining participants, the eye-gaze data was processed to identify times when tracking was lost. This was done by highlighting "NaN" values in the recorded data and eye gaze coordinates outside the screen's boundary. Linear interpolation was used to correct for short periods of tracking loss. The data were converted from units of pixel coordinates to ocular degrees to allow eye velocity measurements and the setting of thresholds. A five-point Gaussian running average filter was used to reduce noise. Fixations and saccades were identified with a velocity threshold of $30^{\circ}/second$ [117]. Samples collected below the given thresh-



(a) Top left (b) Top right (c) Top middle

(d) Mid left (e) Mid right (f) Center

(g) Bottom left (h) Bottom right (i) Bottom middle

Figure 3.8: Each tile is an eye tracking validation image with example eye gaze over-layed. The blue text are the number of samples where the eye was following the appearance of the speed sign. The green text highlights where the eye returned to after the sign disappeared.

old were classified as fixations. Fixations were only retained if the sum of time for neighbouring samples was above 60ms. Fixations with less than 0.5° apart and a time difference less than 75ms apart were merged into a single fixation. All other samples not labelled as fixations were assumed to be saccades. These parameters are standard practice for the equipment used and the area of eye tracking, see Appendix A.1.

The 50 billboards were each used as markers for separate trials. The imagery (point of view), eye tracking, and speed were examined for fifteen to twenty seconds surrounding each of these billboards. This time varied depending on the time it took the participant's vehicle to pass the billboard and speed sign following the onset of some starting distance from the billboard. Each of these instances is referred to as a separate trial. An area of interest (AOI) analysis was implemented to associate eye fixations to items on the road [4, 103]. Within each trial, three rectangular bounding boxes were labelled for each participant's given point of view while passing the billboards; see Figure 3.9. Labels were constructed for each billboard, speedometer, and speed sign for every image visible to the participant during every trial. This data was stored in a CSV file for later analysis in correspondence to the eye tracking data; see Table 3.2. Every ten images were labelled with their respective positions, width, and height. The images between labelled images received interpolated values for position, width, and height and a constant label. This was possible as most of the transformations under these short periods were linear in their difference. Since the vehicle followed the same trajectory for every participant, these values could be translated across participants where the recorded position of the vehicle acted as a key to match these values. The AOI for the billboards was defined as 10mm larger than the corresponding item that appeared on the image plane.

The fixation data was matched to the AOI locations and the sum of the time that was spent on each AOI was calculated in terms of seconds. This was then divided into dynamic and static groups. The dynamic group referred to the amount of time the participants observed these items in the presence of a billboard that included an animation. The static group referred to the number of times participants fixated on an item on the road in the presence of a static billboard. These were highlighted with the columns beginning with the word "static" or "dynamic"

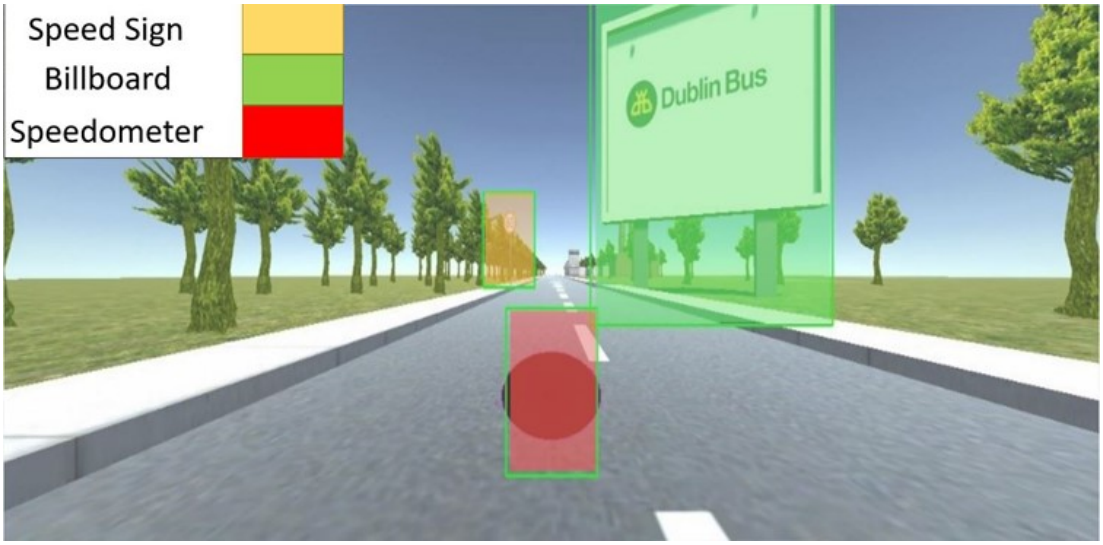


Figure 3.9: Highlighted areas of interest for eye tracking analysis.

AOI labelling					
Image Number	Label	x-position	y-position	Width	Height
1	Speedometer	250	250	50	50
1	Billboard	500	100	100	100
1	Speed sign	50	100	50	50
.	...	.	.	.	.
N	label-M	.	.	.	.

Table 3.2: Labels for AOI labelling

on the first column. The following word in this column indicated what item the participant was observing. “WS” referred to white space, meaning it was a position of the screen not labelled by an AOI. “speedometer”, “billboard”, and “sign” each referred to the times when the participant fixated on the speedometer, billboard, or speed sign respectively. Each other column was numbered, and this referred to a separate participant. It was observed that the participants that did not pass the validation check, participants 3 and 9, had the most instances of fixation and white space crossover. This could be due to participants observing the labelled AOIs, but the eye tracker was mislabelling their coordinates. For this reason these participants were omitted from the analysis. The overall dwell time for each fixation has been plotted for each participant in Figure 3.10. This supports the theory that increased fixation time on white space resulted in less time on other AOIs for participants 3 and 9. For all other participants there is a far greater emphasis on the percentage of attention given to the speedometer, billboard, and road sign, despite these features occupying less space on the screen. For most participants their attention is diverted towards the road sign and speedometer, implying they are following the specified task, "follow the rules of the road". Although the billboard was not related to the specified task it still diverts some of the attention of the participants throughout the experiment. In the next section these results will be explored in more detail.

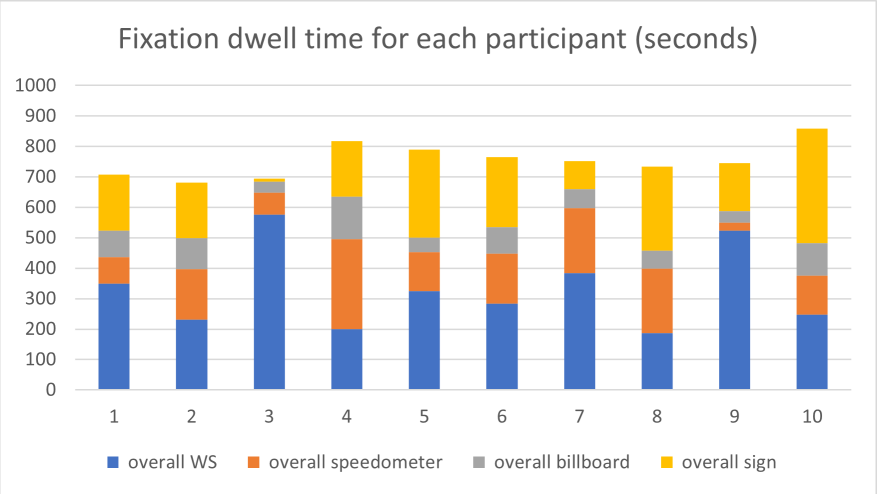


Figure 3.10: Fixations dwell time over each AOI for each participant.

Vehicle position data was sampled from the driver's location and rotation (transform) on the road. Given the transform, it was then possible to calculate the speed the driver was travelling. Assuming a single unit in the space of the simulator world equated to 1 meter in the real world, and the distance between two poses of the participant at time τ_n and τ_{n+1} could be calculated using Equation 3.3. Where S was the speed, P was the pose/transform, τ was the time stamp for each pose, and η was the normalizing factor to convert m/s to km/h. Rotation and steering were not considered given the participant followed a straight trajectory on an even road.

$$S = \left(\frac{||P_{n+1} - P_n||}{(\tau_{n+1} - \tau_n)} \right) \times \eta \tag{3.3}$$

The collection of driving simulator car position data, and eye tracking data were synchronised using the lab streaming layers (LSL) which recorded a global timestamp to enable comparison of these measurements [86]. This allowed for an analysis of driver performance in relation to the stimulus they were exposed to (within the simulator) while driving.

An example of the data for a single participant as they completed the simulator trial can be seen in Figure 3.11. The two graphs show driver speed, and eye-tracking information plotted against distance along the driving route for one participant. Red vertical lines indicated the position of dynamic billboards, green vertical lines corresponded to the position of static billboards. The driver speed graph shows the speed limit in green horizontal bars and driver road speed using a blue continuous line. The eye-tracking plot identified target fixations to areas of interest as a function of road position. Each black triangle represented an observed fixation over a specific AOI.

3.2.5 Analysis

The data collected were analysed to show the driver's overall response over the duration of the experiment. For each of the nine participants, their eye gaze data was represented as fixations over areas of interest (white space, speedometer, billboard, sign). To test for differences between static and dynamic billboards the averages for eye fixation dwell time over the areas were calculated for each

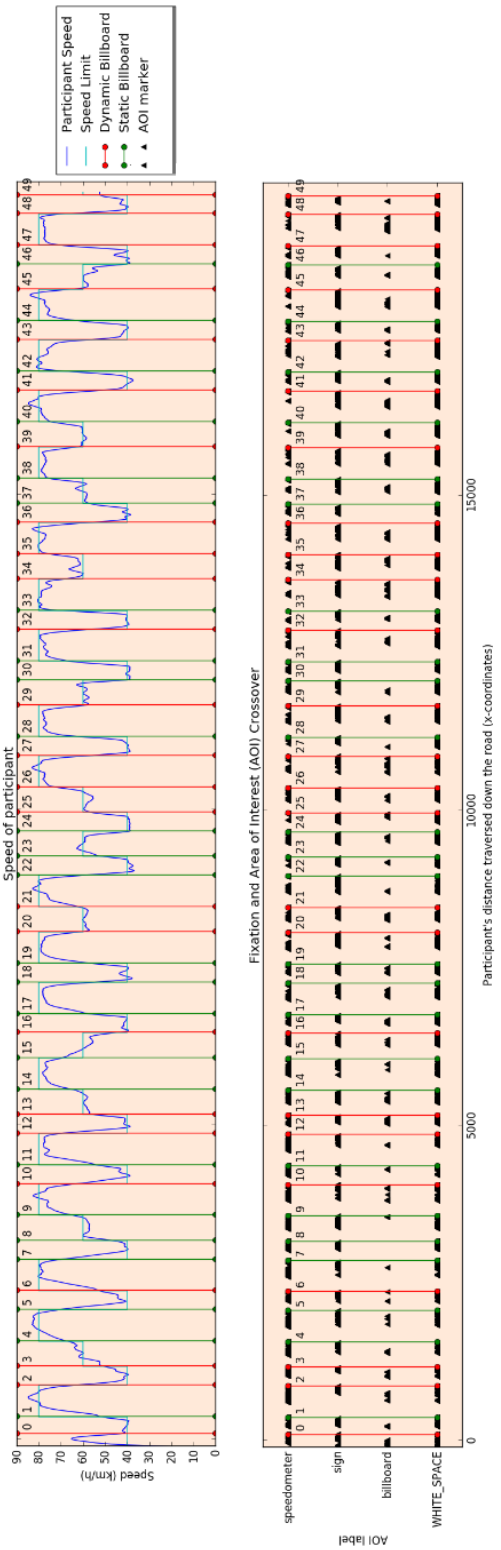


Figure 3.11: Single participant driving controls and eye tracking signals.

event and the difference between these averages was plotted, see Figure 3.12. A paired t-test on each of the average dwell time differences showed that the billboard dwell time differences had a significant effect size, with a p-value < 0.05 and t-value of 3.677. This contradicted the null hypothesis that the average difference of dwell time over the dynamic billboard has either less time or zero difference from the static billboard dwell time.

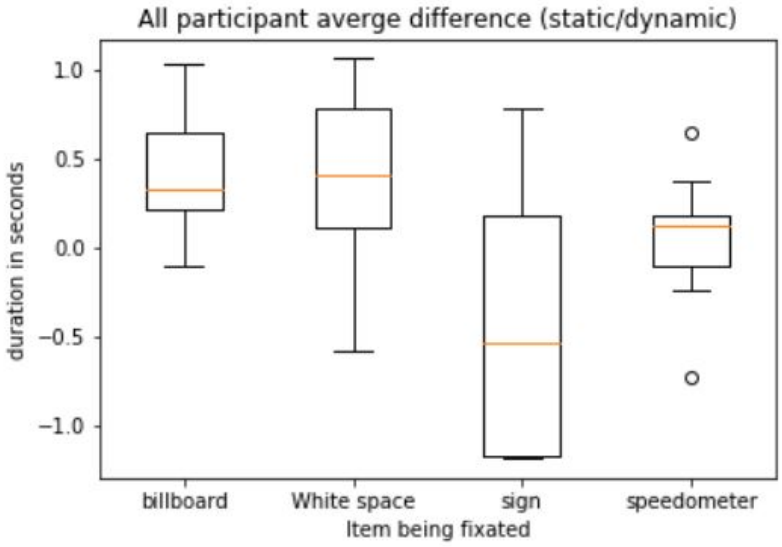


Figure 3.12: Average differences in fixation dwell time between static and dynamic billboards.

The driver controls were represented as the root mean squared error (RMS) between the speed limit and the actual speed the participant travelled at, and the standard deviation of the participant's speed (σ Speed) at each of these intervals. A paired t-test was carried out for each driver performance variable concerning the static and dynamic averages. There was no statistically significant difference for the RMS but there was for the standard deviation size of their speed ($\alpha < 0.05$). The mean of the differences for the standard deviation of the speed was $1.48 km/hr$.

The drivers may struggle to maintain the exact required speed while attending to more distracting features on the road i.e. the dynamic billboard versus the static billboard. These variances in speed and deviations from the prescribed

speed limit has occurred while significant changes in the amount of time participants visually attended to dynamic billboards. However this did not lead to significant deviations from the recommended speed limits. Based on the standard deviations of speed, driver's appear to struggle to maintain consistent speed while distracted by dynamic billboards. Again, this is not yet cause for concern considering $1.48km/hr$ tends not to be flagged as a dangerous variance from the prescribed speed limits. Based on previous studies on driver workload [128], for future studies it would be worthwhile to study the effects of distraction with more diverse surroundings (i.e. traffic, pedestrians, and surrounding buildings) to increase the workload imposed on drivers and further exacerbate the effects of distraction.

Using averaged data across the 9 participants (50 billboards) it suggested that there was an increased fixation time for dynamic billboards over static billboards. The dynamic billboards each required an average additional 0.41 seconds fixation time per short sequence, see Figure 3.12. The speedometer fixations over these periods remained relatively constant. The other two groups (fixations on speed signs and all other spaces) had high variability for fixation dwell times. There was an increase in the variability in driving speed following a dynamic billboard when compared to a static billboard. Driver performance deteriorated marginally from a single dynamic billboard event.

3.2.6 Conclusions/Discussion

This analysis benefitted from having labelled data to inform where attention was focussed throughout the simulation. An understanding of the pose of the participant's virtual vehicle within the simulator allowed for an analysis of the participant's speed and a synchronisation (key indexing) for cross-participant analysis. The position data-informed where the participant was relative to the labelled image planes. This informed what eye tracking timestamps were relevant to the given data period. The inclusion of lab streaming layer software eliminated the requirement of the simulator to synchronize these timestamps itself. The simulator reported simulation metadata including control inputs and orientation and position of the driver to the LSL.

Based on the eye tracking results it appeared that the animated billboards caused an average additional 0.41 seconds fixation time per billboard observation. Previous driving performance studies have highlighted visual complexity of the road and the ability drivers have to drive safely due to inattention [55]. They refer to this as driver workload and it is considered that this operates on a continuous spectrum rather than a binary ability to attend or not. It is possible a more complex environment than what was presented to the drivers in this simulator study could promote a stronger response, whether this would be in response to more complex animations on billboards or a more visually convincing road that the driver observes.

3.3 Concluding Remarks

The video-based eye tracking experiment highlighted multiple features that could be improved. This included the requirement for participants' more realistic driving control within the simulation. The video-playback changes helped give feedback to the participants but were missing a significant factor associated with driving, steering. The study highlighted the importance of labelled video data and the ability to categorize where attention was focused with eye gaze data. There were some observable instances of attention shifts using raw eye gaze and automatic clustering methods. During a qualitative assessment, there were some instances where participants' eye gaze would observe an item on the road, but this did not co-occur. This is similar to the findings of Yarbus while examining people's eye gaze while observing photographs [193]. Labelled data would allow attention to be measured per item rather than the number of participants who observed the item at the same time. Raw, unfiltered eye gaze provided many data points. The eye tracker used in this study operated at 30Hz, but other modern eye trackers can operate at rates as high as 2000Hz [130]. The number of false positives for the clustering approach outlined above could be dramatically reduced by filtering eye tracking data and labelling instances of fixations. Eye tracking research usually associates fixations with an allocation of overt visual attention [50]. This approach was taken for the following experiment using asset-based simulation.

The asset-based simulation contributed to the requirements gathering process. The simulation allowed for the inclusion of specific scenarios in the driving sequence presented to the drivers. In this case, distraction in response to dynamic and static digital billboards was tested. This was done by repeatedly presenting dynamic or static billboards to the participants in the presence of a speed sign. The main driving task given to participants was to obey road safety laws. By extension of that instruction, the participants assumed they should follow the speed limits dictated by the speed signs. In a video-based simulation, it would not be easy to replicate this experiment without considerable video editing, including possible scene augmentation. The asset-based simulation allowed for constructing an environment that could replicate the same scenario for each driver. During experimental design, multiple valuable features were included; the tasks for validating eye tracking data and the temporal synchronization of sensors and simulator metadata through the lab streaming layer (LSL). This ensured that all data was valid and could be measured in combination to aid in the analysis of participant attention. Eye tracking was processed and fixations were labelled. This reduced the number of data points considered during analysis. These fixations were measured concerning specific locations within the participant's perspective using AOIs. AOIs were labelled by hand for each participant who passed a billboard and speed sign. These labels provided a means to analyse driver attention. It was noted that there were significant differences, 0.41 seconds, in the amount of time spent looking at dynamic billboards compared to static billboards. It is hypothesised from this that a more complex environment could elicit more considerable differences in the time spent looking at distractions and the number of mistakes made by participants while driving. This is based on driving research highlighting the influence of scene complexity on driver workload [128]. The environment of this driving simulation was simple to ensure the driver measurements were in response to specific items on the road. A study using a more realistic environment with more diverse surroundings may cause a significant response.

In summary, the following factors should be included in the driving simulator developed:

- Video-based driving simulation should be used to provide a complex envi-

ronment that is similar to driving in the real world and can challenge the driver with complex and realistic content.

- The video-based driving simulation should be steerable to simulate real driving as most people experience it.

- Sensor and metadata from driving simulation should be timestamped using global synchronization tools such as LSL. This allows for analysis that can use modern processing methods informed by the literature and can be adjusted for future changes to the methodology of analysis.

- While doing eye-tracking experiments, attention can be measured using gaze dispersal in video sequences [90, 107]. Clustering is a viable approach but is subject to noise introduced due to the inclusion of saccadic eye movements and misclassification of eye gaze positions. Fixation classification of eye movement is supported by the literature and gives fewer false positives for attention [4, 103].

- Measurement of attention concerning specific objects on the road is best carried out using AOIs[4, 103]. The investment in labelling data by humans is worth the results for the analysis of visual attention. These labels have been used to inform what participants give attention to, and how often they observed these items. An automated method of labelling would be helpful here.

- In driving simulations with total freedom to explore an environment labelling and comparing eye tracking signals would be difficult. Attempting to find the overlap of attention for participants could involve many hours of labelling video data. To resolve this challenge using an understanding of the 3D environment and how to communicate image coordinate data, including eye tracking, between multiple views. This could enable larger scale attention experiments where the participants have more autonomy.

- Although all participants completed eye tracking calibration and received an initial accuracy of $\pm 1.5^{\circ}$, some participants did not pass the eye tracking validation. Validation of eye gaze data before and after experiments resulted

in superior quality in the results observed during analysis. There were higher fixation dwell times over items not labelled as white space (WS) for participants who passed the validation check; see Table 3.10 for participants 3 who did not pass the validation. The validation step was built into the simulator application, which removed the requirement of the experimenter to switch between applications normally required for calibration. This step should not be ignored for future eye tracking experiments as it validates the consistency between eye gaze coordinates and screen coordinates and accounts for the precision of the calibration.

Chapter 4

Dashboard Footage to Photo-realistic 3D Models

This chapter will focus on pre-processing textures for 3D models to provide photorealism for a driving simulator experience. Video sequences can be collected for almost any scene in the real world using cameras. Real-world images can be used to reconstruct 3D models representing the geometry of a static scene captured in those images using open source Structure-from-Motion (SfM) tools [152, 139, 116, 115, 160, 189]. The weighted average of the colours from the 3D model reconstructed using these tools does not provide realistic lighting renderings. This is the reason we will implement projective texture mapping (PTM) in combination with the 3D models produced using SfM. Following reconstruction of a road, it is proposed to treat each image in a video sequence of a road as a separate texture to achieve photorealism over a corresponding 3D model. An artifact of this simulator includes the implementation for rendering 3D reconstructions with real images of that same scene localized and projected onto its surfaces using PTM. It sought to create high visual fidelity for possible future use in human behaviour studies within a driving simulator. This chapter provides a description of the 3D models used to test the practicality of PTM for real-world simulation. The geometric assumptions will be identified for each 3D model. Finally, the methods developed for implementing PTM will be discussed.

Video footage has been used for driving simulation, but it can lack fidelity.

The playback speed can be adjusted to create the effect of speed [77, 81]. A further improvement to fidelity is to add steering. Steering requires the ability to change the point of view away from the initially recorded perspective. Two approaches were applied while creating this effect. The first used a manually constructed simple model of the scene. This was to demonstrate that PTM could be applied to any 3D model created by a designer, there was no requirement for it to be linked to a model made from SfM. The second used 3D reconstruction tools including SfM. In this section, perspective projection of an image of the road will be used over a 3D reconstructed model to give the feel of depth to the driver. Projection onto simplified models, including a plane or open-ended box, can create visual artefacts. These artefacts become increasingly problematic when the transformation of the drivers' views and the projector diverge. When the camera and projector are in perfect alignment, the view becomes independent of the model on which it is projected and is free of distortion (assuming there are no visible gaps/holes in the 3D model). This also assumes the "viewing" camera model contains the same parameters as the projector used on the image. As the camera and projector pose separate, there is an increasing need for the model to be faithful to the real geometry. While generating an accurate 3D model of the scene is computationally intensive, it has the ability to remove many artefacts resulting in improved photo realism.

Section 4.1 formalizes the construction of 3D geometry and localization of 2D images. It was considers a pre-set approximation of an environment (hallway); an open-ended box. PTM is implemented using OpenGL. This implementation is tested using images of a hallway and comparing how well PTM with an open-ended box performs using photo-consistency metrics, including cross correlation.

This section then goes on to expand the generalizability of the 3D models. A 3D model was built as a proof of concept for linking PTM to an existing automatic 3D reconstruction algorithm. The KITTI existing dataset was used to allow for a comparison of the experiments [63]. The limitations of this approach are discussed, and an argument is presented for using a 3D reconstruction algorithm with RBG images.

4.1 Manual 3D Reconstruction and PTM

3D models can be manually constructed by setting 3D coordinates of a basic shape in a computer graphics environment like OpenGL. To simulate the Port Tunnel an open-ended box was used. Each corner of the box required a 3D coordinate, $\mathbf{p} \in \mathbb{R}^3$, and were organised in groups of 3 to form a triangle/face (polygons). This is a standard format to describe a 3D model; a mesh. To render these 3D coordinates to the screen requires a camera model. This model included the transformation of the camera (location, $\mathbf{t} \in \mathbb{R}^3$, and orientation, $\mathbf{R} \in \mathbb{R}^{3\times3}$, relative to the model), $\mathbf{P}_R \in \mathbb{R}^{4\times4}$, see Equation 4.1[1]. The camera parameters which dictate the projective transformation that occurred between a point's 3D camera coordinates and the 2D image space coordinates they map onto was represented by $\mathbf{K}_R \in \mathbb{R}^{4\times4}$. These parameters for the camera are manually set at the beginning by the experimental designer.

$$\mathbf{P}_R = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{SE}_3 \tag{4.1}$$

The pose of the camera can be updated to provide varying points of view. The projective transformation, $\mathbf{K}_R$, typically remained constant. It would be unusual for user controls to affect the interpretation of 3D space to the 2D image plane in a driving simulator.

4.1.1 PTM on Generic 3D Shape Models

PTM allows a 2D texture to be projected onto an arbitrary 3D surface as part of the rendering process used by OpenGL. The PTM technique was integrated into OpenGL version 1.1 and requires a stack based approach using named matrices to be updated while rendering objects. These matrices are inputs to the OpenGL stack to represent each coordinate system [145, 53]. The following labels were used to represent each coordinate system: model matrix (the transformation of the 3D model in world space, in this case, it will be assumed to be the iden-

[1]This notation can be aligned with the formalization outlined in Section 2.4.4. $\mathbb{SE}_3$ refers to the Special Euclidean group referring to the fact that this matrix follows rigid body motions causing translations and rotations but not reflections.

tity), MODELVIEW (the transformation of the rendering camera concatenated to the model matrix), PROJECTION_MATRIX (the perspective projection of the rendering camera), TEXTURE_MATRIX (the transformation and projection of the projector from image space to world space). A simple model consisting of an open-ended box was used to model a hallway (and ultimately a tunnel). The projector was placed in the reconstructed scene in the position and orientation equivalent to the position where the real camera was used to record the view.

Assume vertex coordinates in world space are represented as $\mathbf{p} \in \mathbb{R}^4$, to transform them to camera space an inner product of that point and the inverse of the VIEW_MATRIX, $\mathbf{P}_R \in \mathbb{R}^{4\times4}$, is calculated. For rendering this point from the viewing camera eye-space to the 2D image space on the computer screen the PROJECTION_MATRIX, $\mathbf{K}_R \in \mathbb{R}^{4\times4}$, is set during the program's initialization. This matrix then transformed the point $\mathbf{p}_R$ to image viewport coordinate $\mathbf{u}_v$ following dehomogenization of the point, see Equation 4.3.

$$\mathbf{p}_R = \mathbf{P}_R\mathbf{p} \tag{4.2}$$

$$\mathbf{u}_v = \pi(\mathbf{K}_R\mathbf{p}_R) \tag{4.3}$$

The 3D vertices were then converted to projector-space which allowed measurement with respect to the projector's frame of reference, Equation 4.5. Taking the vector-matrix product of the vertex coordinates, $\mathbf{p}_R$, with the planar coefficients for each of the three projector planes and scalar on those planes ($\mathbf{R}, \mathbf{S}, \mathbf{T}, \mathbf{Q} \in \mathbb{R}^4$) generates $\mathbf{p}_T \in \mathbb{R}^4$, texture coordinates in eye space. The camera position is set by using a homogeneous co-ordinate of the form $[x, y, z, w]^T$ and the projector position set using a homogeneous co-ordinate of the form $[r, s, t, q]^T$ where w and q are scale factors. Assuming the renderer uses perspective projection, the projector and rendering camera, can be described as a combination of a projection matrix based on camera parameters (focal lengths, skew, aspect ratio, resolution) and a view matrix describing their transformation. The main difference is that a camera samples the 3D positions of a model to form an image. The projector samples 3D points to locate the part of the image to reference as a texture coordinate for colouring the 3D point. The planes can then be composed

in a matrix $\mathbf{T} \in \mathbb{R}^{4\times4}$, see Equation 4.4.

$$\mathbf{T} = \mathbf{K}_T\mathbf{P}_T \tag{4.4}$$

$$\mathbf{p}_T = \mathbf{T}\mathbf{p}_R \tag{4.5}$$

The projector was orientated and located by defining planes (**S**, **T** and **R**) along the world coordinate **X**, **Y** and **Z** axes that were coincident with the world coordinate system, centred at the origin and projecting along the Z-axis. In this case $\mathbf{p}_T = \mathbf{p}$, see Figure 4.1. In Figure 4.1[1] vertices further from the **S** and **T** planes corresponded to texture coordinates further from the focus of expansion (FOE) in the texture. Increased distance from the **R** plane moved points towards the FOE, (small far away). The TEXTURE_MATRIX, $\mathbf{T} \in \mathbb{R}^{4\times4}$, was configured to put the FOE at the centre of the texture. The texture matrix was also used to scale, rotate, translate, using $\mathbf{P}_T$, and applied a perspective transform $\mathbf{K}_T$ to the texture, see Equation 4.4.

Normalizing and dehomogonizing the point $\mathbf{p}_T$ provides the 2D (u, v), $\hat{\mathbf{u}} \in \mathbb{R}^2$, required to access the texture, Equation 4.6.

$$\hat{\mathbf{u}} = \pi(\mathbf{p}_T) \tag{4.6}$$

With the correct values of scale, aspect ratio, rotation, and field of view, which coincide with the cameras intrinsic and extrinsic parameters used to capture the image being used, the projector can be constructed such that the projected image aligned with the structure of the model, see Figure 4.2. The white edges in the figure represent the boundaries between walls, ceiling, and floor used to line up with the edges of the cube model. In practice the alignment was achieved using slider bar controls on a Windows application until the super-imposed white lines of the box coincided with ceiling, wall, and floor boundaries within the image.

The position and orientation of the vehicle (rendering camera) can be described using two parameters to set the displacement, d, which is the position on the road and a parameter to set the direction, θ. These parameters are used to

[1]Figure 4.1 and Figure 4.3 are from our previous publication [31], but there is no higher quality vector graphic available.

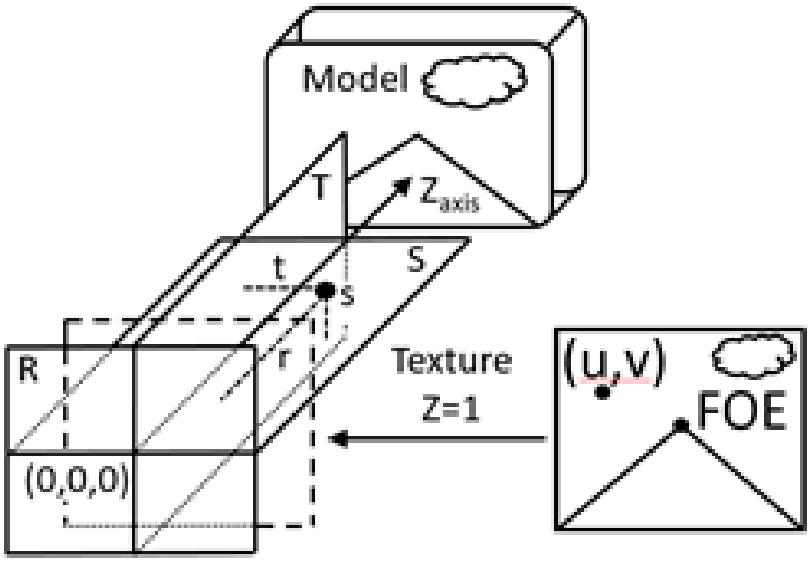


Figure 4.1: PTM, automated generation of texture co-ordinates.

Figure 4.2: Image projection onto a open-ended box.

build the VIEW_MATRIX for the camera, see Figure 4.3.

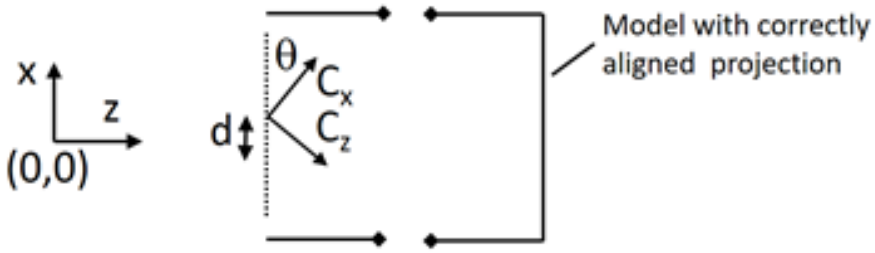


Figure 4.3: Moving the camera relative to the model to simulate vehicle motion (plan view).

4.1.2 Testing PTM on Generic 3D Shape Models

To commission the projection system 6 images were collected of a straight hallway (75 meters in length, 1.8 meters wide, and 2.5 meters high). The images collected included, a central straight view image (placed 90 centimetres from the right and left wall, and approximately 1 meter from the ground leaving the camera front and parallel to the central axis of the corridor) from the end of the hallway, centrally positioned with left and right yaw rotation, and left and right translation of the camera (placed 45 centimetres from the right and then the left wall). The central straight view image was used as the input to the PTM program and using this software all other views mentioned were simulated.

In practice, the simulated images and real images were cropped to limit the field of view presented, see Figure 4.4. The original real images contained additional information at the edges that the forward-looking image did not capture and so could not be recreated. The main effect of cropping was a reduction in the field of view from what was available with the camera used to capture the original texture image. The simulated images are quite close to the real-world representations. However, there are some minor artefacts that become apparent such as sampling and aliasing effects in the rendering of fine lines of the floor. There is also stretching of objects contained within the hallway rather than on the walls, floor, or ceiling. An example of this can be found in the simulated right position central view versus the right position central view (waste bins) Figure 4.4.

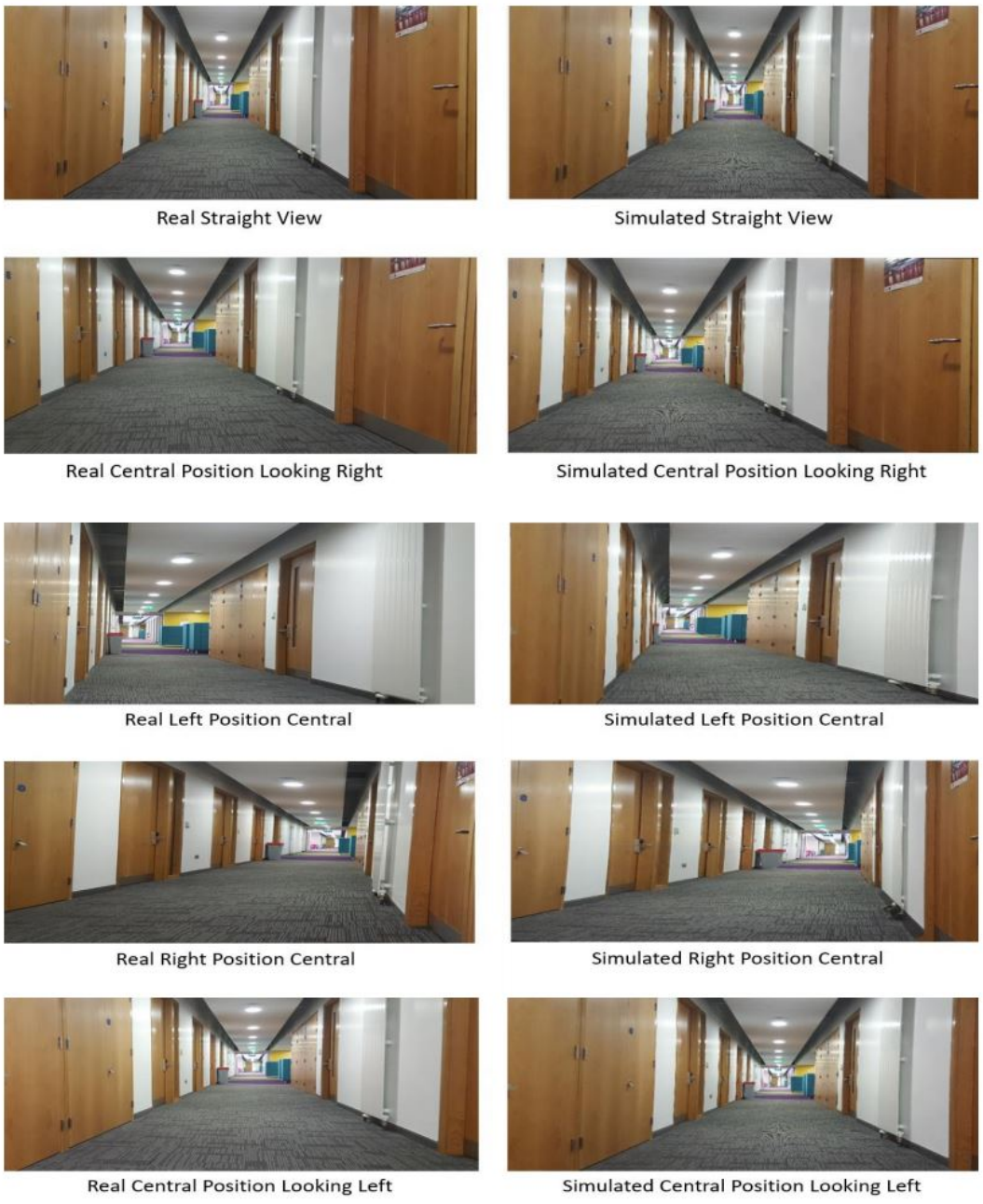


Figure 4.4: Left-aligned are the real images of a hallway. Right-aligned are the simulated images using PTM.

Cross-correlation was used to compare how well the input image represented the other points of view, before and after simulated projection of the images above in Figure 4.4. Using this metric, a score of 1 represents complete correlation/similarity, and a scores closer to 0 represent opposing signals to the input template image. The average normalised cross correlation score when all points of view were compared against the simulated images was 0.926 ± 0.003, and the score for the non-warped input image against each of the real points of view was 0.909 ± 0.021. These values are represented in the box plot below in Figure 4.5. The sample over the "input image" box is an outlier because the real image was compared against itself, so it reports a cross correlation value close to 1. The speed of this application during render time operates at 31.25 frames per second on a laptop with an Intel core i5 processor and a GeForce GTX 950M graphics card. The original texture images of the hallway had a resolution of 5312×2988 pixels, and the simulated image acquired 2916×2360 of those pixels[1]. The reduction in resolution is 57%-caused by the reduced field required to accommodate lane movement and steering.

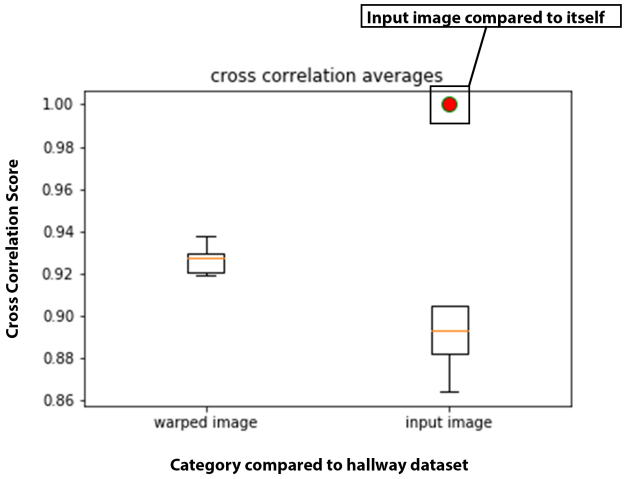


Figure 4.5: Average cross correlation for warped images versus real images.

Figure 4.6 presents the application of the PTM tool on an image of the Dublin port tunnel. There were no real-world images to compare various positions and

[1]This will be compared to future implementations of PTM

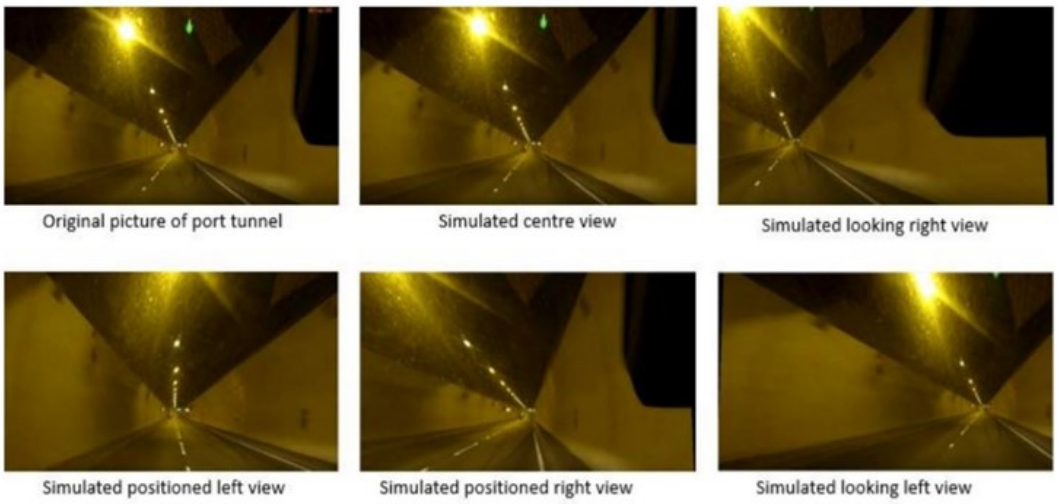


Figure 4.6: First image of the Dublin port tunnel as labelled, and all other images are the simulated orientation made.

orientations, however, it highlighted that it is possible to make a set of simulated images based on what they would be expected to look like from a driver's perspective. The original picture contains a sun visor in the top right corner of the image. In other images where this image is warped to fit a new perspective the sun visor is stretched across the screen, see Figure 4.6 in the top-right panel and the bottom center panel.

Summary: In this section, rendering features have been discussed. The projection of images onto approximate 3D surfaces works well for images with few foreground objects. Although it is possible to calculate the projection of an image onto 3D surfaces such that it appears photo-realistic from varying viewpoints, it is limited to the time and effort of the user to align the projector's transformation matrix (TEXTURE_MATRIX). More considerable transformational distances for the rendering camera (VIEW_MATRIX) from the projector transformation matrix result in less fidelity due to visual artefacts, see Figure 4.4. These include inconsistent reflections and distortion of foreground objects (less noticeable when the camera extrinsic matrix and projection extrinsic matrix are aligned). The following section will introduce solutions to these challenges.

4.1.3 Limitations and Benefits of Manual 3D Reconstruction

Projections onto predefined 3D surfaces allowed for the commissioning and testing of perspective projection implementations. Initial studies focussed on the projection process. Projecting onto an open-ended box revealed the artefacts that can occur with imperfect models. These discoveries informed later stages of development. General 3D reconstruction-from-image algorithms, such as SfM, can be time-consuming and often require domain knowledge to generate accurate 3D models. The main limitation of using the assumed shapes is that the geometry will not always match the surfaces observed on real roads. A different model would be required for each type of road (wide open scenes/tunnels/building-enclosed streets). Projections will need to be realigned to the model depending on the trajectory taken on the playback camera relative to the road. For example, turning into bends shifts the focus of expansion, and either the assumed 3D model or the projecting camera would have to be transformed to reflect this. Orienting the projecting camera with the focus of expansion would improve texture and model alignment. However, this approach has similar complexity to SfM methods already established [189, 139, 115]. A generalized shape cannot model unexpected foreground objects. This often results in visual artifacts, including the stretching of objects witnessed in the texture by the geometry. Using accurately localized cameras and 3D models solves each of these issues. The process of achieving this will be explored further in the next section.

4.2 Reconstruction and PTM - COLMAP

COLMAP is a general-purpose SfM and Multi-View Stereo (MVS) pipeline that allows for reconstruction with unordered and ordered image datasets. COLMAP was successfully used to reconstruct a road using solely RGB data images from the KITTI dataset [139]. Its inputs were camera intrinsic parameters of the sensor/camera and a set of RGB images. In other experiments done throughout this thesis RGBD images were used from the ICL-NUIM dataset using Elastic Fusion [68, 184], see Appendix C.1. The approach developed in this section aimed to

use a standard dashboard camera that would not require an accompanying depth sensor. In this case, COLMAP was preferable. Although LIDAR approaches (e.g., Velodyne) have been preferred by others in the past, they often require dedicated vehicles for data acquisition due to their high prices and complexity [202]. Ideally, a road engineer in an unmodified vehicle could use the techniques proposed in this thesis with commercial non-specialist equipment. The output of COLMAP included a set of odometry measurements corresponding to the RGB images and a dense mesh of 3D points and colours.

$$\mathbf{P}_t = \begin{bmatrix} \mathbf{R}_t & \mathbf{t}_t \\ 0 & 1 \end{bmatrix} \in \mathbb{SE}_3 \tag{4.7}$$

COLMAP was used to create a map, $\mathcal{M}$. This map was composed of a set of vertices where each vertex has a position $\mathbf{p} \in \mathbb{R}^3$, and colour $\mathbf{c} \in \mathbb{N}^3$, The image space domain is defined as $\Omega \subset \mathbb{N}^2$, where an RGB image was used. The colour image C of colour pixels $\mathbf{c} : \Omega \to \mathbb{N}^3$. A set of transformations were computed to represent the poses of each of these images relative to the mapped vertices but was rewritten with a subscript t for each individual pose at a given time, see Equation 4.7. This used the same interpretation of 3D points described in the previous section while describing the rendering camera $\mathbf{P}_R$. The camera intrinsic parameters are the same for every image and follow the same convention as the cameras described in Equation 2.1.

4.2.1 The KITTI Dataset

The data used for these reconstructions included RGB images of real roads from the KITTI benchmark dataset [63]. Sequences 0, 2, 5, 13, and 18 colour images from the "odometry dataset" were used. For each sequence there was at least one 3D model reconstructed based on between. Two models made for sequence 13 where two separate roads were reconstructed and later tested. The conditions for the selected sequences were that each sequence must have overlapping paths within its sequence. An overlapping path will be defined as the camera observing the same road multiple times. The application of these sequences will be described in more detail in later sections. The analysis of system performance is then

discussed. The results of using the KITTI dataset data with the COLMAP SfM reconstruction tool can be seen in Figure 4.7.

Figure 4.7: Image of road (top) 3D reconstruction of same road (bottom)

While using COLMAP, the best results were achieved during the SfM pipeline by extracting image features with default settings, exhaustive feature matching (compared to sequential and vocabulary tree), default triangulation, and bundle adjustment settings. The MVS pipeline used default settings for all stages (undistort, stereo patch match, Poisson mesh conversion). Using COLMAP provided a dense 3D model with a set of images, each localized relative to the model. The camera's intrinsic parameters were known. All images could be projected into 3D space so that the images' features matched the model's features. This was done using only RGB image measurements.

4.2.2 Video to Simulation

Overview: The three 3D models considered in the previous section (open-ended box, Elastic Fusion mesh, and COLMAP mesh) are inputs for this sections and Appendix C.1. The data included vertex positions, vertex colours, images, and odometry. PTM uses the input images to synthesize the view of the model. The texture is rendered on the model. It is photorealistic, given that the texture is a

photograph of the real world. 3D reconstructed models often make Lambertian world assumptions, and the vertex colours on the model are typically a weighted average. In Section 4.1 PTM was implemented using OpenGL's built-in shaders from version 1.1. Following this, a formalization of PTM was connected to 3D reconstructed scenes. Initially, this was completed using the ICL-NUIM 3D models reconstructed from Elastic fusion, see Appendix C.1. Our implementation of PTM shaders was discussed to adjust to the complexity introduced using reconstruction system parameters. The default PTM shaders provided a working but minimal implementation. These shaders are re-implemented to include extensions for visual artifacts such as reverse projection [53], projection onto non-front-facing vertices [43], and logical operators to create a hybrid PTM implementation; vertex colour, and multiple projections for 3D points not exposed to a single projector. This was a novel contribution to the field [33]. Given the similarity between the outputs for Elastic Fusion and COLMAP, the techniques developed for PTM are portable between the two systems.

4.2.3 System-level design for custom shaders

Section 4.1.1 explained how the the integrated shaders for PTM in OpenGL could be applied to surfaces in general. However, it was challenging to introduce the changes required in the graphics pipeline to develop the driving simulator. As a result, a modern approach (model view projection, MVP) to writing OpenGL shader code was taken. The original PTM shaders lacked the versatility to render different effects based on perspective projection criteria. Rather than treating the OpenGL API and the access to the graphics card as a "state machine" managed using a stack, code was explicitly written to the graphics card in the form of shaders. It was controlled using a model, view, and projection approach [145]. This approach provided access to the graphics card. It was possible to choose to compute on the CPU and the GPU. This helped achieve the high frame rate operations required for driving simulation (above 30 fps). Writing shaders enables logical operations on the graphics card, which enables the hybridization of shading methods for a single 3D object. This feature will be helpful for shading surfaces not exposed to the projector. The following section outlines how the shaders

were included in a system-level design. It introduces the methods used to read 3D models, images, and the associated odometry from memory. It then describes the processing of this data from the CPU to the GPU. The interactive game loop and the shaders developed are also discussed. The rendering system is described in terms of initialization, game loop, and key-call-back phases. Frequency of use and execution times are referenced to give the impression of the run time performance. For example, the steps during initialization are not required to run at framerate as they are required at the beginning, and loading for a driving simulator at the start is considered standard. However, the game-loop operation is required to run at a minimum of 30 fps. Key-call-backs are executed on a separate thread and should not affect the rate that the screen is rendered.

3D Mesh: Reading 3D reconstructed models are implemented in the same way as reading any other type of 3D model used in computer graphics. However, the models carried vertex colours and textures in this work. Assuming the outputs of 3D reconstructed models had undergone some meshing procedure [83, 139], they can be interpreted as a list of triangles. The typical format used in the fore-mentioned sections was Polygon File Format (ply). Based on the header description of these files, they were parsed to include each vertex's position, colour, and normal. Colour and Normals were accepted to enable the shading of 3D coordinates not exposed to the viewing space of the projectors used for texturing. The vertex colours were a weighted average of the image projections from the previous 3D reconstruction step. This approach enabled the removal of seams where a projector was not available.

Open Asset Importer Library was the library for loading these models [141]. These vertices and their drawing order were loaded in buffers which would later be interpreted by programs on the GPU (shaders). See Figure 4.8 for an example of the vertex buffer layout. An index buffer could also indicate the drawing order of the triangles. This is a numbered array that selects from the vertex buffer which vertices are part of the triangle to be drawn. This format often reduces the number of vertices saved in memory. Instead of having multiple instances of the same vertex, where a vertex uses nine floating point numbers worth of memory, a single index reference in the index buffer can be used to point to the nine vertex attributes in the vertex buffer. This significantly reduces the amount of

memory used if the same point were involved in drawing many polygons. OpenGL required that the buffer objects (vertex buffer and index buffer) receive attributes describing exactly how many bytes to set aside for each component. This included the number of bytes per position, colour, and normal for the vertex buffer and the number of bytes between each vertex for the index buffer. A detailed review of how to apply similar setups for a graphics pipeline can be found here [145, 178]. When texture information is rendered, the UV-values are calculated at render time; the file carries the RGB data.

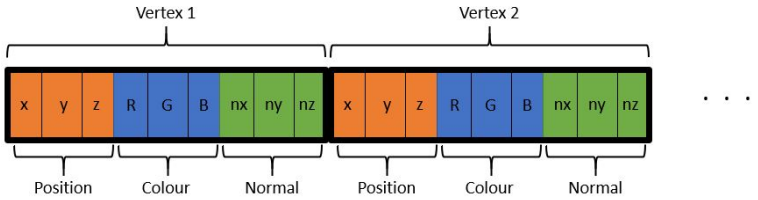


Figure 4.8: Vertex Buffer Layout. Each vertex containing three numbers for position, colour, and normal direction in that order, respectively.

Images: At initialization, 200-300 images are read as texture objects in CPU memory. This is the number of images used for each 3D reconstructed model. The images also have a data file that defines their relationship relative to the 3D model. For 3D models produced using Elastic fusion, this is called a Freiburg file (using the Freiburg odometry formatting). This data was not from the Freiburg dataset, it was from ICL-NUIM. Elastic Fusion produced odometry in this format by default. This contains the timestamp, location as a 3D coordinate $[p_x, p_y, p_z]$, and orientation as a quaternion $[Q_x, Q_Y, Q_Z, Q_W]$ relative to the 3D reconstructed model. COLMAP uses similar formatting, but the ordering of the data is different. It uses the image ID, the orientation as a quaternion in a different order $[Q_W, Q_X, Q_Y, Q_Z]$, and the 3D coordinate as the location $[p_x, p_y, p_z]$. Another file is loaded to indicate nearest neighbour images that can be used as projection partners while texturing the scene. Appendix D develops a method to calculate this data on the fly. This file is created by comparing multiple sequences traversing the same road. It is intended that this projecting partner will cover projections missing parts of the 3D model within their view. Projection partner

groups are formed based on the minimum Euclidean distance of the poses found between the two sequences. Depending on the graphics card, 16 to 32 images can be stored within the GPU memory. While traversing the road and rendering the 3D model, the images loaded into GPU memory were held within a queue. New images enter this queue as their transform approaches the driver transform. The image already in the queue popped out when further away from the driver transform compared to the set. Camera parameters associated with each video sequence were stored locally on the disc contained within a ".calib" file. This file contains the images' focal length, resolutions, and camera centre in the associated sequence. After this was parsed, it was possible to construct a camera object that could later be used to calculate the image's perspective projection.

Shaders: Shader programs themselves must be loaded from disk memory and compiled by the OpenGL API while initializing the game loop for the driving simulation. Each file has been saved as a ".shader" file and written in GLSL (graphics library shader language). In this graphics engine, two shader programs are used in combination for rendering each object. These are the vertex and fragment (pixel) shader. The vertex shader is a program executed for each vertex attached to the object associated with that shader. It is often used for calculating transformations on the vertex coordinates. These calculations include model-based transformations (moving the entire associated object with a rigid body transformation), viewing camera transformations (rigid body transformations providing the illusion of the viewing camera moving), and projective transformations (converting from camera viewing 3D space to the 2D image plane). The fragment shader is a program executed for each pixel on the viewport. This is usually the moment to make final adjustments to the appearance of pixels in the viewport. These adjustments usually have to do with colour, illumination, and shadows. Each ".shader" text file contained both the vertex and fragment shader in this system. Each file line was parsed and queried against a regular expression of "#shader vertex" and "#shader fragment." The vertex shader program was located between "#shader vertex" and "#shader fragment". The fragment shader was between "#shader fragment" and the end of the file. These shaders would later be used in the graphics pipeline to interpret 3D data on the graphics card. Each shader can also accept a set of custom parameters called uniforms at ini-

tialization and during the game loop.

Cameras: To enable the rendering of the 3D model, a camera object was used. The camera contained a view, an inverse view, and a projection matrix. The view matrix was used to model the camera's position within the 3D space containing all the objects being rendered. If that camera was set as the rendering camera, only the objects in front of the camera were rendered. The inverse view matrix was used for instances where the camera needed to be modelled in 3D space according to other camera view matrices. The projection matrix described the 3D points' transformation into the 2D image plane. The camera calibration directly affects the projection matrix. The centre of projection and focal lengths need to be known to model a camera for use as a projector. Guides for understanding the connection between the OpenGL projection matrix and the Hartley and Zisserman definition [70] of these matrices were given here [150, 162]. If the camera was the rendering camera, then the 2D image plane was the image shown to the driver. Projectors can be used interchangeably with camera objects.

Frame Buffer: The frame buffer was created as a source of rendering objects into a 2D space that was not the final 2D viewport exposed to the driver. This would later be useful for rendering depth values from perspectives outside the viewing camera while doing shadow calculations for the projectors. The contents of the frame buffer could be transferred to a texture image and read as a uniform in another shader. This resource was helpful as a guide to using frame buffers [178].

During the game loop, there are multiple threads executed at once. These included the rendering, GUI call-back, and key call-back functions. The rendering function was responsible for delivering updates to shaders and issuing draw commands from the CPU to the GPU. The key call-backs were associated with the user input, and the user input for the driving simulation was used to vary the motion of the rendering camera. The GUI call-back was used for rendering the speedometer, but call-backs to these functions were not typically used.

Rendering: At the beginning of each render loop, the viewport was cleared of previous depths and colours from the last render cycle. The camera view matrix was queried to identify updates to the viewing state of the camera and projectors.

This would later inform shaders of updates to the viewing transformations of each object relative to these projectors and cameras. Each object had its associated shader bound to the graphics card, and selected uniforms (CPU-GPU shared variables) were then updated. These uniforms were usually associated with the camera/projector view matrix and allowed the application of locally referenced textures for projection. After all uniforms were set, the shader would then be used to draw to the viewport or a frame buffer. This was done for all objects included in the 3D mesh.

Key call-back: Initially, the keys from the keyboard were used to update the rendering camera pose during system testing. Each key would change a different axis of translation and rotation of the rendering camera. The updated translations are delivered using a vector $\dot{t} \in \mathbb{R}^3$ added to the current translation t. The updated vector $\hat{t} \in \mathbb{R}^3$ was then composed into a transform matrix, and the matrix-vector product between that matrix and the current rotation matrix, $\mathbf{R} \in \mathbb{R}^{3\times3}$, was calculated. To update the rotation, the input was an angle, $\theta \in \mathbb{R}$, and a vector describing the axis the camera was to rotate around $\mathbf{r} \in \mathbb{R}^3$. The angle-axis notation was then converted to a matrix $\dot{\mathbf{R}} \in \mathbb{R}^{3\times3}$ and this was used to update the current view matrix of the rendering camera. Chapter 6 further develops this with the pedal and steering wheel interface.

4.2.4 Implementation of Shader Algorithm for PTM

The approach described in Section 4.1.1 used default shaders built into the stack-based (OpenGL version 1.1) implementation of OpenGL. The following section describes the modern approach using customised shader programs. This approach uses shader programs implemented in GLSL to allow customised rendering processed on the graphics card. This code takes as its input the reconstructed models produced using Elastic fusion and COLMAP for the ICL-NUIM and KITTI datasets, respectively. The reconstruction using the ICL-NUIM dataset with Elastic Fusion have their specifics described in Appendix C.1 for commissioning these shaders. The shaders developed implement PTM. The approach developed in this section rejects reverse projection, a limitation of the default projector, which projects both forwards and backward. The shader limits projections to

objects closest to the camera, preventing background surfaces from being incorrectly textured. The shaders also allows for a hybrid approach of PTM, including a mix of vertex colours and PTM over regions where PTM leaves seams due to perspective occlusion.

Shader implementation of PTM: The projector positions are known at initialization, provided by the reconstruction steps in Section 4.2. Only the camera view needs to be updated. The code developed loads the projector positions into the system at run time alongside the model data. Only the camera view is changed during run time. The following section outlines a workflow for combining 3D reconstruction outputs, the model $\mathcal{M}$ and the camera odometry $\mathbf{P}$, to allow for PTM over a set of points given the relevant transforms of the projector.

Using the map $\mathcal{M}$, representing a reconstructed model and the extrinsic parameters of the cameras used to reconstruct the scene of the form Equation 4.1, it was possible to construct a workflow that implemented PTM with a reconstructed model. Assuming good odometry (low root mean squared error from the ground truth trajectory), the colour values of the image C should re-project onto the surfaces of $\mathcal{M}$ with high accuracy when comparing image features [99, 137]. It is helpful to consider the odometry as a list of poses in the order of the frames captured of the form Equation 4.7. The calibration matrix, $\mathbf{K}$, is the same for all poses $\mathbf{P}_t$. Every previously captured pose could be expressed between all points $\mathbf{p} \in \mathbb{R}^3$ in the map $\mathcal{M}$ with the frame corresponding to that pose, where $\mathbf{u}$ is the texel to be sampled for the surface. The inclusion of the camera parameters eliminates the need to manually adjust the projections to fit the model, as discussed in Section 4.1. Algorithm 1 details how vertex coordinates, colours and normal's are read and transformed. The positions $\mathbf{p}$ of the vertices can be read using a model-view-projection matrix M_{cam} on line 12. This should convert the vertices to normalized device coordinates to be interpreted by the fragment shader. This will be developed later in Algorithm 3.

Limiting shader to forward projection: Ideally the process would be the same for the texture coordinates, $\mathbf{p}_{texndc}$. OpenGL automatically carries out perspective division and selecting front facing points, however, it does not do this for variables used in the shader (aside from glPosition). Points behind the projector are textured with a flipped image, this is referred to as reverse projection

Algorithm 1: PTM (Vertex Shader)

Result: Vertices mapped to projector and camera coordinates

1 **Input**: $\mathbf{p}$, $\mathbf{M}_{cam}$, $\mathbf{P}_{proj}$, $\mathbf{K}_{proj}$, $\mathbf{c}$, $\mathbf{n}$;
2 **output**: vTexCoord, glPosition, vColour;
3 $p_{texeye} = \mathbf{P}_{proj}.\mathbf{p}$;
4 **if** $p_{texeye}[z] < 0$ **then**
5 $\quad p_{texndc} = \mathbf{K}_{proj}.p_{texeye}$;
6 $\quad$ vTexCoord $= \frac{p_{texndc}}{ptexndc[z]}$;
7 **else**
8 $\quad$ vTexCoord $= vec4(inf, inf, inf, 1)$;
9 **end**
10 vcolor $= \mathbf{c}$;
11 vNormal $=\mathbf{n}$;
12 glPosition $= \mathbf{M}_{cam}.\mathbf{p}$;

[53] see Figure 4.9 for an example. This is addressed by considering the points in eye-space and checking the condition: $\mathbf{P}_t\mathbf{p}[z] > 0, \forall \mathbf{p} \in \mathcal{M}$, to ensure the points are in front of the projector, see line 4 in Algorithm 1.

Limit shading to nearest objects: Projecting onto items behind the intended foreground object was a significant artefact to remove. This problem was covered by [43] but was not considered in the previous Section. A projector uses a viewing frustum pointing towards some set of 3D points. See Figure 4.10, where the frustum is the red-tinted cone emitted from the camera. The left image outlines the vertices of the objects, which will receive the colours from the projected texture from the projector position. In this case, it includes all the points within the viewing frustum. The right image shows the vertices that should be solely considered while projecting from this perspective, assuming occluding objects receive the projection over background objects, ensuring lamp image colours do not cover the chair's vertices. See Figure 4.11.c for an example of a foreground object's colours projected onto occluded background 3D points.

To eliminate the projection onto occluded surfaces a variant implementation of shadow mapping was considered [178]. To check if a vertex, $\mathbf{p}$, should receive colours, $\mathbf{c}$, from the texture image, Figure 4.11(a), the depth was tested from the perspective of the projector using its matrix $\mathbf{M}$ from Equation 4.3. A depth map $\mathcal{K}_t$ was retrieved from this test and stored in a separate texture, see Fig-

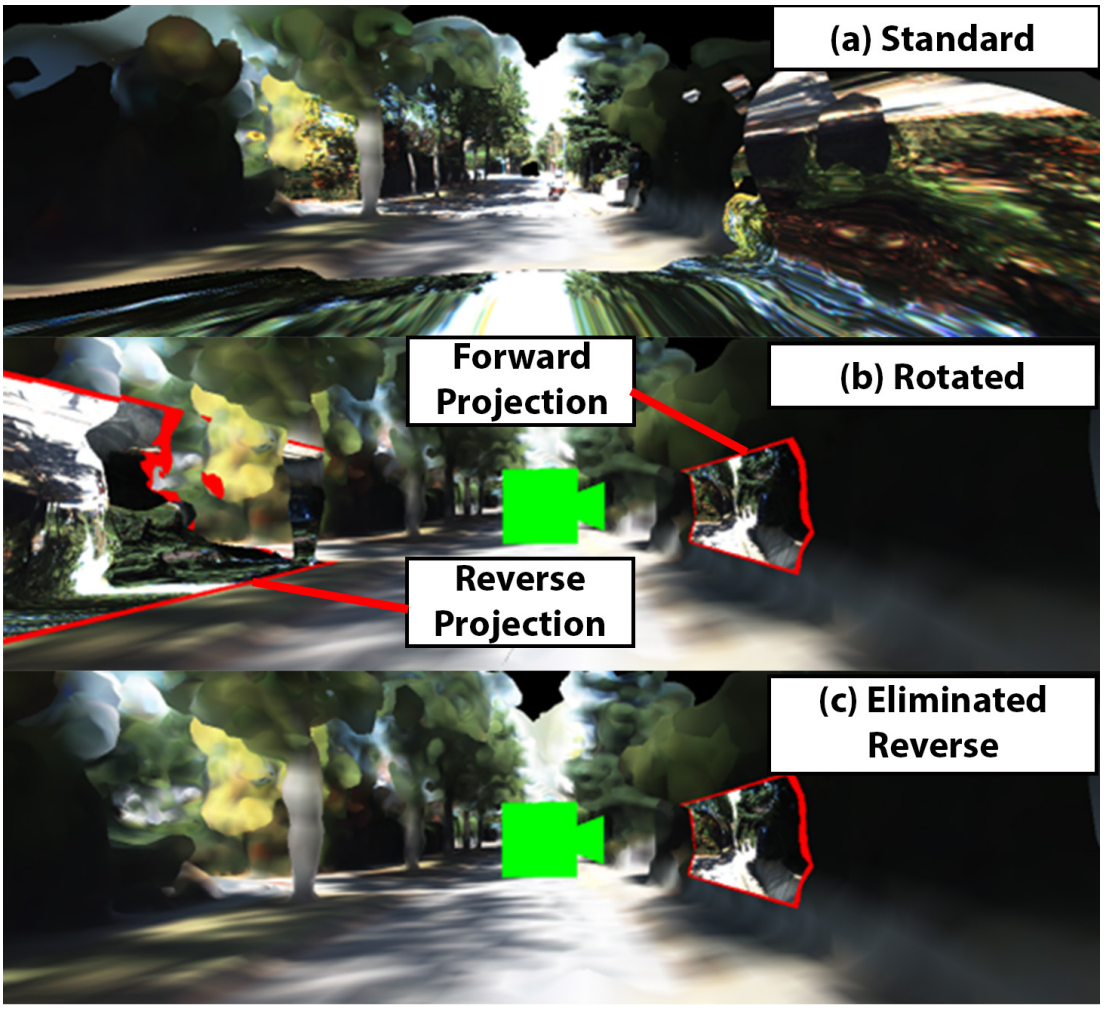


Figure 4.9: (a)In this example there is a projection of an image further up the road. The projection is reversed and texturing the vertices behind the correctly oriented projector. (b) to highlight the reverse projection the projector has been rotated around its own y-axis so the reverse projection can be rendered in view of the rendering camera. The projection has also been surrounded by a red bounding box. A green camera highlights the position of the projector (c) shows the forward projection without the back projection.

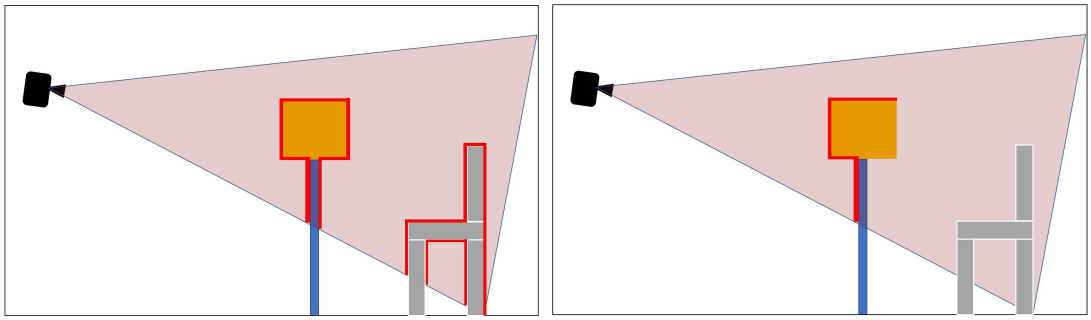


Figure 4.10: Projection of vertex coordinates to image. Left image: Projected texture shown on all surfaces., Right image: Projected texture limited by shader to nearest surface

ure 4.11(b) for an example of the depth map. While rendering the 3D points from the perspective of the viewing camera their positions were transformed into projector space so the z-value (depth) could be compared to the depth map $\mathcal{K}_t$. If the z-values transformed from the camera coordinate frame to the projector coordinate frame were larger than what was found on the depth map from the rendering camera then it could be concluded that the point was occluded and should not receive the projectors colours, see Algorithm 2. Other considerations included the removal of shadow acne [178]. Depth values in the $\mathcal{K}_t$ are limited by the resolution of the texture used to store it. For this reason, some of the fragments received incorrect depth values which they originated from neighbouring pixels. This results in some missing regions of the projection. Using the approach by [178] for shadow maps, a bias calculation was made, such that a small decision boundary would be formed around the surfaces of the model being projected on. This was based on the difference in distance and angle between the surface of the model and the projector allowed for texturing, see lines 8-9 Algorithm 2.

Following PTM, points were clipped to ensure that only real points were mapped from image $\mathbf{C}$ to the model $\mathcal{M}$. The points were clipped if they lay outside the clipping points (normalised device coordinates) for $\mathbf{p} \in \mathbb{R}^3$ in the range $[-1, 1]$, see the first condition in line 4 Algorithm 3. Using normalised device coordinates, the clipping space technique is a typical method for rendering a 3D scene with perspective geometry in OpenGL [145, 2].

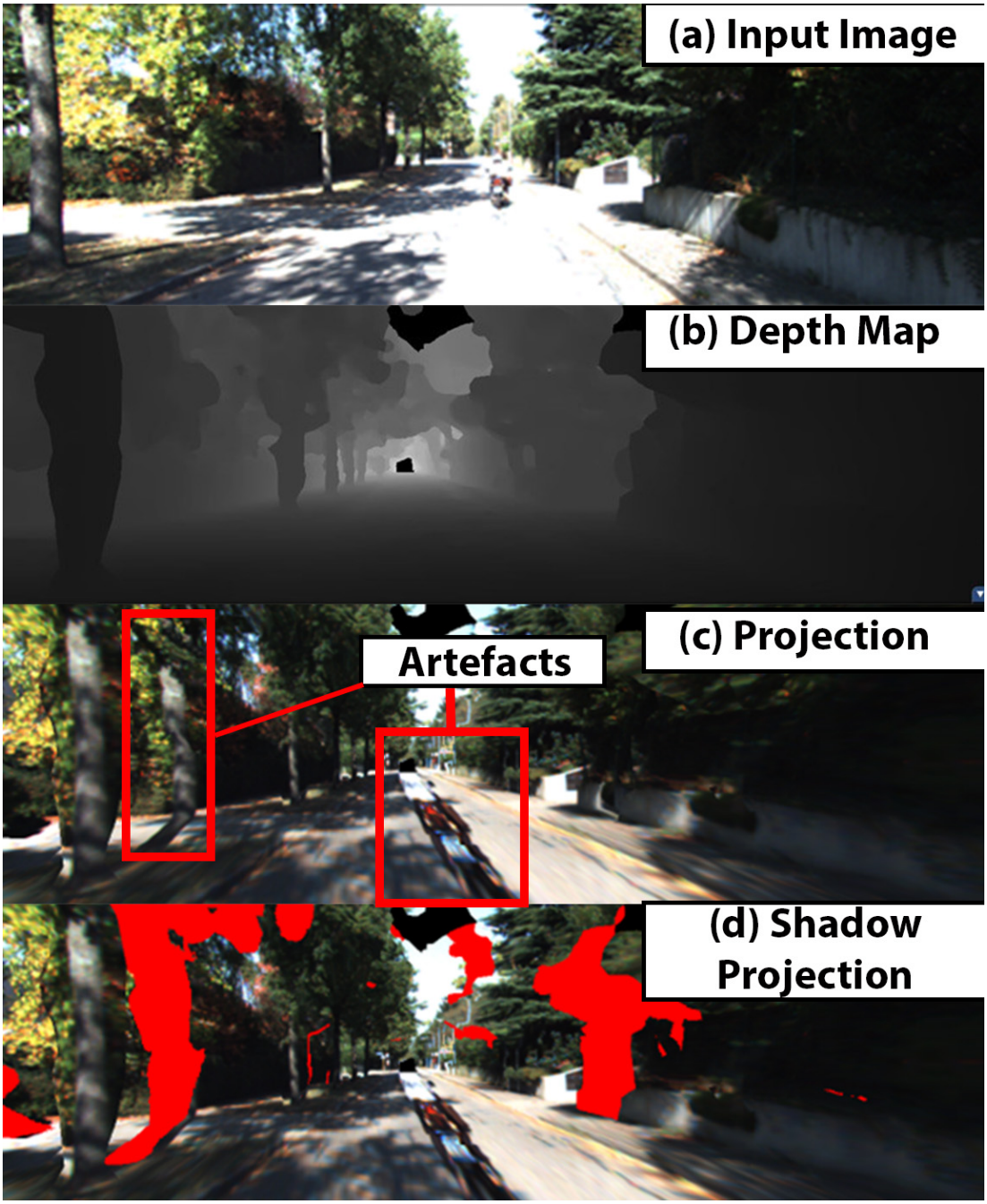


Figure 4.11: (a) An image projected onto a reconstructed surface rendered from the same pose as the projector. (b) a depth map of the 3D model sampled from the pose of the projector in (a). (c) Following the projection of the image in (a) while observed from another view it is noticeable that the projection goes onto surfaces behind desired surfaces. d) a shadow calculation has been used to remove projections on surfaces occluded by another surface. The red shaded areas highlight an area where an occluding object should block the projector's texture from the surfaces behind it. In image (a) the projection is used on all surfaces in front of it however in (d) it projects only on the surfaces visible to the projector.

Algorithm 2: Shadow Calculation

Result: Vertices mapped to projector and camera coordinates

Input: fragmentCoord, $\mathcal{K}_t$, vNormal, $\mathbf{P}_t[t]$, vTexCoord;
output: occluded;
projCoords = (fragmentCoord $\times 0.5$)+0.5;
projDepth = linearize(texture($\mathcal{K}_t$, projCoords[xy])[r]);
depth = linearize(projCoords.z);
normal = normalize(vNormal);
projNormal = normalize($\mathbf{P}_t[t]$−vTexCoord[xyz]) ;
bias = $\max(0.5 \times (1.0 - dot(normal, projNormal)), 0.05)$;
difference = depth - projDepth - bias ;
occluded = difference > 0 ;

Algorithm 3: PTM (Fragment Shader)

Result: A rendering calculation for each pixel on the viewport

Input: vColour, vTexCoord, C, $\mathcal{K}_t$, $\mathbf{P}_t[t]$, vNormal;
output: oColour;
projCoord = vTexCoord[xyz];
if *ClipCoords(projCoord)* ***and*** *ShadowCalculation(projCoord,* $\mathcal{K}_t$*, vNormal,* $\mathbf{P}_t[t]$*, vTexCoord)* **then**
    projCoords = (projCoord $\times 0.5$)+0.5;
    oColour = texture(C, projCoords[xy]);
else
    oColour = vColour;
end

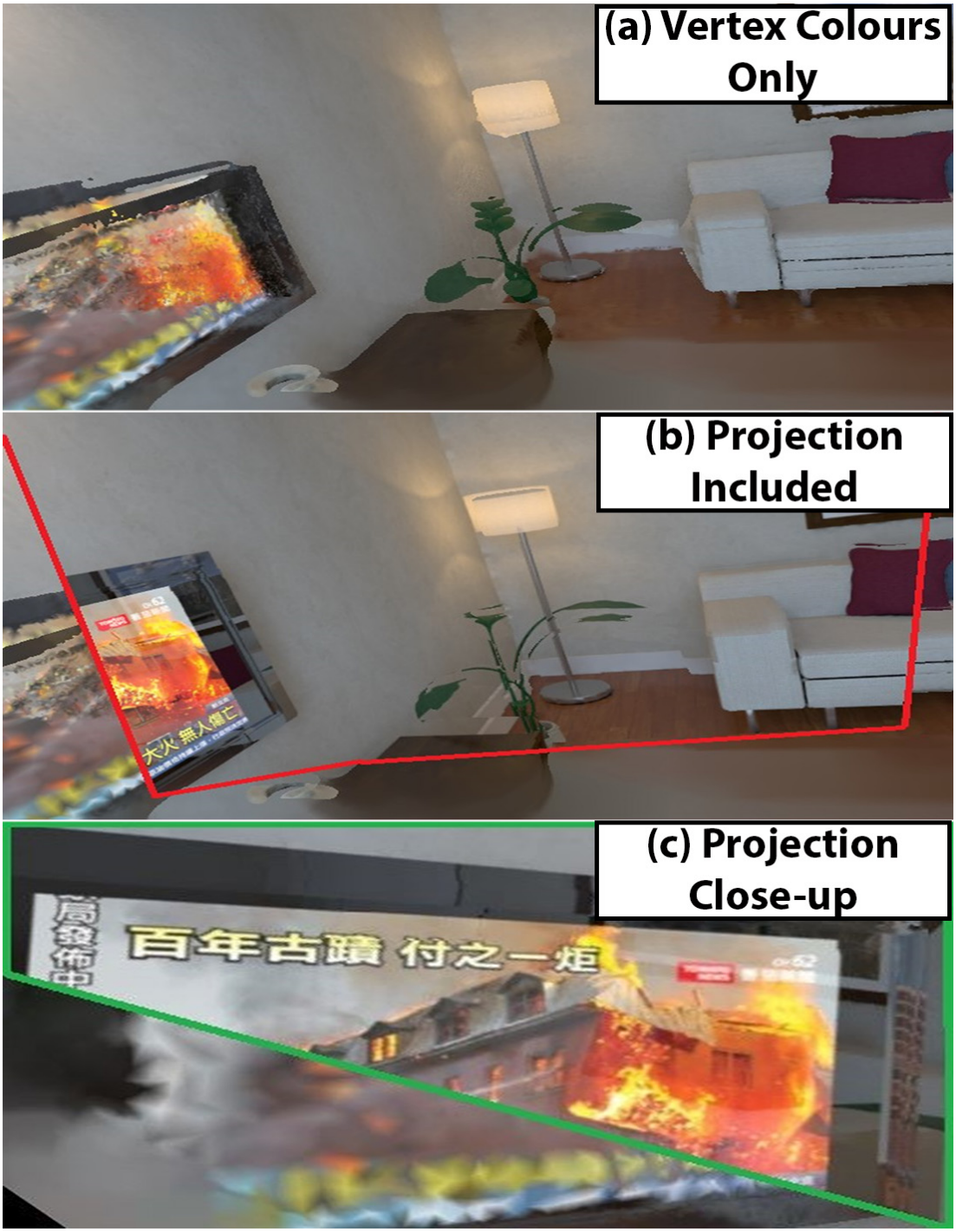


Figure 4.12: Image (a) is the 3D model using the weighted average vertex colours calculated by elastic fusion. Image (b) is a projection of a single image onto that 3D model. The projection is outlined using the red border. Image (c) is a close-up of the projection over the television screen. It highlights the difference in the reflective properties maintained in a single image (within the green border) but lost in the weighted average (vertices outside the green border).

The previous example of PTM required cropping of images because the same field of view was not available while attempting to look at variant angles around the projection. Here it was possible to project with more than a single image. When the current set of projectors could not cover some points in the camera's viewing frustum, the vertex colour provided by the SfM or SLAM could be used, see lines 4-9 Algorithm 3. This partly solved projector occlusion covered by past examples of PTM of 3D models [43].

An example of the output from this workflow with the above shaders can be seen below in Figure 4.12. The top image included the model using the colours of the vertices following the mesh reconstruction algorithm. The second image is the projection of an image onto the model, as highlighted in red to indicate the image boundaries. The third image was used to highlight the increased photorealism captured by the projection over the television screen, which included reflections of the room over the PTM region. Qualitatively lighting features such as the reflections over the television screen can be witnessed. This feature improves the photorealism of the 3D model. The reverse projection prevents projected images from being used over surfaces behind the projector, and this could distract users whose viewing camera is off-axis to the projector. The front-facing surface to the projector consideration has prevented projections of image colours onto surfaces behind the front-facing surfaces. Viewers observing points behind these projector front-facing points will no longer witness these artefacts.

4.2.5 Testing PTM with COLMAP Reconstructions

To evaluate the performance of PTM on the 3D reconstructions the following test was carried out: given the two video sequences of each road from the KITTI dataset, see description in Section 4.2.1, one video was treated as ground truth (how the road should appear) and the other video would be used to project an image onto the surface of the 3D model to simulate that ground truth image. Each render of the 3D model was captured with a pose that matched the ground truth image relative to the 3D model. The effect of using PTM was first compared to using vertex colours given by COLMAP. An example of the 3D models using vertex colours and PTM as their rendering methods can be seen in Figure 4.13.

Following this the effectiveness of each update to PTM (projection onto front facing surfaces only and removing shadow acne) was evaluated. To measure how closely each type of rendering matched the ground truth structural similarity index metric (SSIM) was used to compare the rendering with the ground truth.

Figure 4.13: 3D models rendered using vertex colours (above) and PTM (below).

Figure 4.14 highlights the SSIM score between the rendering of vertex colours and PTM against ground truth images. In all sequences the average SSIM was higher for the renders using PTM in comparison to vertex colours. Figure 4.15 highlights the SSIM between the shadow projection with varying bias and the ground truth images. A bias of 0 gives the lowest SSIM score in all cases. In almost every case 0.005 gives a SSIM score that plateau's for the values greater than that, of those tested. Larger values than those that were tested tend to render a projection that dis-regards the shadow projection. The rendering would

be very similar to the basic implementation of PTM and the scores highlighted in Figure 4.14.

4.3 Conclusions

Section 4.1 demonstrated the implementation and testing of PTM using basic shapes to approximate a 3D environment. It was shown that images could be warped using PTM onto these 3D shapes such that they could simulate perspective changes fitting other images of that same environment. This was shown using SSIM photo-consistency metrics. However manual manipulation of the pose of the projector was required. This implementation of PTM was construction using built-in OpenGL shaders and operated at 30 frames per second.

To address manual manipulation of the projector pose 3D reconstruction systems were employed, including Elastic Fusion and COLMAP, see Appendix C.1 and Section 4.2 respectively. Not only did this provide a good approximation of the projector pose, but it also gave an accurate 3D model to project onto. To integrate the poses of the projectors and address issues with PTM including reverse projection and projection onto non-front facing images, was a requirement to develop our own custom shaders. This not only addressed these issues but also enable a performance boost up to 60 frames per second on the same hardware.

In testing PTM was shown to perform better than vertex colours in terms of photo-consistency while rendering similar perspectives to the image used for texturing. The vertex colours of the final model are a weighted average of the images from different views projected into 3D space. This uses a Lambertian world assumption that 3D points reflect light equally in all directions. The material characteristics of 3D objects in the real world often do not follow this Lambertian world assumption. This reduces the photorealism of the final 3D model. This is evident in the SSIM photo-consistency performance of vertex colours in comparison to PTM, see Figure 4.14. The material characteristics of the 3D models are better reflected using projections of images from a similar perspective as the rendering camera. The pose of the rendering camera and the projector was a significant contributor to error in the PTM renders.

Shadow projection was initially proposed as a solution to the problem of pro-

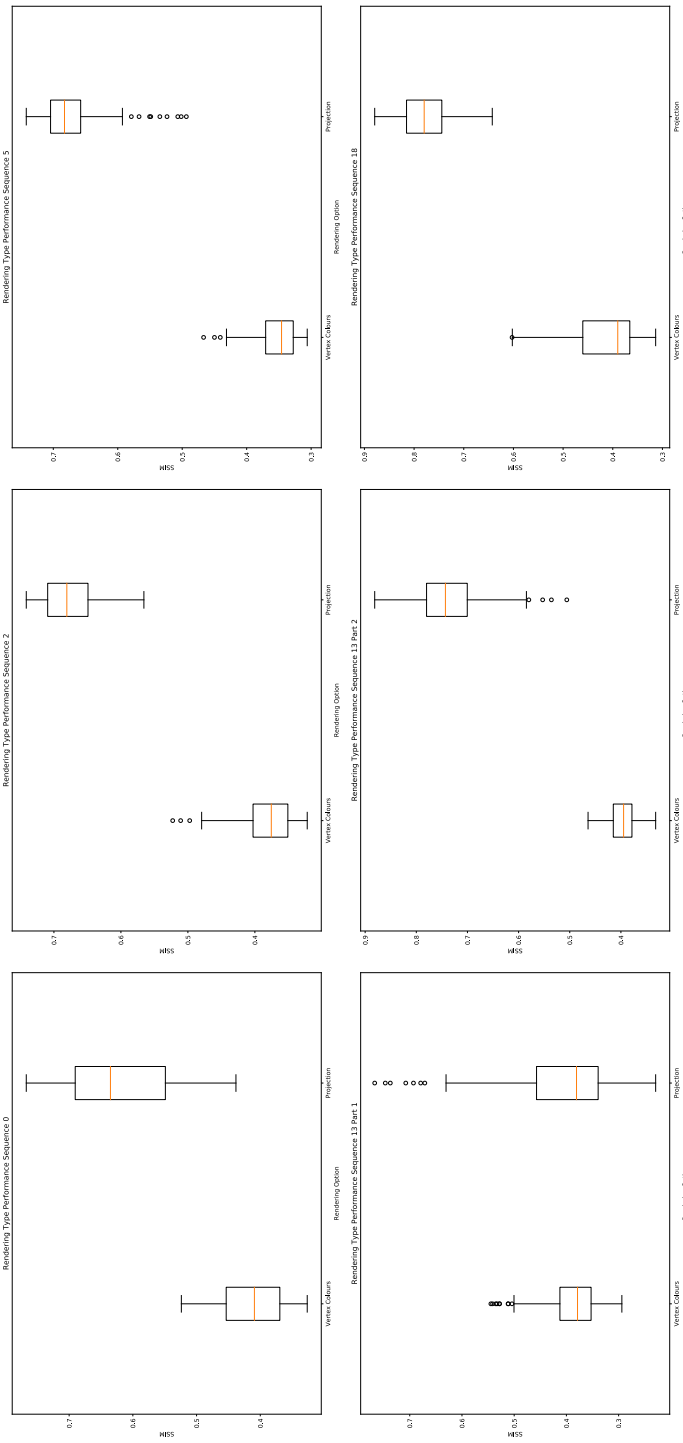


Figure 4.14: KITTI dataset video sequences reconstructed using COLMAP produced a model with vertex colours. This model was rendered with vertex colours and PTM. These renders were compared against ground truth using SSIM.

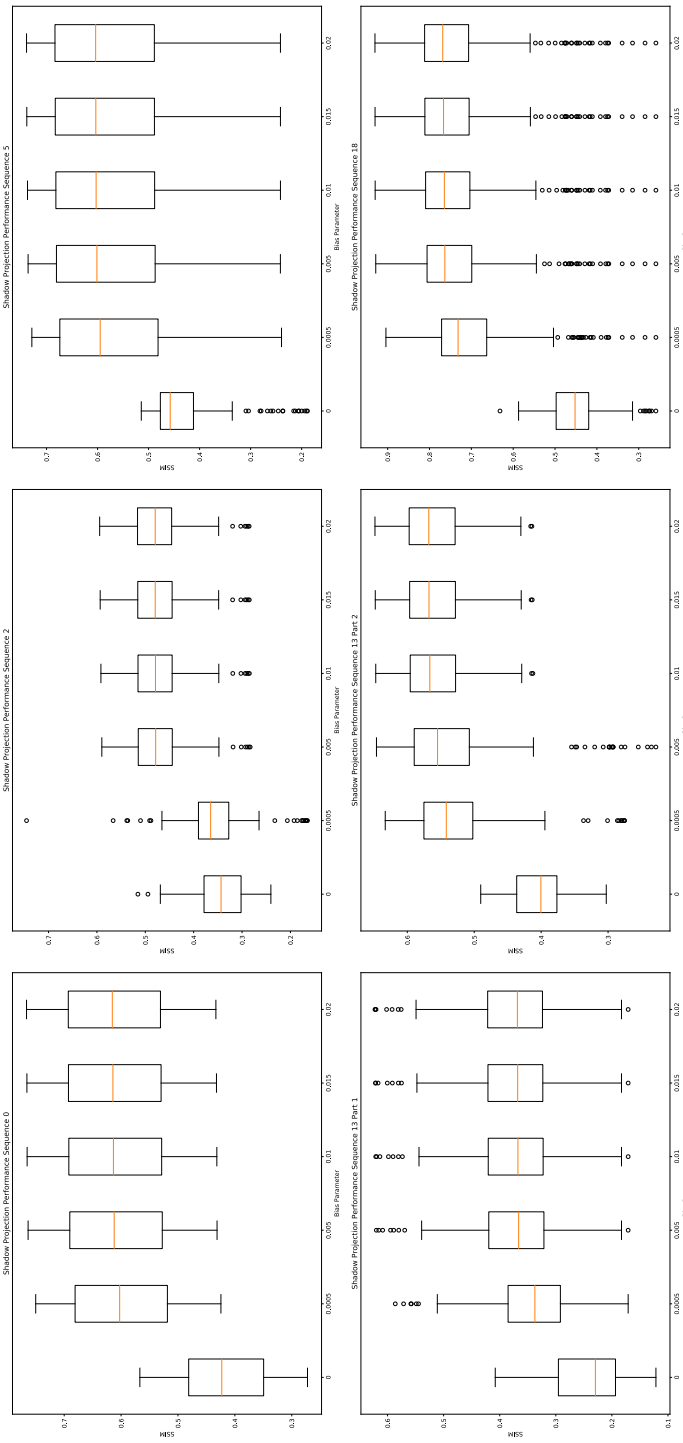


Figure 4.15: Comparison between ground truth images and the 3D models using shadow projection using SSIM. For each shadow projection a several bias parameters were tested $(0, 0.0005, 0.005, 0.01, 0.015, 0.02)$

jecting an image onto all surfaces, including those that were not front facing to the projector. However, after experimentation it was shown that photo-consistency does not improve if vertex colours are used in place of the projection, see Figure 4.15. To increase performance in this regard a projection of another image into this space may improve photo-consistency. The use of multiple projectors will be discussed in the next chapter.

Objects can enter the view of the images used to reconstruct the scene, this can increase photo-consistency error while using PTM. Due to the random sampling approaches used by MVS, transient foreground objects are often excluded from the weighted average colour of 3D surfaces. However, these transient objects will still be present in the images localized relative to the 3D model and inhibit the ability of PTM through reducing photo-consistency while simulating views of the road without a moving vehicle. This could also affect other rendering techniques, including NERF (neural radiance fields), to simulate photo-realistic views. "NERF in the wild" was proposed to address this [105]. However, the results using "NERF in the wild" for regions that contained foreground objects in many of the images of that space resulted in ghosting artifacts.

Another challenge with the 3D models produced using COLMAP was that it only reconstructs 3D surfaces within a threshold distance from the camera. Surfaces far away and points labelled as being infinitely far away, sky, are not reconstructed and leave gaps in the final model, see green regions in Figure 4.16. This can further detract from simulator fidelity, given there is no default rendering of sky or gaps in the 3D model that would make sense for all 3D models given the colour of the sky can change depending on the time of day or location.

The loading times can be excessive for large 3D models. Loading the model into memory presents a challenge when models are used with a simulator, as this can cause drops in frame rate. Fragments load the scene efficiently (smaller portions of a larger map). A single road segment is loaded at a time. Loading subsequent models are complete when the driver's viewing frustum approaches the model boundaries. After departing, the fragment can be released to free memory. Loading smaller models allows for smaller amounts of RAM and GPU memory to be required during render time. Reading from disk may occur more frequently, but the time between the beginning of a read and the end will be

Figure 4.16: Top: Images moving along a real road. Bottom: 3D reconstruction rendering of the same road from the estimated pose relating to the real image highlighting reconstruction gaps (green).

shorter. A list/queue data structure could apply to this problem if the user is driving along a straight road. However, if there is an extensive network grid of roads, then a bi-directional graph data structure may be required. A sparse 3D map was constructed to provide odometry to find the connection between fragments. The odometry can provide the transformations between fragments to ensure they are loaded using the correct orientations and positions relative to one another. This is discussed furth in in Appendix D.

Chapter 5

Removing Vehicles and Inpainting

This chapter discusses methods to remove foreground dynamic objects found in images but not represented in the final 3D model. There were two approaches considered for this. They both used semantic segmentation to localize vehicles within videos. There were multiple sequences of the same road captured on video. A calculation was done to find which sequence contained the minimum total area of dynamic foreground objects captured within the images within the video sequence. From this point, the two approaches diverge. The first approach used an image template outside the area being removed. This was used to find the corresponding area in another image in another video of the same road using place recognition. After this area was found, the pixel intensities found in the "mixing region" (the space previously occupying the dynamic object) were used to fill the hole left in the query image. The second approach used the final 3D reconstruction from COLMAP, as discussed in the previous chapter. After removing the foreground object, it was possible to render vertex colours of the 3D model or use projective texture mapping (PTM) from a separate image. The chosen inpainting image did not contain a vehicle at those exact 3D coordinates. The second method achieved better results with fewer seams than the first. The first method was included as this was the first iteration of a prototype that demonstrated the general principle of foreground object removal using multiple video sequences. This chapter finishes with a presentation of the results comparing the photo consistency of the ground truth scene to the rendered 3D model using vertex colours, PTM before the inpainting methods were applied, and PTM after the

inpainting methods were applied. These photo-consistency measurements were recorded sequence 2 in the KITTI dataset [63].

5.1 Introduction

As discussed in the previous chapter, moving (dynamic) foreground objects within a scene are not geometrically reconstructed in most structure-from-motion systems, such as COLMAP. Most structure-from-motion algorithms/pipelines use a static world assumption and attempt to find a common set of image features to match across images and estimate both the location of the camera and the matching points within 3D space [189, 139, 115]. For the features associated with moving objects in the scene, although these correspondences are often found between images, if they are relatively infrequent and cannot be coherently matched with the static world assumption, they should not be reconstructed as part of the final 3D model. Some algorithms have been developed to allow users to mask parts of the image set, such that the image features located at these masked coordinates would not be considered during the 3D matching and mapping process [139]. This solved the challenge of reconstructing the 3D model without the noise introduced by the moving objects. While using PTM, the raw images are used to texture this 3D model. Visual artefacts were introduced when the image and model did not match. This was due to the absence of the moving object in the reconstruction. See Figure 5.1 where this stretching across the 3D model has been labeled as "Projection Splatter". To solve this challenge, it was proposed to use image labelling techniques to locate the position of these objects within the image. Following object localization, an inpainting solution was applied to this area. This hid the presence of this foreground object within the texture.

The remainder of this chapter included two approaches taken to solve the removal of dynamic foreground objects. These methods will be discussed, and their performance will be presented and compared to other algorithms using both our datasets and a publicly available benchmark, KITTI [63]. Both approaches used semantic segmentation to label the location of vehicles in all images. However, the approach to inpainting varied. The first approach attempted to solve the problem based entirely on image matching/place recognition within video

Figure 5.1: Projection of dynamic object onto 3D reconstructed static background. Resulting in the dynamic object stretching across the 3D model.

sequences. The second approach used 3D reconstructions to localize where the camera was while capturing the image and used PTM to aid in the recovery of image data for texturing of the 3D model.

5.2 Inpainting: 2D Patch-Based Method

Three video sequences were collected of the Moyglare Road, Maynooth, County Kildare, Ireland using a dashboard camera mounted on a car. The following notation was used in the development of an algorithm that could process video sequences. The system would allow for multiple video sequences of the same road. A set of n cameras were used in this process, each camera $\mathcal{C}_i \in \mathcal{C}$, where $i \in [1 \dots n]$. Each camera explored the same scene but was recording the scene at different times. Camera's produced images I_i^t, where the subscript i denotes the identity of the camera, and the superscript t denotes the timestamp (frame number) of that image from the sequence.

5.2.1 Vehicle Localization and Image Selection for Inpainting

For each set of images produced by each camera $\{\mathcal{C}_i\}$, a corresponding semantically labelled image was produced using a convolutional neural network (CNN) [198, 199]. The network used for this task used a pre-trained U-net architecture including a Resnet-50 encoder and a Pyramid Pooling Module decoder [74, 197].

Figure 5.2: Input image and its semantically labelled counterpart.

This produced corresponding labelled images L_i^t, where i is the corresponding camera it came from and t is the timestamp, (timestamp number) of that image from the sequence, as the standard colour images described previously, Figure 5.2. In this figure, the semantic labels were represented using colour. For example, blue corresponded to cars, maroon-people, and green-trees/shrubbery. The labels were made more robust by employing tracking algorithms, see Appendix D.1.1.

Using the labelled images, it was possible to identify individual pixels corresponding to a given object, $L_i^t(x, y)$. The total area occupied by pixels with specific labels could be used to find which video sequences contained the fewest vehicles. Finding the sequence with the minimum area occupied by these objects would result in fewer instances where foreground object removal was required. Converting the labelled images to bit masks, $m(L_i^t(x, y))$, for all labelled vehicle pixels within all the sequence of images produced by camera $\{\mathcal{C}_i\}$ provided a method to find this "clearer" sequence. Calculating the sum of pixels corresponding to these labels provided the sequence with the minimum number of vehicles,

see Equation 5.1.

$$\mathcal{C}_p = \min_i \sum_0^t \sum_0^x \sum_0^y m(L_i^t(x,y)) \tag{5.1}$$

From Equation 5.1 the camera/video sequence with the minimum captured pixels associated with vehicles, $\mathcal{C}_p$, where p is the selected camera. From this point onward this was the camera that was chosen for foreground object removal. In the camera sequence $\mathcal{C}_p$ the pixel locations matching those marked as vehicles $m\left(L_p^t(x,y)\right)$ were removed, for all images, I_p^t. To replace the missing regions in I_p^t the other two sequences were queried and used for inpainting. To find a suitable image in the other sequences, the feature matching process bag of visual words was used (DBOW) [62]. This algorithm used ORB features [116] and identified for each image in $\mathcal{C}_p$ the corresponding top 10 closest images from the remaining sequences $\{\mathcal{C}_{n-1}\}$.

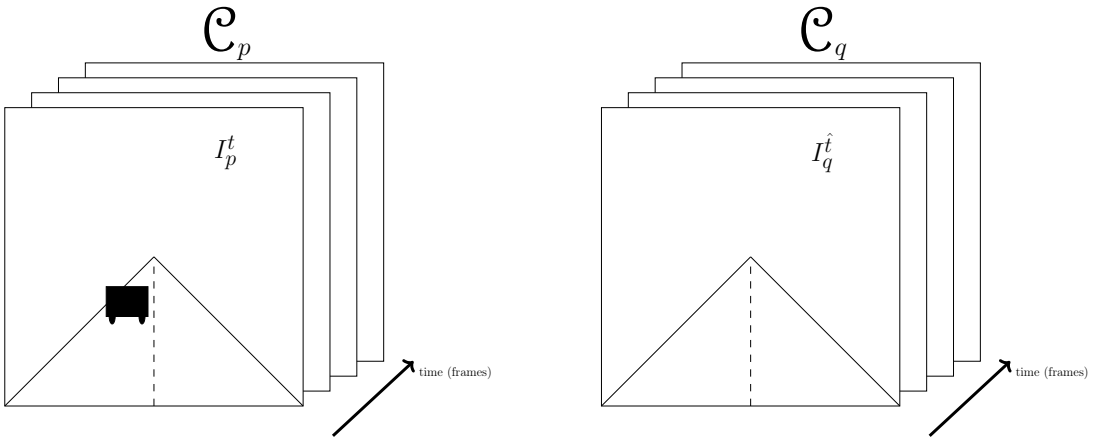


Figure 5.3: Background matching between camera sequences.

5.2.2 Inpainting the Missing Region

To find the replacement regions for each image in $I_p^t \in \mathcal{C}_p$ within each of the other sequences $\{\mathcal{C}_{n-1}\}$ the following algorithm was carried out. Given each image $I_p^t \in \mathcal{C}_p$ and each query sequence for camera $\mathcal{C}_q$, where $\mathcal{C}_q \in \mathcal{C}_{n-1}$, see Figure 5.3. All images were queried against each of the other sequences for those

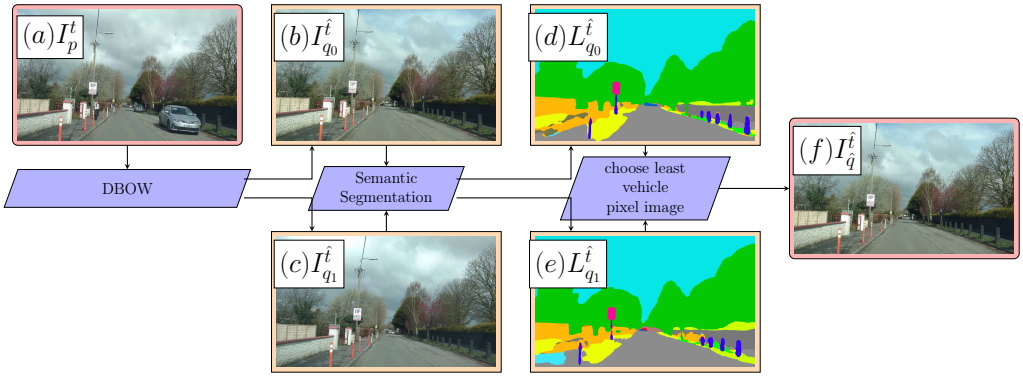


Figure 5.4: Workflow showing the selection of an image suitable for inpainting.

that contained the most similar visual features found in I_p^t. To find the top ten[1] most similar images DBOW was used and was stored in a list $\mathcal{I}$, that contained matching images from the other sequences $I_q^{\hat{t}} \in \mathcal{I}$. See Figure 5.4(a) that shows the query image and Figure 5.4(b) and 5.4(c) for examples of matching images. An additional condition was applied to this set of images, where only images that were chronologically ahead of that previously used inpainted image, could be applied. This was done by caching the frame number of the previously used image for this process, $I_q^{\hat{t}} \in \mathcal{I}$ where $\hat{t} \geq \hat{t}-1$. Following this, each of the 10 images, selected by DBOW, were semantically segmented using the labelling model from the previously mentioned CNN, see Figure 5.4(d) and Figure 5.4(e) for examples of semantically segmented images for Figure 5.4(b) and 5.4(c) respectively. This provided a new list where each element contained the labelled version of each image in $\mathcal{I}$, giving labelled images $L_q^{\hat{t}} \in \mathfrak{L}$. For each of the labelled images in this set the number of vehicle pixels was calculated. The image that was found to contain the minimum number of pixels corresponding to these labels was selected for inpainting, see Equation 5.2. This resulted in the final selected image to aid in inpainting, see Figure 5.4(f).

$$I_{\hat{q}}^{\hat{t}} = \min_{\hat{t},q} \sum_{0}^{x} \sum_{0}^{y} m\left(L_q^{\hat{t}}(x,y)\right), \forall\ L_q^{\hat{t}} \in \mathfrak{L} \tag{5.2}$$

[1]Ten images were queried but often only the top 5 were relevant. This number of images was selected through trial and error with this specific dataset

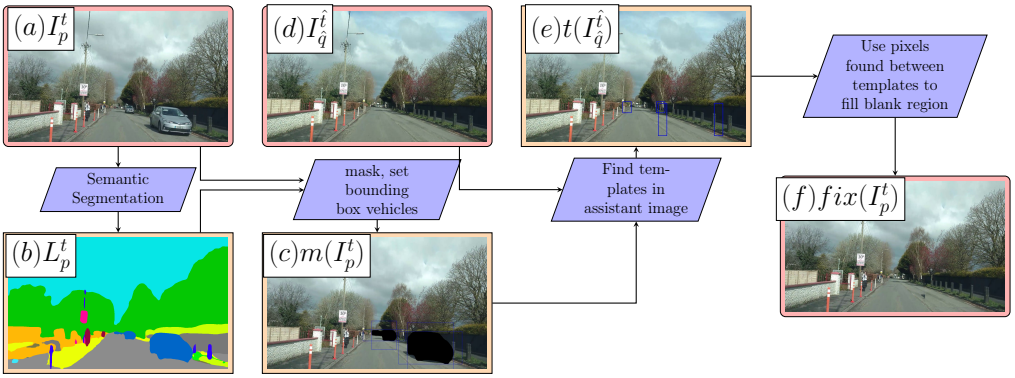


Figure 5.5: Workflow to replace car pixels given proposed matching images.

Following matching of the images $I_p^t \leftrightarrow I_{\hat{q}}^{\hat{t}}$, a template matching algorithm was applied to the regions of the image I_p^t surrounding labels corresponding to vehicles and compared to the image $I_{\hat{q}}^{\hat{t}}$. The labels were again acquired from the semantically segmented image, see Figures 5.5(a) and 5.5(b). Four blocked regions (above, below, left, and right) were selected that surrounded each cluster of pixels assigned the label of vehicle, see Figure 5.5(c). These templates were compared (by minimizing the squared difference of the image data using OpenCV) to the matching query image from the previous algorithm/workflow, see Figure 5.5(d) for the image it was compared against and Figure 5.5(e) that highlights where these templates found their matches. After these boundaries were located the pixel colour values from the query image were cut from the query image, $I_{\hat{q}}^{\hat{t}}$, and written to the image that required inpainting over the vehicles, I_q^t, see Figure 5.5(f). Rather than using the rectangular region, the semantic labels corresponding to vehicle were used as bitmasks to query where in the image the blending should occur. This ensured the minimum amount of inpainting was applied instead of inpainting a perfect rectangle.

5.2.3 Results

A qualitative assessment was carried out to measure the performance of the above workflows. The same masking and labelling procedures were applied for each test. Different inpainting algorithms were applied for comparison when it was required

Figure 5.6: Inpainting results for a single image from the Moyglare Road experiment.

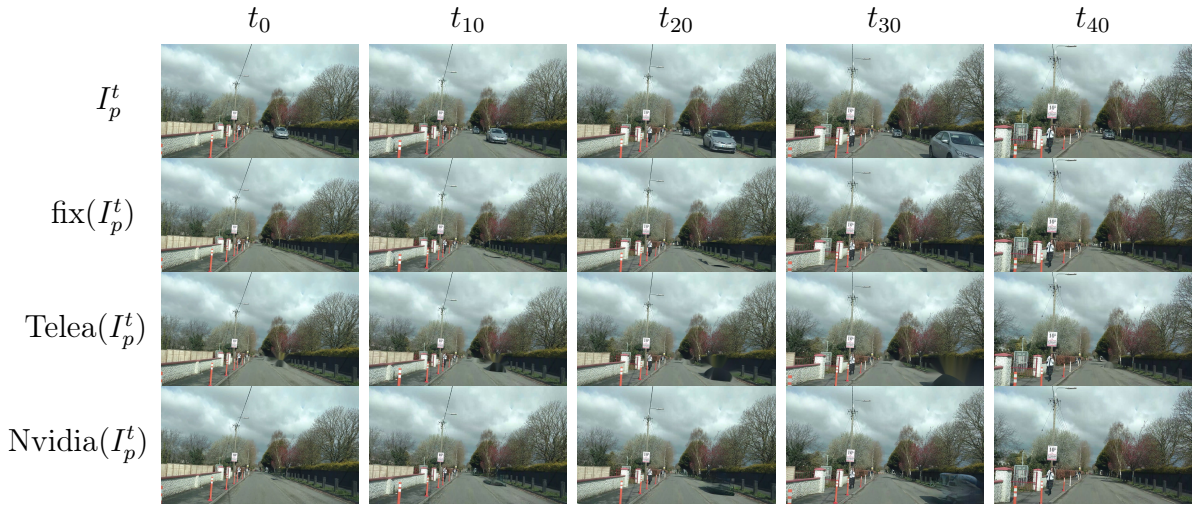


Figure 5.7: Comparison of inpainting over time for 5 frames separate 10 frame apart ($t_0 - t_{40}$). Each inpainting algorithm is applied to each image shown on column I_p^t. The algorithms fix(I_p^t) was the algorithm developed in this section. Telea(I_p^t) refers to an algorithm developed by Telea et al.[165], and Nvidia(I_p^t) refers to an algorithm developed by Liu et al.[97].

to find a suitable replacement for the pixels located over the masked region. A single example of our algorithm's performance can be seen in Figure 5.6. The results of this are seen over several images Figure 5.7. A comparison was made over 50 frames of video of the Moyglare road. Figure 5.7 shows these 50 images over intervals of 10. The input image to be inpainted is shown in the top row, I_p^t. The workflow output described in this section is shown in the row labelled $fix\left(I_p^t\right)$. The row labelled $Telea\left(I_p^t\right)$, was an example of a basic inpainting algorithm computed over the same masks using an OpenCV implementation of this paper [165]. The final row labelled $Nvidia\left(I_p^t\right)$ was an example of a more recent inpainting algorithm trained on a large image dataset using deep convolutional networks computed over the same masks [97].

For instances including very close matches supplied between two image sequences using the workflow described in this section, the boundary of the inpainted region was less visible. Visual artefacts, including shadows from the inpainted objects, proved difficult to remove. This can be witnessed as the vehicle being inpainted came closer to the capturing camera, see Figure 5.7 at timestamps $[t_{10} - t_{30}]$. Similar effects could be seen when using the other inpainting algorithms. In some cases, the camera misalignment between the matching images in the two sequences is evident; see Figure 5.7 at t_{30}. The vehicle on the right-hand side has been removed, but the image replacement has become visible due to the roadside curbs not aligning within the border where the image has been inpainted. The Telea algorithm appears to be the worst performing of the set tested. This algorithm uses neighbouring pixels to estimate how the masked region should appear. This often resulted in colour bleeding between large contours on the road; see the roadside curb in all timestamps in Figure 5.7 for the row labelled Telea $\left(I_p^t\right)$. The Nvidia sample performs exceptionally well for timestamps t_0 and t_{40}. The other timestamps perform poorly by comparison. This is because of the presence of the shadows of the vehicle. The network carrying out this inpainting procedure uses image context cues to estimate what should be in the masked area, and the shadow indicates that something else should be present on this part of the road.

5.2.4 Discussion

The workflow described in this section removes vehicles from videos at the expense of some visual artefacts. The artefacts are less extreme than those produced by some existing inpainting algorithms applied to the same masks (covering these vehicles). This workflow performs best when the videos contain fewer differences in terms of lighting and capturing angle. This setup was ensured by the design of the data collection. The same road was recorded multiple times, driving in the same direction, within the same lane, at a similar time of day (within a few minutes), and under similar lighting conditions (on an overcast day). Large deviations in the appearance of the scene, especially those located adjacent to bounding boxes of vehicles to be inpainted, can inhibit the performance of this workflow. Feature detection algorithms, including SIFT and ORB, could be used in future implementations instead of using template matching [99, 137]. These image features would be less sensitive to scale and orientation factors during correspondence matching. This could also allow features found while using DBOW to be used again instead of wasting computational resources on finding another set of image features. This approach relies on the time and space consistency of roads captured in video. However, it does this within the 2D image plane. The world being captured exists in 3D space. Image features and appearance can be consistently modelled using multiple view geometry concepts instead of attempting to match image features within a 2D space. The previous chapter highlighted the possibility of reconstructing 3D environments for view synthesis. Using a similar approach, a view can be synthesised for image inpainting. This would involve synthesising the view for the area being inpainted. The final technique will build on the existing discussion and include rendering vertex colours in the masked area of the image when mapping another image into that space is impossible.

5.3 Inpainting: Multiple View Texture Mapping

The previous section described how background and foreground objects could be separated such that only background pixels were present. Background pixels

were labelled in correspondence with the semantic labels assigned to them based on their similarity to vehicles, using a semantic labelling prediction network. A method was developed to identify videos containing the fewest foreground objects instances. This would approach produced a video that required the minimum amount of inpainting. Within this video, each instance of a vehicle was tracked and removed from the image. The pixels corresponding to these labels were masked. The masked area was used as a query to the database of images from other videos. Image blending was used to fill the gaps left after removing vehicles. 2D localization introduced artefacts, caused by differences in the appearance of the road due to perspective and projective relationships between the 3D geometry and the capturing camera. In this section, these artefacts will be corrected.

It was possible to reconstruct the 3D scene using multiple view geometry methods discussed in the previous chapter, via structure from motion and multiple view stereopsis [152, 139, 189]. Given a dense 3D reconstruction and localization of camera intrinsic and extrinsic parameters, it was hypothesised that a video of that scene could be inpainted using the vertex colours mapped within the 3D model. The 3D model IS a continuous representation of the set of views/images of the same scene. It does not reconstruct dynamically moving objects and should only map the static background objects, if anything. A method was developed to render background vertices from the same perspective as the image being inpainted. PTM allowed an image to be painted onto the 3D model. The algorithm developed provided a choice to render the entire scene using a combination of single images, parts of multiple images, and/or vertex colours. Using the same masking technique as Section 5.2 and developing the vehicle tracking techniques, see Appendix D.1.2, rendering the image's texture was possible when the mask corresponded to pixels not labelled as vehicles.

5.3.1 The Dataset

The dataset: A 3D reconstruction of the road and camera localization was used to aid the recovery of background colours required to replace vehicles within a set of images from the KITTI dataset [63], see Figure 5.8. Camera localization removed the requirement for template matching and place recognition (DBOW)

[62]). Neighbouring images used for inpainting were found using nearest neighbours identified using camera transforms (location and rotation). Another option was to query the set of 3D points visible for each camera. Those with the most overlap had the highest likelihood of appearing most similar.

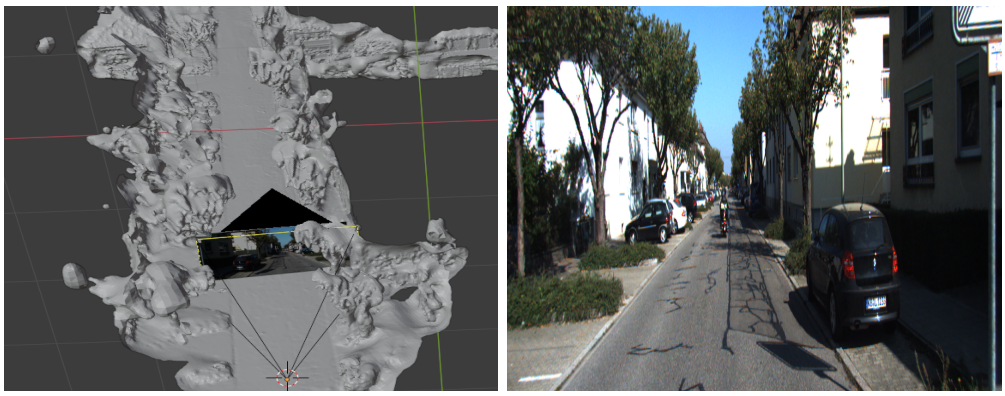


Figure 5.8: Left: 3D model of road with a localized pose. Right: the view from that pose with colours.

5.3.2 The Rendering Process

It was possible to ensure the semantically labelled images acted as a stencil while synthesizing the views of the road without vehicles. The labelled parts of these images were masked over the 2D image plane. As 3D points were transformed from 3D space to the viewport of the rendering camera, they underwent projection onto the 2D image plane of a texture. In instances where the 3D points were transformed outside the image's bounds, the vertex colours were used. If an image is available for inpainting (inside the bounds), this was used for texturing. A labeled mask corresponding to vehicles can be applied to conditional rendering. Texturing from this image is only used when found outside the bounds of the masked parts of the image. An example of this concept can be seen in Figure 5.9. The top image was the original, including a motorbike projected onto a 3D model. The middle image highlighted the boundary labelled as a vehicle and masked. The final image showed how the motorbike could be replaced in the original image by using a projection of an image that did not contain the motorbike.

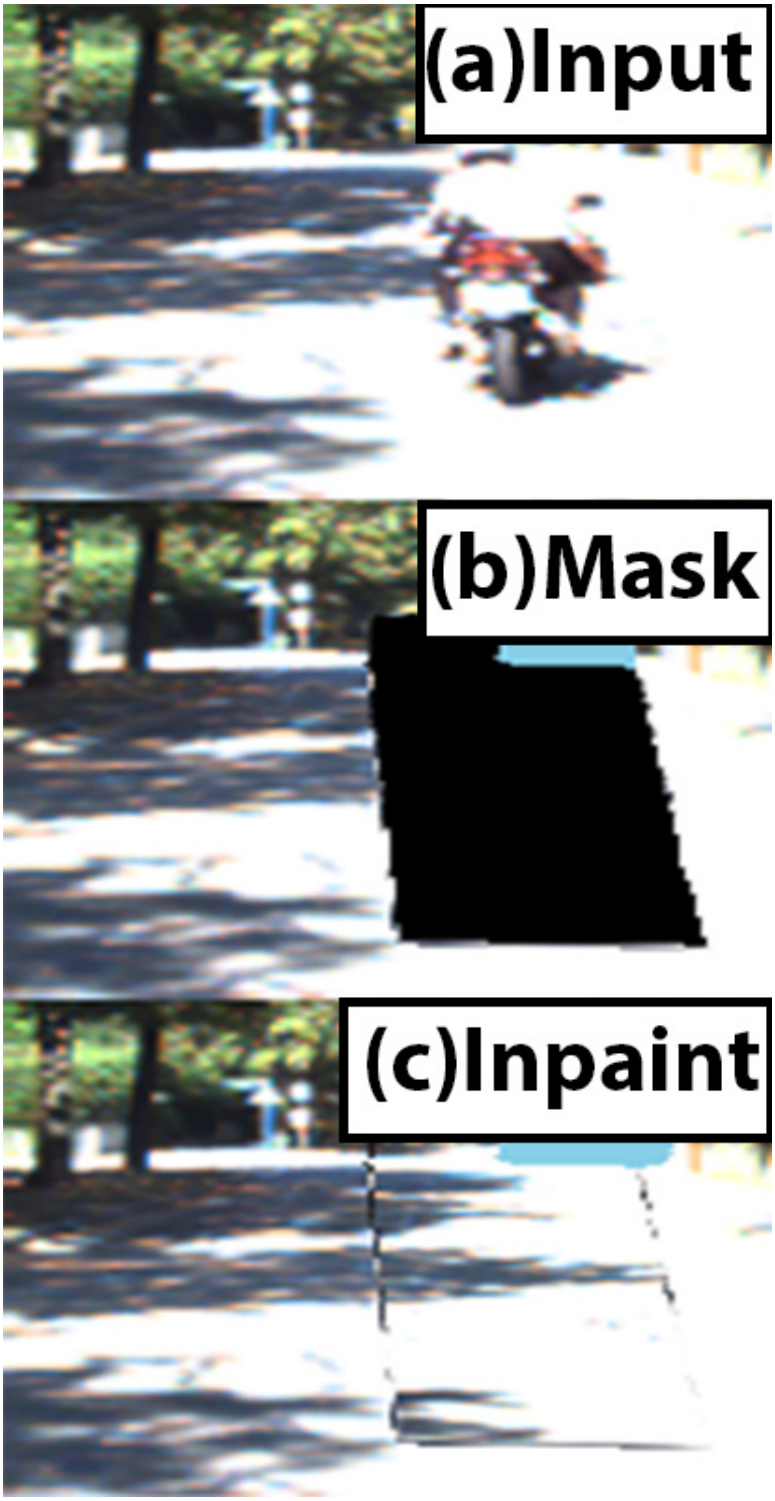


Figure 5.9: Inpainting using the 3D model. (a) The image to be inpainting before PTM. (b) PTM and Mask applied to image from (a). (c) PTM applied to the masked area from (b).

The rendering process was adjusted from the description in the previous chapter to enable the use of semantic labels in combination with PTM. An additional condition that checks for the presence of a vehicle masked pixel would need to be considered. Vertices would only be projected using that texture if this was not true. This was done in real-time using a shader, and the vertices must pass through shader programs regardless of the requirement of inpainting. The additional condition and inclusion of a texture mask do not take significant time during runtime, assuming these masks already exist. In these instances where the mask covered some set of vertices, another image projection or set of vertex colours should be considered. Testing was done to measure the performance of both approaches. The shaders to make this computation will be presented in the next section.

5.3.3 Theory & Implementation

Given the map $\mathcal{M}$, the ego-motion (odometry) of the camera $\mathbf{P}_t$, and a semantically labelled set of images L_t, with the goal of rendering the road without moving vehicles, from the perspective of the capturing camera. Two videos were used to capture the same scene from similar perspectives. One video contained moving vehicles and the other did not. View selection for matching the images between sequences was based on the minimum Euclidean distance in $\mathbb{R}^3$. This used the camera pose $\mathbf{P}_q$, the pose of the camera that required the moving vehicles to be removed, and $\mathbf{P}_r$, the pose of the camera that did not contain the obstructing vehicles within its captured image. There existed a semantically labelled image for each camera pose and this was used as a mask to remove the parts of the image corresponding to vehicles. The vehicle masks used white pixels to represent vehicles and black pixels to represent everything else, see Figures 5.10a and 5.10b. Perspective projection was applied to the bitmask in 5.10b to find what 3D points corresponded to the vehicle labelled pixels in Figure 5.10a, see Figure 5.10f. It was possible to texture from another image not containing the vehicle, Figure 5.10c, or the colour contained within the 3D model's vertices, Figure 5.10d.

A function was created to check the intersection of the image colours/texture,

(a) C_q

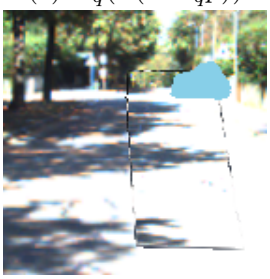
(b) L_q

(c) C_r

(d) p_{Colour}

(e) $C_q(\pi(\mathbf{KP}_q p))$

(f) $\bar{L}_q(\pi(\mathbf{KP}_q p))$



(g) $f(L_q, C_q) + f(\bar{L}_q, C_r)$

(h) Fig 5.10g with skybox

Figure 5.10: Rederings of each of the transformed images used for inpainting. a: is the image to be inpainted. b: is the bitmask used to label the bounding area of the vehicle to be inpainted. c: is the image used to aid inpainting. d: is a rendering of the 3D model $\mathcal{M}$ from the same perspective as c. e: is the image a following PTM onto the model. f: is the bounding area of the vehicle mapped onto the 3D model. g: is the two images a and c projected together onto the model. h: is the same as 5.10g with a skybox added.

$\mathbf{C}_q$, projected onto the 3D model and the projection of the bitmask, L_q, this function had the form $\bar{L_q}\left(\pi\left(\mathbf{KP}_q\mathbf{p}\right)\right)$, this produced Figure 5.10f. In summary, the image that contained the vehicle was first mapped to the model through a projective transformation, see Figure 5.10e. The semantic labels in the bitmask were not rendered during projection, see Figure 5.10f. The projection of the image without the vehicle was then applied to the region that previously contained the vehicle, see Figure 5.10g. A function that accepted two parameters was used to generalize this process, $f\left(\cdot,\cdot\right)$. The two arguments included a semantically labelled bitmask and a colour image that had been used for PTM. The colour image texture would be used for PTM if and only if the bitmask contained positive values on the intersected surfaces of the bitmask while it was also projected. This function could be generalized to what is contained within Equation 5.3[1]. The subscript e and w is just used to refer to the first and second arguments i.e. each texture being passed to this equation will already have a corresponding pose, $\mathbf{P}$, and the subscript is used to match one another.

$$f(L_w, C_e) = L_w(\pi(\mathbf{KP}_w p)) \oplus C_e(\pi(\mathbf{KP}_e p)) \tag{5.3}$$

The shaders from the previous chapter for implementing PTM were extended to cover these additional cases, see Algorithm 4 and 5. The vertex shader included the intrinsic and extrinsic parameters for all projectors used during PTM, see $\mathscr{P}$ and $\mathscr{K}$[2] respectively on line 1 of Algorithm 4. The rendering camera parameters were also required, M_{cam}. The outputs of the vertex shader now included texture coordinates for all image projections, see the list of $vTexCoord_i$, where $i \in [1, \ldots, n]$, on line 2 of Algorithm 4. The vertex colours, and the position of the vertex after passing through this shader while using the model-view-projection matrix based on the rendering camera parameters. Algorithm 4 contains the same basic logic as Algorithm 1, with the exception that texture coordinates were collected for a set of projectors, see lines 4-12 where a loop was used to collect these coordinates for all projector parameters passed to the shader. The remainder of the shader code is the same as Algorithm 1.

[1] $\oplus$ is the exclusive or function

[2] This is not a typing error. $\mathbf{P}$ and $\mathbf{K}$ were previously used for a single projection. These new letters represent lists of this same types of matrices.

Algorithm 4: Dual PTM (Vertex Shader)

Result: Vertices mapped to projector and camera coordinates.

1 **Input**: $\mathbf{p}$, $\mathbf{M}_{cam}$, $\mathscr{P}$, $\mathscr{K}$, $\mathbf{c}$ $\mathbf{n}$;
2 **output**: $\mathcal{T}$, glPosition, vColour;
3 i = 0;
4 **foreach** $\mathbf{P}_{proj}, \mathbf{K}_{proj} \in \mathscr{P}, \mathscr{K}$ **do**
5 $\quad p_{texeye} = \mathbf{P}_{proj}.\mathbf{p}$;
6 $\quad$ **if** $p_{texeye}[z] < 0$ **then**
7 $\quad\quad p_{texndc} = \mathbf{K}_{proj}.p_{texeye}$;
8 $\quad\quad vTexCoord = \frac{p_{texndc}}{ptexndc[z]}$;
9 $\quad$ **else**
10 $\quad\quad vTexCoord = vec4(inf, inf, inf, 1)$;
11 $\quad$ **end**
12 $\quad \mathcal{T}$.append($vTexCoord$);
13 $\quad$ i++;
14 **end**
15 vcolor = $\mathbf{c}$;
16 vNormal =$\mathbf{n}$;
17 glPosition = $\mathbf{M}_{cam}.\mathbf{p}$;

Algorithm 5 was developed to implement Equation 5.3. The semantically labelled image corresponding to the image to be inpainted was used to find where within the texture the model should be mapped, ignoring vehicle pixels. In instances where these labels overlapped a fragment on the image plane, another projection was used instead of the initial image. To accomplish this the texture coordinates from the vertex shader were received, $\mathcal{T}$, and its corresponding texture to sample from was passed as a uniform to the shader in an ordered list of textures, $\mathscr{C}$. The first image in each of these lists corresponded to the image to be inpainted. A Boolean variable was created to differentiate between the image to be inpainted and the other images within the set used for PTM. These lists were iterated through see lines 4-20 in Algorithm 5. The calculations used to find 3D points within the clipping coordinates and front facing according to the shadow calculation were queried on line 6. If the current image within the loop was not the first image in the list then the texture was sampled from their corresponding image pixels, see lines 8-10. Otherwise, if the image is the first image in the list

Algorithm 5: Dual PTM (Fragment Shader)

Result: A rendering calculation for each pixel on the viewport

Input: vColour, $\mathcal{T}$, $\mathcal{C}$, L_t, $\mathcal{K}_t$, $\mathbf{P}_t[t]$, vNormal;
output: oColour;
notFirst = False;
foreach $vTexCoord, C \in \mathcal{T}, \mathcal{C}$ **do**
    projCoord = vTexCoord[xyz];
    if *ClipCoords(projCoord)* ***and*** *ShadowCalculation(projCoord,* $\mathcal{K}_t$*, vNormal,* $\mathbf{P}_t[t]$*, vTexCoord)* **then**
        projCoords = (projCoord ×0.5)+0.5;
        if *notFirst* **then**
            oColour = texture(*C*, projCoords[xy]);
            break;
        else
            if *texture(*L_t*,projCoords[x,y])==0.0* **then**
                oColour = texture(*C*, projCoords[xy]);
                break;
            end
        end
        notFirst=True;
    else
        oColour = vColour;
    end
end

and the semantic labels did not correspond to a vehicle then the texture could be sampled from its corresponding image, see lines 11-15. In the case that no image in the set could be mapped to any texture due to clipping space or shadow calculations then the vertex colours were used, see lines 18-19.

There was an additional challenge which had not been previously considered. Although COLMAP produced a dense 3D model, there remained instances where the model contained "holes". These were instances where there was no 3D point mapped for some region in the map $\mathcal{M}$ when observed from some subset of perspectives $\mathbf{P}_t$. This could be due to a small number of observations for some real 3D points such that a consensus could not be met. It could also be because the points were too far away from the capturing camera to make an appropriate estimate. Whatever the reason, this resulted in perceived holes in the 3D model, see the blue region near the center of Figure 5.10g. This issue was identified within the Apolloscape inpainting dataset and was addressed by using smooth colour interpolation [95]. The work presented within this thesis solved this issue by implementing graphics technique known as skyboxes/cubemaps [178]. This involved creating some set of far planes that would surround the map $\mathcal{M}$, and existed at a far distance from the set of projected images $\mathbf{P}_t$. In instances where the local depth buffer of an image rendered from the perspective $\mathbf{P}_t$ of the map $\mathcal{M}$ and it contained no corresponding depth, the projected image would be mapped to the far planes rather than the model, see Figure 5.10h. The planes remained orthogonal to the optical plane of the camera to prevent skewing. The overall result was that holes would be removed from the projection, see the difference between Figure 5.10g and 5.10h.

5.3.4 Results

To evaluate the performance of this approach, a comparison was carried out using a combination of projections against the ground truth. An image was deemed ground truth if it was some image of the model of the road $\mathcal{M}$, without a vehicle. For each image containing a vehicle to be inpainted C_q another image was found that had a close proximity and pose but did not contain the vehicle, C_r, see Appendix D.2 for details on how this was selected. This image was also used

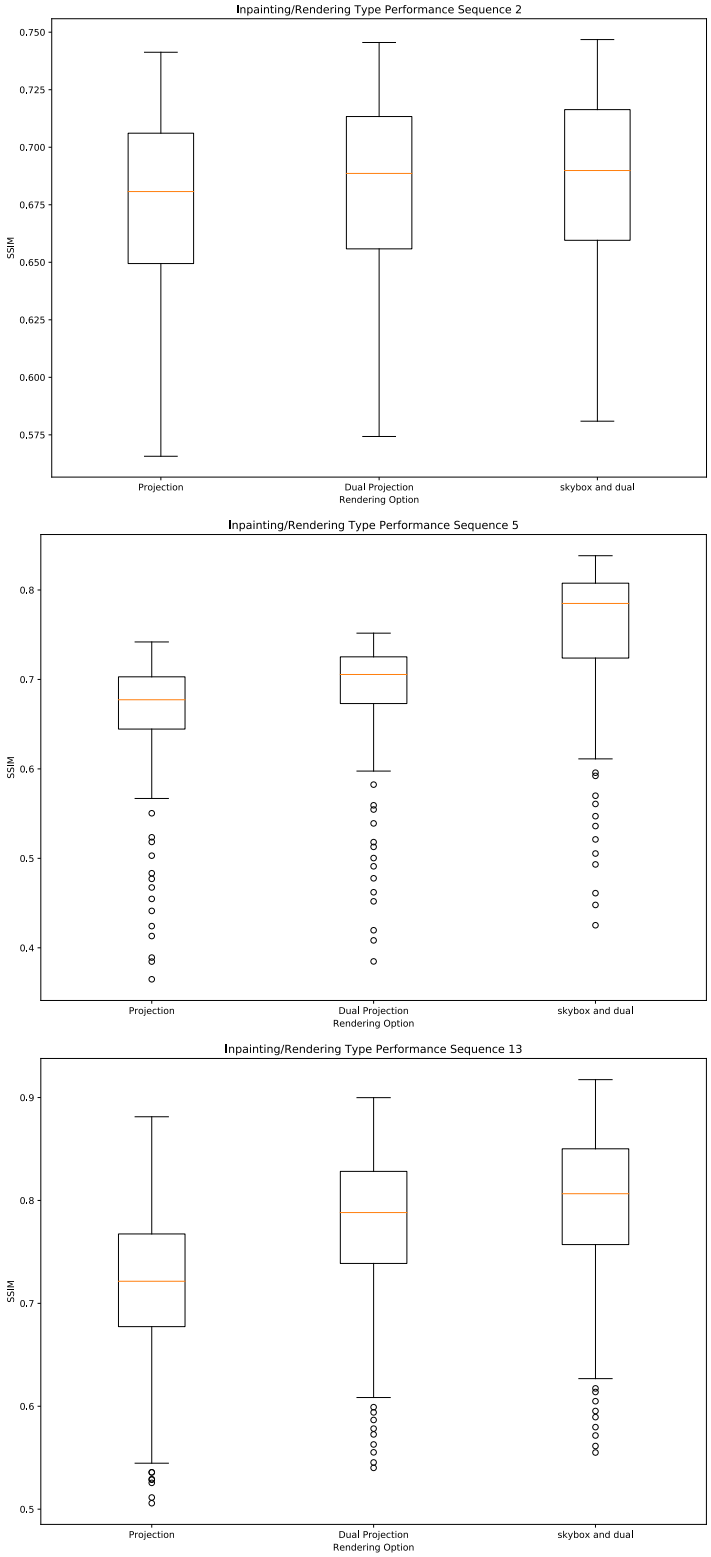


Figure 5.11: Structural similarity index metric (SSIM) for each rendering option, PTM ("projection"), PTM with inpainting ("dual projection"), and PTM with inpainting and a skybox ("skybox and dual")

to project the texture for the parts where the vehicle appeared in the other image. Following the rendering process described in Figure 5.10g. The rendering of this image was constructed from a camera pose matching the image C_r, $\mathbf{P}_r$. The photo-consistency metric used for comparing the images was structural similarity index measurement (SSIM), see Figure 5.11. An average score was recorded for each metric in three video sequences from the KITTI dataset, including sequence 2, 5, and 13. The PTM technique provided better photo consistency than vertex colours in all sequences in the previous section. The projections that used inpainting performed better than those that did not for PTM and the introduction of a skybox performed better again.

5.4 Conclusions

This chapter focused on removing foreground dynamic objects, including vehicles. The initial implementation was completed on a real dataset based on video footage collected from Moyglare Road, Maynooth, Co. Kildare. A qualitative performance assessment was demonstrated in Figure 5.7. On reflection it was decided that a more robust approach was required. This was due to visual artefact because of the following:

- Misclassification of semantic labels in a small percentage of images.
- Misalignment between the image being inpainted and the region blended from other images.
- Inconsistencies related to matching images based entirely on the presence of image features while using DBOW and not considering their scale within the image. This used temporal consistency and did not consider geometric consistency.

To address these challenges, an inpainting approach was developed that was aided by 3D reconstruction. A 3D model of a real road from the KITTI dataset was reconstructed using COLMAP. COLMAP considered the entire set of images during reconstruction following image feature identification and matching. It used multiple view geometry techniques to triangulate 3D points and bundle

adjustment to update the 3D model and image pose alignment based on reprojection error (a reprojection error loss function). Low reprojection error had a positive correlation with good PTM performance, as was also observed in Appendix C.1. This made COLMAP an ideal system for 3D reconstruction in this case.

The overall performance of this inpainting approach developed in Section 5.3 was measured using ground truth images. This included images without a foreground object projected onto the 3D model. It was noted that projections including and excluding foreground objects had higher photo-consistency than vertex colours rendered on the 3D model. There was a higher photo-consistency performance in the inpainted solutions applied to PTM compared to the unmodified images. The solution presented here poses a possible solution to 3D model obstruction from dynamic objects for other view-dependent texturing [24] techniques or view synthesis approaches using data driven models [111, 195]. This chapter offers a possible solution to rendering skies/holes in 3D models, which improves photo-consistency metrics.

Chapter 6

Conclusions

6.1 Discussion

This thesis presented a novel driving simulator that used reconstructed 3D models of real roads and view-dependent texture mapping to increase photo-realism. The principal contribution of this thesis was the synthesis of computer graphics, machine vision, and human testing technologies to present a driving simulator as a photo-realistic experience. The use of projective texture mapping (PTM) in combination with 3D reconstructed models, and localized camera poses has provided a method of:

1. Improving the photo-realism of a 3D model through PTM in place of vertex colours.
2. A view-dependent rendering of the 3D model.
3. A method of inpainting/background subtraction using multiple-view texture mapping.

The requirement-gathering phase was carried out for the development of the driving simulator, see Chapter 3. This included human trials with video and asset-based driving simulators. The benefits of photo-realism from the video-based simulation were identified as a desirable trait for the simulator developed in this thesis. The control over the asset-based driving simulation was also appealing. To gain the best of both worlds a photo-realistic 3D model would have to be

constructed and sets forth the requirements for the remaining sections. As well as this, some observations on driver distraction were made. The video-based driving simulator, although photo-realistic did not contain discrete distracting events to measure against. However with the use of clustering of eye tracking data, large shifts of attention were highlighted across groups of people to items on the side of the road. To measure how distracted people were; discrete distracting events had to be set up. Using the asset based driving simulator this was accomplished by creating digital and static billboards. The degree to which someone would be distracted was dependent on the activity of the distractor itself. This was shown through animations (dynamic billboards) causing more higher levels of distraction when compared to static billboards, see Section 3.2.5.

The driving simulator's 3D model and PTM rendering process included two main iterations, see Chapter 4. Initially, the 3D model was represented as an open-ended box to simulate a hallway and tunnel. This provided a basic structure to test an implementation of PTM. This iteration highlighted the feasibility of PTM for producing multiple photo-realistic views using a single image. This was demonstrated using structural similarity index metric (SSIM) on real images compared to the synthesized views, see Section 4.1.2. The second iteration used COLMAP to reconstruct 3D models from multiple RGB images from the KITTI dataset. COLMAP used structure-from-motion that localized images relative to the 3D model with a loss function that optimized toward correcting the re-projection error of the entire set of images relative to the triangulated 3D points. This made it a more suitable system when combined with PTM, because PTM performs better when the reprojection error is lower, see Appendix C.2.1. PTM in combination with 3D modelling has been demonstrated to provide greater photo-consistency than vertex colours using SSIM on the KITTI dataset for several routes, see Section 4.2.5.

During development, it was noted that removing foreground dynamic objects were required to marry 3D reconstructions from COLMAP and PTM. Initially, these objects were removed, and background pixels were found using place recognition over multiple video sequences to find a matching image. Template matching within matching images was used to localize the area in the query image that required inpainting. This was demonstrated on an in-house dataset of video se-

quences of Moyglare Road, Maynooth, Co. Kildare, Ireland [32]. This approach produced visual artefacts when the two images did not align. Qualitatively these results were promising but it was difficult to gauge quantitatively how well this approach performed, see Section 5.2.3. Using the KITTI dataset [63] another approach was taken to correct this misalignment using 3D reconstructions to localize the images. The localization of the images removed the need for place recognition. It was then possible to carry out inpainting using PTM from multiple views. Using videos from KITTI where the roads were traversed multiple times, ground truth images, without vehicles, could be used to compare against images containing vehicles and those same images with the vehicles removed. It was demonstrated that using inpainting could again improve the photo-consistency of 3D models while using PTM, see Section 5.3.4. In this same section, it was also shown that skyboxes in combination with PTM could serve as a way to fill gaps in 3D models left by 3D reconstruction algorithms. This in turn increased the photo-consistency. Background such as skies and far away objects can often be a source of inconsistency for high performing view synthesis algorithms including NeRF [111], the solutions presented here with skyboxes may present a potential solution to these problems.

6.2 Future Directions

Initially, the work done in this thesis set out to contribute to the research in the area of driver distraction. Specifically distractions external to the vehicle. The major contribution of this thesis was the construction of a photo-realistic driving simulator. The driving simulator highlighted PTM as its principle rendering approach. Rendering using PTM in combination with 3D reconstructions has been fully formalized and implemented. However approaches to selecting a single or multiple views for texturing, filling in gaps in the 3D model not captured in a set of images used for texturing, and blending multiple textures seamlessly over the 3D model remain open problems. Without research in these areas, visual artifacts will likely reduce the photo-realism of this rendering technique in some instances. If it is intended to simulate a person driving down a road in a similar trajectory as the camera used for collecting the images used for texturing,

then this simulator will perform exceptionally well. For larger deviations from this trajectory, further research into the aforementioned areas will need to be considered. Some general approaches to view synthesis may also be applied, most notably Free View Synthesis [132], or NeRF [111, 195, 105]. View selection and gap filling for textures can likely be solved through optimization techniques. There is a lot of similarity between the view selection problem and the solution proposed by NeRF for view synthesis. For gap filling, this is similar to the area of inpainting, which has seen great strides in the last few years through deep learning [121, 177, 97]

The driving simulation presented in this thesis provides the required functionality for a driving simulator. There are several directions in the work that could be extended:

1. Experimental Applications:

   (a) Extend current software to include augmentation of models with distractors. Additional 3D models/assets can be added afterward. This would be similar to the static and dynamic billboards with timed animations in Chapter 3. Road engineers' main distractors of interest are roadworks, street art, and advertising.

   (b) The same 3D model can be used for simulation with multiple recordings of the same road from different times of the day. PTM could compare night-time vs. daytime driving by texturing with night-time or daytime videos, respectively [181].

   (c) Vehicle odometry collected to construct the 3D models and describe the motion of the acquisition vehicle could be used to simulate automated driving. This could be used for human-in-the-loop experiments where drivers must take control back from the vehicle. This is often seen in partially/semi-automated driving vehicles [108].

   (d) Using the PTM inpainting techniques in Chapter 5, it would be interesting to create a dataset of videos free of vehicles to benchmark data driven models for inpainting. Often there is an issue with finding enough data to enable this, but this technique may help. This would

then in turn remove the requirement to reconstruct a full 3D model for inpainting.

2. Technical Advances:

   (a) Higher resolution data for more photo-realistic driving simulation. KITTI images have a resolution of 1280×384 [63], and the Moyglare Road dataset had a resolution of 1920×1080 [32].

   (b) Improved vehicle motion model as the current implementation uses arc-ball single velocity pedal. Advanced models would include the effects of inertia, braking, and acceleration. The steering could be extended to Ackermann, see Appendix B.

   (c) Real-time integration of eye-tracking for dynamic simulator response. For example, part of the scene changes its state in response to eye gaze. See Appendix B for an initial description of the eye-tracking to model calculations. This would give some insight into the timing of human vision while driving.

   (d) PTM can be connected to eye-tracking. This allows eye gaze to be mapped from screen coordinates to the 3D model and back to images used for reconstructing the 3D model. This would allow for new forms of eye-tracking analysis not previously considered.

   (e) The design of this simulator's rendering identifies the transformations required to map between the viewport, the 3D model, and images used for PTM. It was noted that labelling images for each participant's perspective were required to carry out eye tracking and area of interest analysis in Chapter 3. A mapping of eye gaze to semantic labels could be developed using these transformations.

   (f) Training a small convolutional network to remove artefacts from renderings using PTM. Given the previous tests in Section 4.2.5 and 5.3.4 there is a way to compare the renderings to ground truth. A small convolutional network may be able to filter the remaining artefacts (projection onto non-front facing objects or poor shadow projection) through training.

6. CONCLUSIONS

This thesis was completed with the intention of providing a method to analyse distractions outside of a moving vehicle. As a result a driving simulator has been constructed. This required merging of methods and techniques from the fields of driver distraction and simulation with the computer graphics and vision. It is hoped that as a result of this research a roads engineer may gain insight in the levels of distraction present on real roads, without having to put people in harms way.

Appendix A

Tool Details

A.1 Eye Tracking

This section is a discussion of eye-tracking, including the methods used to acquire, process, and interpret eye-tracking data. The ability to map eye tracking data from different participants back to a common frame of video used to construct the scene is developed. This mapping process is a significant contribution to the work carried out in this thesis, see Chapter 3.

Eye-tracking will be the method chosen to measure reactions and gauge the driver's perception within the simulator. Eye-tracking is the process of observing and reporting the state of an eye at a given time. The main components considered for tracking are the pupil and iris, located behind the cornea and in front of the eye's lens. The pupil is the part being tracked by most eye trackers while attempting to gauge where the eye is looking. Rays of light are radiated through the eye's pupils and absorbed by the rod and cone cells at the back of the eye. A person's observable range of vision is captured based on the light allowed to travel through the eye pupils. Other items people perceive outside of this range have often been observed previously within the range. The perceptive visual systems could also estimate the presence of these items. The iris, in relation to the pupil, is tracked to gauge how much light enters the eye at a given time. The pupils' dilation is associated with engagement and emotions towards the item the person is observing. However, this reaction could also be a result of the change

in brightness of the room. This factor is why pupil dilation is not considered an essential factor in the experiments within this thesis. Specific lighting settings may be required to control the eye's reaction. This research focuses on the point at which the eyes are looking, otherwise known as the point of regard. Eye gaze is often associated with which items people are allocating attention to. This signal will analyse driver behaviour and attention within the simulator.

This section will begin by discussing the methods to acquire eye tracking data and the tools used. It will then cover the calibration protocol used to fit the eye tracking estimation model to the person being tracked. Following this, the methods for processing eye tracking data are covered. This includes the filters used to clean the data and the interpretation of eye gaze in pixel space to ocular degrees the eye moves around within its centre of rotation. This differentiation will be significant in the following section covering eye movement classification. The eyes are noted as giving attention to regions in their line of sight when the eye is static. This is referred to as a fixation. Fixations are classified by measuring their rotational speed, a threshold of 80 degrees per second. This threshold is measured in ocular space because participants between studies can be seated at varying distances from the eye tracker or viewing screen. A measurement threshold based on pixels may vary in terms of the speed of the eye. For example, a person sitting very close to the screen would have to move their eyes faster to bring their focus from top to bottom than someone sitting further away. Measuring in ocular space allows for consistency between studies in the general research area of human attention while using eye-tracking equipment.

A.1.1 Acquisition

Eye-tracking and the sensors used to observe it has been developing over the last hundred years [50]. For the research carried out here, there is a focus on using video-based sensors combined with pupil/corneal reflection techniques. These eye trackers use cameras and image processing techniques to compute the point of regard (usually the observed point on a monitor) both offline and in real-time. This can be done for on-screen computer desktop scenarios and using head-mounted displays for testing virtual reality and tracking the eyes while interacting with the

real world. Most eye tracking systems use similar cameras and image processing techniques but vary in their calibration methods for the types of surfaces they observe. A review of methods for retrieving point-of-regard measurements for the interested reader may consider the following [50].

Given that most eye-tracking systems are considered concerning egocentric views across the image plane, our novel contribution will be constructing a 3D model based on real-world images (Chapter 4), allowing for exocentric eye-tracking over a 3D map (Appendix B.1). This will be done by projecting the eye gaze from the image plane/viewport into the 3D map and provide 3D points for analysis. Another contribution will be to map these eye gazes/fixations from the 3D map to another image plane corresponding to a viewing camera used to construct the 3D map initially.

Among the most popular desktop-based eye trackers are the Eyelink 1000, a high-precision eye tracker produced by SR Research [130]. This equipment has a sampling rate of up to 2000Hz for binocular eye tracking. It is among the most accurate and precise eye trackers currently available. The accuracy of eye tracking systems is measured in terms of angular degrees about the centre of the eye. In head-mounted and remote setups (head free-to-move), its accuracy is typically $0.25° - 0.50°$, while approximately 60cm away from the screen and tracker. The head-mounted setup can be as low as $0.15°$. This equipment is typically used for highly accurate eye tracking for reading experiments or stimuli containing small movements. Its free-to-move setup involves participants wearing a sticker on their heads for the camera to track against (head localization).

There are many other sources for standard eye tracking experiments involving free head movement (without the use of tracking stickers). For example, Tobii Technology has a multitude of eye-tracking sensors. Their most recent research-grade desktop-eye-tracker samples up to 250hz of binocular tracking. It has an accuracy of $0.4°$ and a precision of $0.5°$ while seated approximately 60cm away from the screen and tracker. A review of how they test for these specifications can be found in [80]. Most human activity studies, such as driving, do not require accuracy better, which $0.4°$ makes the Tobii eye-tracker an attractive option. Typically, an eye-tracker with higher accuracy than this is more suited to a study on micro-saccades, an eye movement not typically studied concerning the area of

driver experiments. The Tobii eye-tracking system comes complete with software to enable the processing of the eye-tracking data. A specialized method has been developed for processing and labelling eye gaze measured using these cameras. This is detailed in the Tobii I-VT fixation filter documentation [117]. This process will be described in more detail later in this chapter, as this fixation filter has been used within the experiments reported in this thesis.

A.1.2 Calibration

Calibration is used to match 3D models of the eye to the actual eye of the participants during an eye-tracking experiment. Since human individuals contain variation in the shape and positioning of their eyes, eye-trackers often develop individual calibration procedures to optimize gaze estimation [122]. As a result, an accurate estimate of gaze points is produced. Calibration is often done before eye-tracking is recorded for either an experiment or a human-computer interface. This process considers shapes, and light refraction and reflection on different parts of the eye, including the cornea and placement of the fovea [183]. Participants are required to look at calibration points on the screen. At the same time, images are recorded by the eye-tracking camera to be analysed during the calibration process. The analysis following the processing of these images is used to develop a model of the participant's eyes.

Figure A.1 shows an example of the calibration pattern used for desktop eye-tracker calibration. During calibration, participants are required to gaze at each calibration dot. This is done for a set period or based on experimenter requests. There can be between 4-9 dots, sometimes 16 for large screens. The more dots used for calibration, the more accurate the calibration tends to be. However, this comes with the expense of participant attention, as participants may become bored with long calibration sessions. The grey dots represent the calibration dot area (where participants should focus their visual attention), and the green lines highlight the difference in where the eye gaze was estimated to be focused based on its current eye model. Ideally, the length of these lines would be minimized across all dots present for the calibration pattern. Taking note of the distance of the participants (often recommended to be 60cm for modern Tobii cameras),

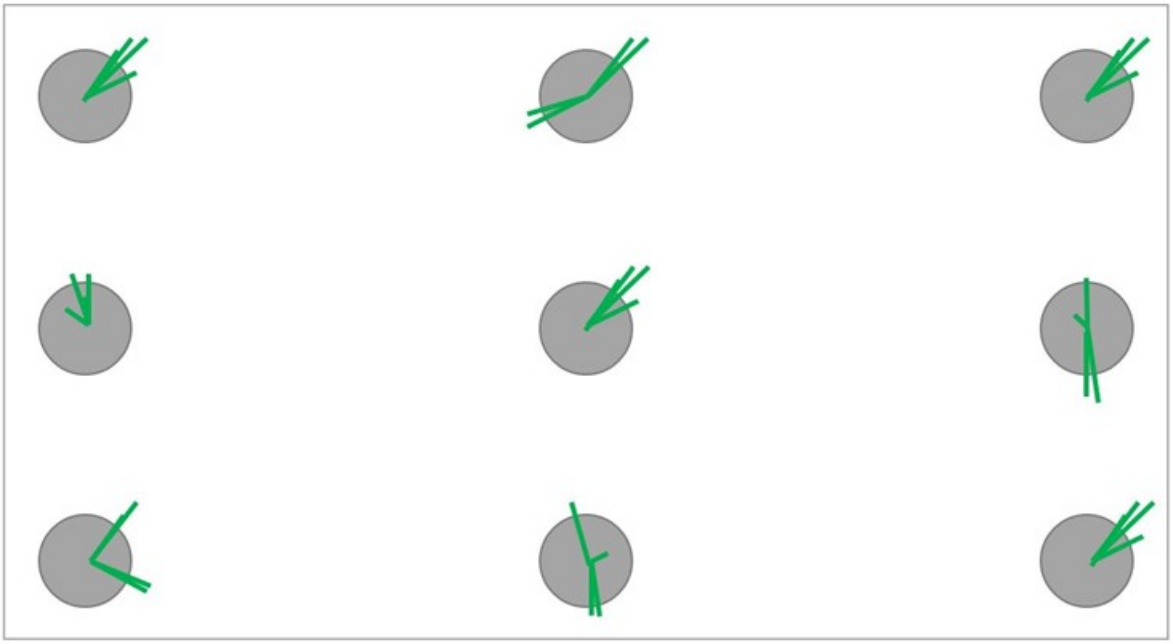


Figure A.1: Calibration pattern for eye-tracking.

the resolution and size of the screen being used, and the distance of these lines, an error rate in terms of optical degrees can be determined. This error rate can be influenced by many factors, including participants not focusing on the points, the eye tracker not being set up appropriately, or excessive head movement [122].

Validation is often used to test the actual error rate of the eye model. The calibration fits an eye model that returns a minimum error rate while conforming to a rational eye model. It is possible that this eye model could overfit the data. For this reason, validation is used to check the actual error rate of the eye model based on unseen data. This involves a similar process to the calibration where a participant is required to focus on calibration dots, except that these dots appear in positions not included in the original calibration pattern. This tests the ability of the eye model to generalize to other parts of the screen. Error is measured in the same way, with ocular degrees, and will inform the experimenters if calibration needs to be repeated.

A.1.3 Processing Gaze Points

Consider the point of regard for an eye gaze position on a planar surface (the screen the user is looking at). This point can be considered a position in the image space domain $\Omega \subset \mathbb{N}^2$, where Ω represents the current frame presented on the screen for the viewer. The gaze point or point of regard will be referred to as position $\mathbf{x} = \{x, y\}$, where x and y are the horizontal and vertical components, respectively. These can be represented as points on the image plane presented to the participant. See the right image in Figure A.2 for an example of the scan path of a participant in a driving simulator. Blue circles are points of regard closer to $\mathbf{x}_0$, green circles are closer to $\mathbf{x}_{n/2}$, and red circles are closer to $\mathbf{x}_n$, where the subscript reflects the temporal ordering of the points of regard.

Figure A.2: Points of regard captured in image frame.

Often eye tracking data is analysed collectively with many points-of-regard to filter out the noise and evaluate what the eyes can observe. This "noise" can be any number of things; it depends on what signal the experimenter is searching for. There includes sensor noise that comes from the camera. Cameras are not perfect and can contain radial distortions that affect the assumptions that may be made during post-processing of the image plane and the point of regard. This can usually be adjusted during calibration, but collecting many measurements and using averaging techniques can also help filter this type of noise. For this reason, calibration is carried out for each participant prior to experimentation. Eye gaze is also rarely considered independently. Other factors to consider are the head motion of the participant or alternate sources of light interfering with corneal reflection tracking. Speed and acceleration-based techniques can be used

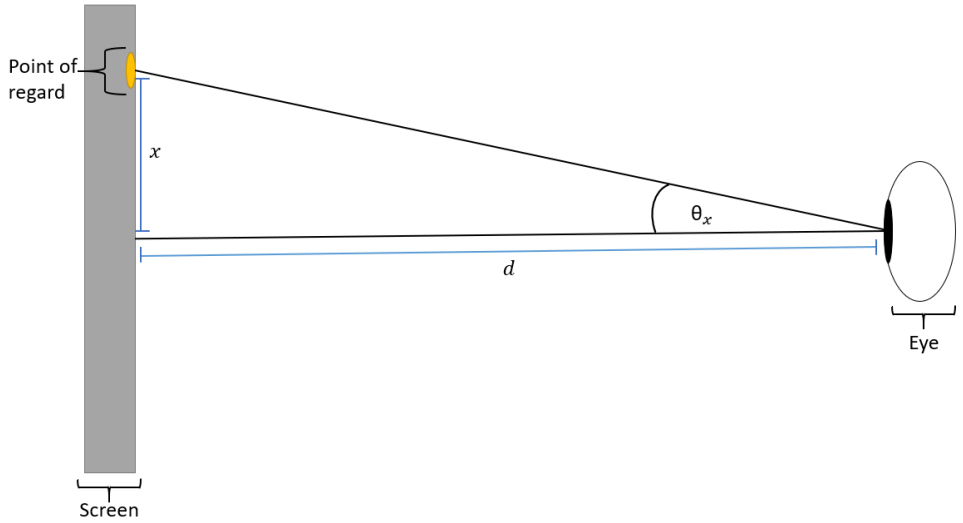


Figure A.3: Visualization of variables used to calculate ocular degrees from pixels while viewing a screen.

to remove outliers. Factors known generally about eye movement can be used to sensibly eliminate possible eye measurements, such as the potential speed an eye can rotate in degrees per second. For example, [50] discusses using an upward expectation of saccade (sudden eye movements) velocity of $600°/sec$ (visual angle, will be discussed below) or saccade duration of $120-300ms$. These thresholds can be chosen empirically as they may depend on the events the participant witnesses.

Given the temporal nature of eye tracking, data points are measured relative to the previous and future positions (future if the processing is not in real-time). Now consider each point of regard contained in a temporally ordered set $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\}$, where n is the number of samples. Each of these points of regard $\{x_i, y_i\}$ can then be converted into visual degrees, which is an angle of the participant's eyes relative to their centre of rotation. This is estimated given the size of the screen (w_s, h_s), the expected position of the participant relative to the screen d, and the resolution of the frame used for tracking (w_r, h_r). Given these variables, the following equations can be used to get the ocular degrees.

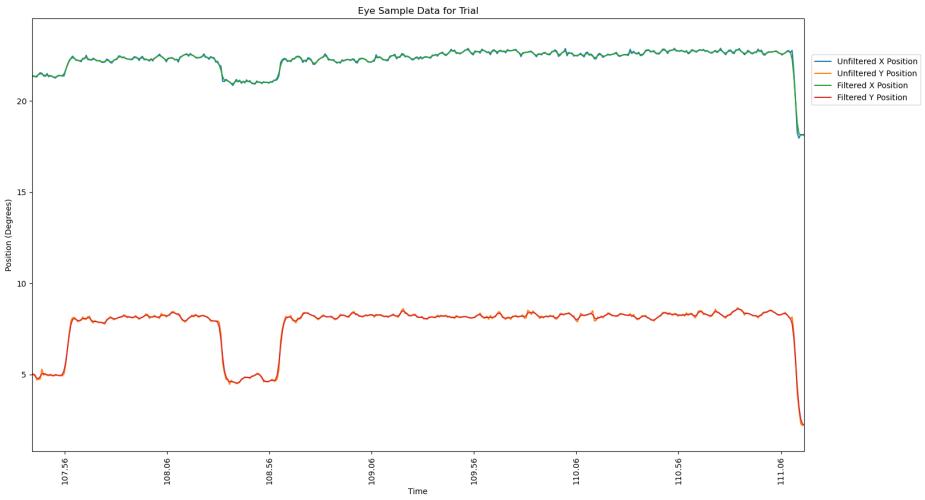


Figure A.4: Ocular angle vs. time for a participant, showing the x and y degree changes.

$$\theta_x = \arctan \frac{x}{d\frac{w_r}{w_s}} \tag{A.1}$$

$$\theta_y = \arctan \frac{y}{d\frac{h_r}{h_s}} \tag{A.2}$$

These equations appear obvious to solve when considered with Figure A.3 and the appearance of a right-angled triangle. A similar setup can be considered for visualizing θ_y. Plotting the ocular degrees in their ordered series after using filtering techniques such as the 5-tap FIR (finite impulse response) filter should produce the following, Figure A.4. This filtering technique is a good option for reducing sensor noise and head movement from the participant.

Now the ordered set of gazes can be considered in the set $\mathbf{\Theta} = \{\theta_\mathbf{1}, \theta_\mathbf{2}, ..., \theta_\mathbf{n}\}$, where each $\theta_i = \{\theta_x, \theta_y\}$ is generated from the above Equations A.1 and A.2 using the set $\mathbf{X}$ as inputs for x and y. To calculate the speed of the eye, the difference between adjacent members of the ordered set $\mathbf{\Theta}$ and its corresponding timestamps $\mathbf{T} = \{T_1, T_2, ..., T_n\}$, and dividing the ocular degree difference by the time-step difference will provide the speed (similar to the derivative of the ocular

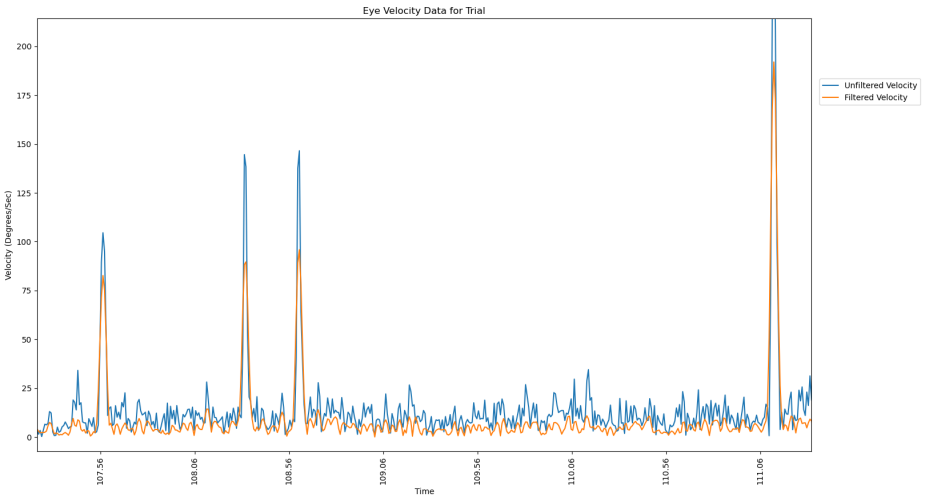


Figure A.5: Ocular degree velocity vs. time for a participant.

degrees to time), see Equation A.4.. Note that the difference between θ_i and θ_{i-1} is calculated after θ_{ix} and θ_{iy} are transformed to a Euclidean distance format, Equation A.3. These velocities can then be represented in a time series as the ocular degrees were represented before; see Figure A.5.

$$let, \theta_i = \sqrt{\theta_x^2 + \theta_y^2} \tag{A.3}$$

$$\dot{\theta}_i = \frac{\theta_i - \theta_{i-1}}{T_i - T_{i-1}}, \forall \dot{\theta}_i \in \dot{\mathbf{\Theta}} \tag{A.4}$$

Observing Figure A.4 and A.5 together, high velocity, or peaks in Figure A.5 , corresponds to more significant eye movements, seen in Figure A.4. With the given speeds held in, a threshold can be set to some limit to label types of eye movement under specific time bands. This will be discussed in the next section. These intermediate representations of the eye gaze are essential for classifying eye movement. Labelling eye movement based on pixel coordinates alone can have a reasonable accuracy; however, comparing across studies can be difficult as not all screens have equal resolution or size for all experiments.

A.1.4 Classification

Saccade and fixation identification is the most prevalent eye movement classifications. They are usually considered when testing participants' overt attention. Saccades are rapid eye movements used to re-position the fovea to a different visual field region. Saccadic eye movement can be controlled and reflexive. The average transition time for a saccade is between 10 to 100ms. During this time, the actor is partially blind [148]. These movements occur so fast that the actor does not perceive the temporary blindness. While collecting raw eye gaze coordinates, it is possible to capture the position of the eyes during saccadic movement. It is unlikely the actors are attending to an object at this position as they would be blind at the time in question. For this reason, eye gaze data is usually labelled before being analysed.

Fixation involves the stabilisation of eye movement to focus on an object. They appear with minor, if not null, velocity. The most straightforward approach to classifying fixations considers each of the sampled points of regard in the ordered set $\mathbf{X}$ and a Euclidean distance between each of these samples. A collection of samples below some threshold may be considered a fixation. This is equivalent to putting a bounding box around each sample $\mathbf{x_i} \in \mathbf{X}$ and checking for neighbouring samples within this bounding box. This technique is helpful for a first approximation of fixations; however, it may only be valid for certain types of stimuli where the features of interest are far enough away from each other.

A slightly more complex and robust way is to use eye speed. Using the speeds calculated in ocular degrees, $\dot{\mathbf{\Theta}}$, it is possible to use the knowledge of the physiology of the eye to inform what classification to provide for each sampled eye gaze. A typical lower bound for saccades would be $80°/sec$. Using a simple classification model, all samples below $80°/sec$ would be a fixation, and all samples above would be saccades. Fixations can also be classified by merging neighbouring fixations. This depends on the amount of time (below 75ms) and the number of degrees (below 0.5°) between them. Small fixations below 60ms are removed based on the time it usually takes participants to register information visually. All these considerations are explained fully in the Tobii I-VT fixation filter documentation [117]. Using these filters, the following should result from labelling

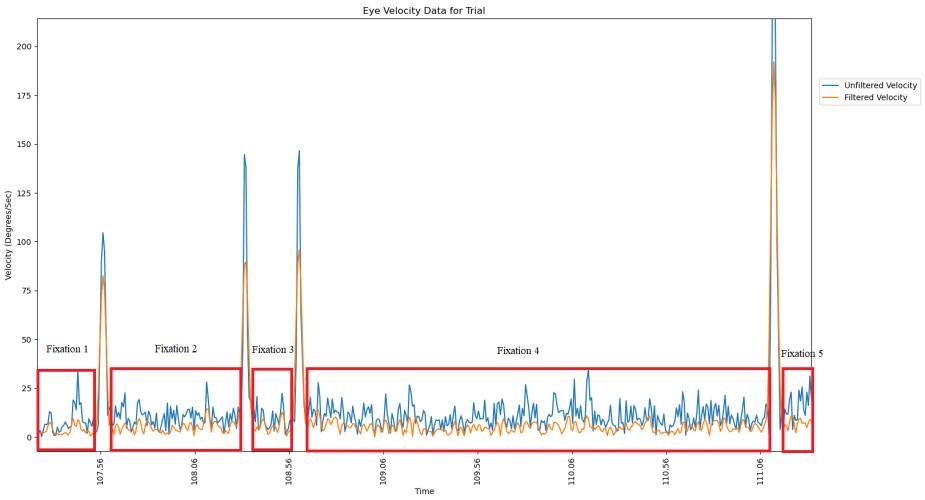


Figure A.6: A participant's Ocular degree velocity vs. time, including fixation labels.

the velocity, see Figure A.6.

Smooth pursuit eye movements are used for tracking moving objects. Depending on the object's speed, the eyes can match the object's velocity with this type of eye movement [50]. Smooth pursuit is a negative feedback loop, which predicts where the object being tracked is going based on the observed speed at which the object travels [93]. The speed the eye should travel is calculated from its difference from the speed of the pursued object, and the eye accelerates until the difference equals zero. This model for eye movement may be ideal when considering a driving context. The visual field in a driving context is often dynamically changing, and objects are either being passed by or passing out the driver. Smooth pursuit eye movements are likely one of the tools drivers use to track these items while maintaining the driver's overt attention.

Summary: Eye tracking can be used to measure human attention. The above methods can acquire, calibrate, process, and classify eye tracking data. The acquisition can be made using corneal/pupil reflections with high-end eye trackers. Processing eye tracking data requires filters to clean it and make it temporally consistent/coherent. The temporal nature of eye tracking is vital as a single eye gaze conveys less meaningful information. This only highlights the direction of

the eye at that moment, not whether the person was cognizant of what they were seeing. The transformation of eye gaze from screen pixel space to ocular degree space is essential to relate eye tracking studies in the field of human attention. This allows for specific speed threshold algorithms to be used to classify eye gaze points with the support of the literature. The classification algorithms discussed in this section will be used in Chapter 3.

A.2 Structure From Motion Pipeline

The following is a summary of the main processes involved in structure from motion. Structure from motion is used throughout this thesis and different design choices have been made based on these processes. For example, the choice to use COLMAP SfM [139] instead of OpenMVG [115] in Chapter 2. Or the choice to use the camera priors from ORBSLAM2 [116] before reconstructing roads from the KITTI dataset [63] in Chapter 6. There are two main steps in SfM: correspondence search and incremental reconstruction. Each of these are described in detail below:

Correspondence Search

- **Feature extraction**: To localize common image features, distinguishable and rich image points are identified, see Figure A.7a. Approaches with small baseline differences between the two views use image edges and corners [45]. However, for large baseline differences that were rotation and scale invariant and robust to illumination changes, features like SIFT (Scale-Invariant Feature Transform) were used [99].

- **Matching**: A scene graph is constructed to expand the problem to multiple images/views; see Figure A.7b. Image features must be matched to construct a scene graph, where each node in the graph would be an image representing some view, and the edges were the shared features both images observed [57, 1].

- **Geometric verification**: a geometric verification will be used to refine the number of matches such that outlier contaminated feature matches are

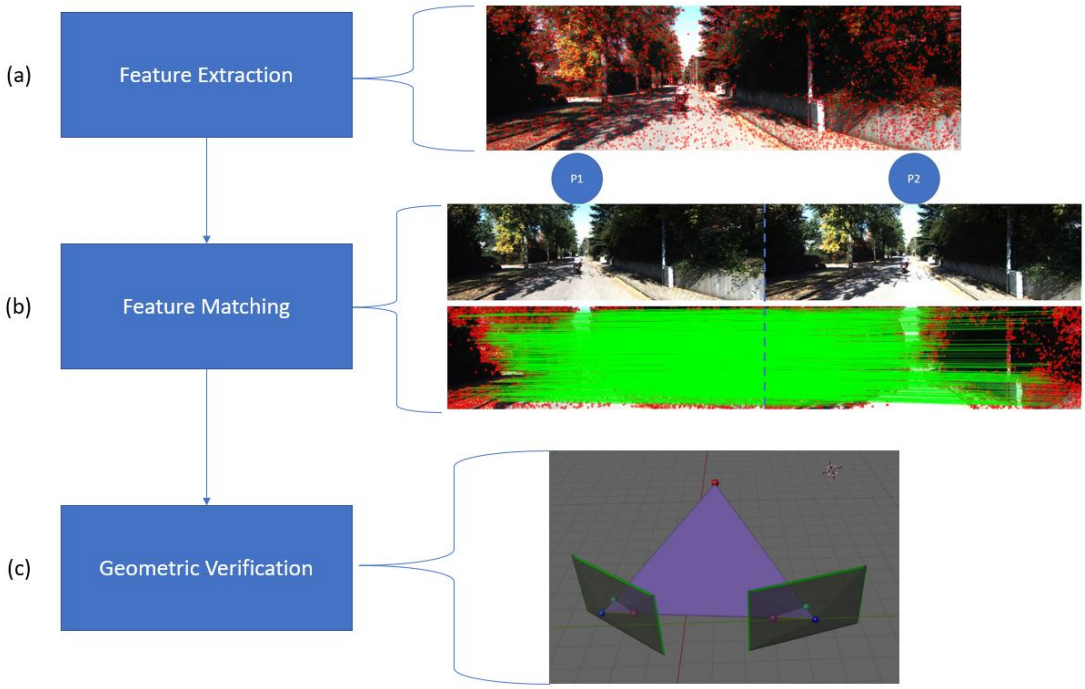


Figure A.7: This is the correspondence search workflow for SfM. (a) Highlights the feature extraction over an image, as seen with the red dots. (b) shows how a scene graph is built using each image as a node, and the edges between nodes are the feature correspondences between images. Again, features are shown as red dots, and the green lines show correspondences. (c) shows a representation of geometric verification. In this instance, a single 3D point can be seen from two views. The rectangles with green borders are the image planes of the two views, and the observed 3D point is a red dot. The observed 3D point on each image plane is represented with a green dot, and the epipoles highlighting the direction of the other camera centre are shown on the image plane as a purple dot.

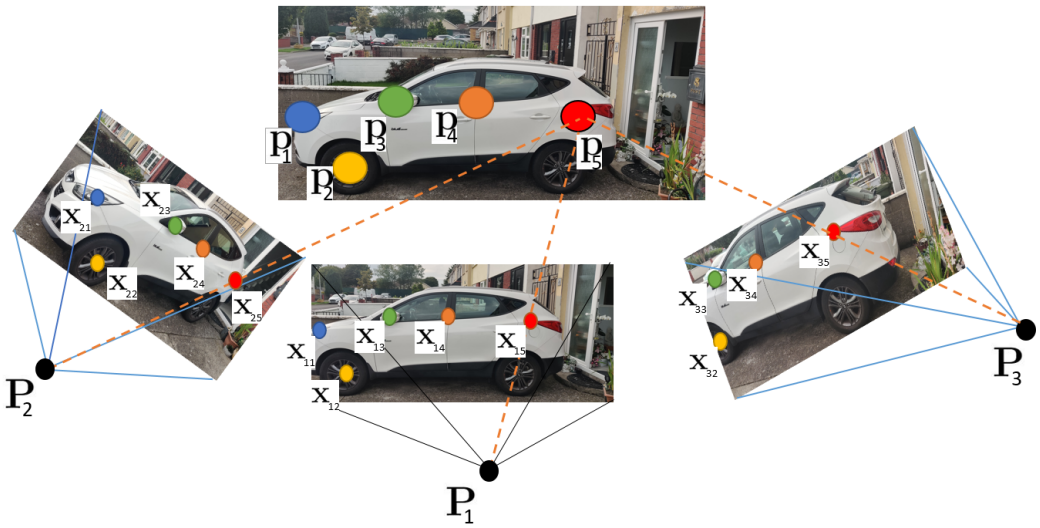


Figure A.8: The 2D image points, $\mathbf{x}_{ij}$, of each image, i, is triangulated to some 3D point $\mathbf{p}_j$. They are projected from a viewing image plane $\mathbf{P}_i$ where the image corresponding shares a subscript i with its respective pose.

omitted. This involves estimating a projective transformation of image features for overlapping image pairs, see Figure A.7c. A homography is used for pure rotational movement or a camera capturing features on a planar scene [70, 102]. An essential matrix $\mathbf{E}$ is estimated for calibrated cameras, providing the projective transformation for non-planar and non-rotational (not just rotation only) movement. For uncalibrated cameras, an estimation of the fundamental matrix is required, $\mathbf{F}$. If a threshold number of image features are found for a valid transformation, those image pair features in the scene graph are geometrically verified. To remove outlier-contaminated features from these images' methods like RANSAC (Random sample consensus) are typically used [54]. The appropriate geometric relation (homography/essential matrix) to use can be calculated using GRIC (Geometric Robust Information Criterion) [169]. GRIC uses performance metrics from the projective transformations to grade which transformation would be more suitable for the image pairs features.

Incremental Reconstruction

- **Initialization**: SfM carefully selects an image pair to start the reconstruc-

tion [5]. Selecting a good image pair is critical to future steps in incremental reconstruction. An initial good image pair includes a dense set of features in the images with a lot of overlap between themselves and other images in the scene graph. This overlap can be measured as the number of edges between pairs of nodes in the scene graph.

- **Image Registration**: additional views can be added by solving the Perspective-n-Point (PnP) problem, provided a 3D reconstruction from the initial two views exists [54]. This is done by matching the projected 3D features already in the 3D structure to 2D features in the new view. Solving this problem produces a pose for the new view. The set of poses contained in the newly registered scene graph is extended by this new pose when localized. RANSAC, and other solvers, have been used to remove outlier-contaminated correspondences between the 2D and 3D features.

- Triangulation: To register a new image in the existing set of poses, image features must have matched the current set of the triangulated points. Once this image has been registered, it can expand the number of scene points through triangulation, assuming this image also has a new perspective not already registered [71]. The expanded geometry enables the registration of more images. An example of triangulation can be seen in Figure A.8, where each camera pose, $\mathbf{P}_i$, projects the observed 2D features, $\mathbf{x}_{ij}$, into 3D space with a ray, and the intersection forms the 3D points $\mathbf{p}_j$. The intersection of the 3D rays can be seen with the orange dotted line going through image features $\mathbf{x}_{i5}$, where i is the individual pose, and they intersect at a 3D point $\mathbf{p}_5$.

- Bundle Adjustment: The poses and 3D points produced from image registration and triangulation come with uncertainty. Uncertainties related to either the 3D triangulated points or the estimated pose impose uncertainty on the other. The increased number of triangulated points improves camera pose accuracy reducing the influence of individual 3D points [171]. Bundle adjustment acts as the joint non-linear refinement of the camera and point parameters currently associated with the registered images and geometry.

This refinement prevents SfM from drifting away from the optimal solution. This minimizes the reprojection error of the 2D-3D features across all images. See Figure A.9 for a visualization of the reprojection error. This is formalized in Equation A.5, where the minimization of the loss function $\mathcal{L}$ is calculated. The loss, in this case, is the reprojection error between the triangulated features $\mathbf{p}$ and the image features $\mathbf{x}$. The parameters minimized are the 3D feature points and the poses $\mathbf{P}$. The 3D points are localized on the image plane using the pose and the function $\pi(\cdot, \cdot)$, which calculated the reverse projection to find the point on the image plane. Levenberg-Marquardt is the method of choice for solving this optimization problem [171].

$$\mathcal{L} = \sum_{j=0}^{m} \sum_{i=0}^{n} \rho_j(||\mathbf{x}_j - \pi(\mathbf{P}_i, \mathbf{p}_j)||^2) \tag{A.5}$$

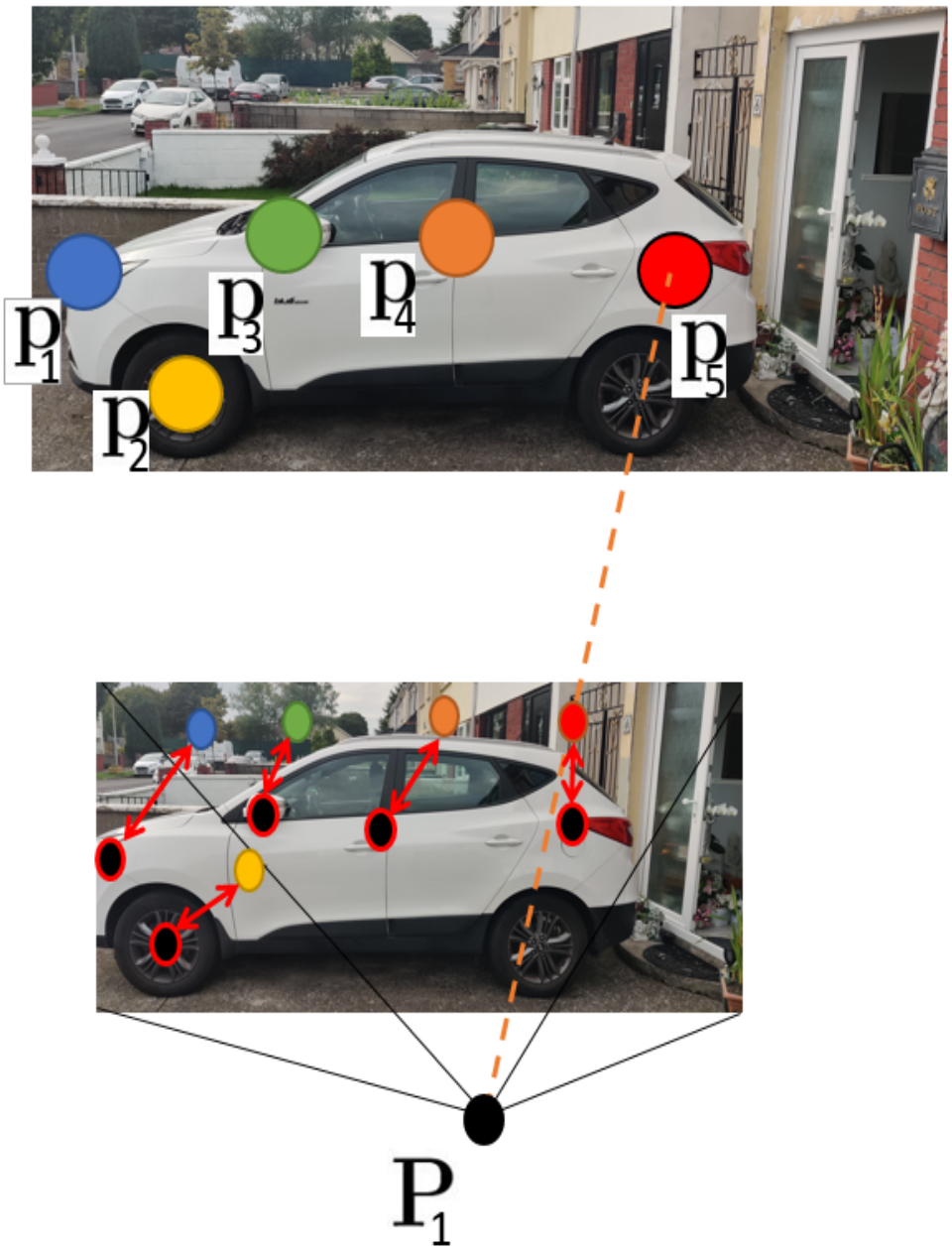


Figure A.9: An example of reprojection error of the corresponding 3D image feature points, $[\mathbf{p_1}, \ldots, \mathbf{p_5}]$, back project to the image at different locations from where they are localized (black nodes with red border). The distance between the features' back projection and the features' location on the image place was used to calculate the reprojection error across all images.

Appendix B

Future Works Details

B.1 Eye Tracking Over 3D Models

During development of the driving simulator eye-gaze to 3D model measurement was impleneted and tested using ray picking algorithms. The projective transformations described in this thesis could take a 2D coordinate over the image plane (viewport) and map them to a vector pointing away from the virtual camera in the environment. This vector pointing away from the camera is referred to as a ray and is represented in Equation B.1[1]. Figure B.1 shows how the ray relates to the image point $\mathbf{u}$.

$$r = \mathbf{P}^{-1}\mathbf{K}^{-1}\mathbf{u} \tag{B.1}$$

To find 3D points overlapping with the ray r a sampling was taken iteratively along the ray until an intersection was found, see Figure B.2. In this example, a ray r is defined based on the camera's eye-gaze projection. The algorithm steps along the ray until it reaches a length w where there exists an intersection with some 3D point p, see Equation B.2, where r_0 is the origin of the ray r. This method can be costly depending on the step size, λ, along the ray. The method's

[1] $\mathbf{P}^{-1}\mathbf{K}^{-1}$ can be computed fast using a few useful facts. $\mathbf{P}$ is composed of a rotation matrix $\mathbf{R}$, and translation matrix $\mathbf{T}$. The inverse of a rotation matrix is its transpose, and a translation matrix is $-\mathbf{R}^{\intercal}\mathbf{T}$. Although computing $\mathbf{K}^{-1}$ might be an expensive computation; it can be computed once at the start of the program, given that the camera intrinsics are unlikely to require a change.

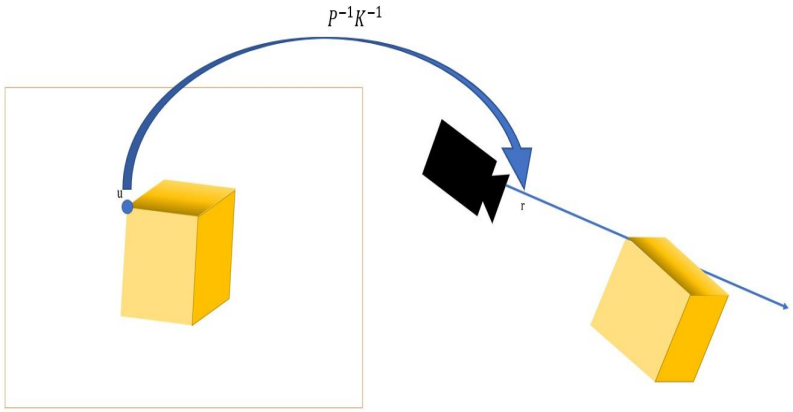


Figure B.1: Computing a ray from the 2D image plane to 3D space.

complexity grows based on the number of polygons/vertices. Each of these will have to be tested against each step along the ray. A more straightforward option might be to create a line equation for the ray and query the point(s) p sit on that line. It should also only consider the point if it is the minimum distance compared to all other points on the line and is in front of the camera.

$$p = r_0 + wr \tag{B.2}$$

A method proposed in Duchowski's book was to consider planes on which many vertices sit (assuming shared faces) and find where the ray intersects with the plane [50]. This could drastically reduce the number of vertices considered, as many points could exist on this plane. The next step would be to calculate if the point at which they intersected was within a rendered polygon. This method was chosen to work in combination with the first version projective texture mapping case described in Chapter 4. This used a simple open-ended box model. In this case, the set of faces being rendered was produced using triangles, so for ease of calculation, the plane was found relative to the positions of the vertices, x_{ij}, of each triangle (face), $\mathcal{T}_i$, where i was the face, the vertex belonged to, and j corresponded to the individual vertex. The planar norm, ν_i, was calculated using

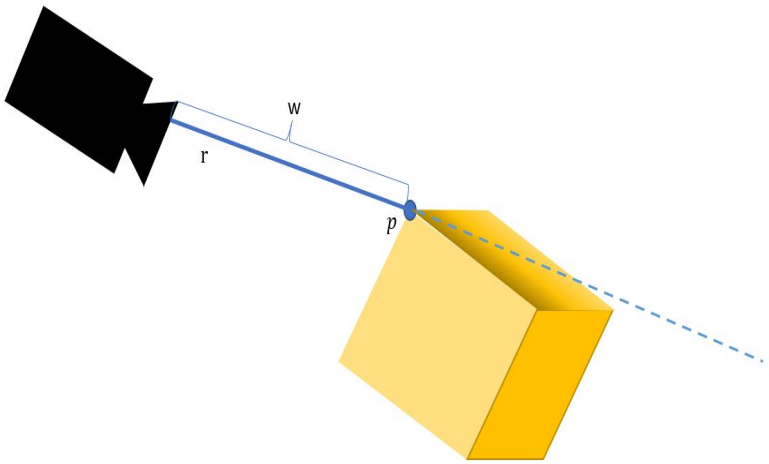


Figure B.2: Ray tracing: iterative method

Equation B.3[1], where ℓ_{ijk} is the line between vertex x_{ij} and x_{ik} on the face $\mathcal{T}_i$. The inner product of the planar norm, and the ray, r, would then show if they were parallel and non-intersecting if the result is not equal to 0. Following this, the planar equation was used to find the distance w, given the ray r substituted for some point in the plane Equation B.4, see Figure B.3 for a visual representation.

$$\begin{aligned} \ell_{i,2,1} &= x_{i,2} - x_{i,1} \\ \ell_{i,3,1} &= x_{i,3} - x_{i,1} \\ \nu_i &= [\ell_{i,2,1}]_\times \cdot \ell_{i,3,1} \end{aligned} \tag{B.3}$$

$$\begin{aligned} \nu_i \cdot p &= d \\ \nu_i \cdot [r_0 + wr] &= d \\ w &= \frac{d - \nu_i \cdot r_0}{\nu_i \cdot r} \end{aligned} \tag{B.4}$$

[1] $[u]_\times$ is being used here to represent the skew-symmetric matrix. It is being used instead of the cross product $(u \times v)$ as this can be slower. Other representations of the skew-symmetric include $u_\times$ and $\widehat{u}$ in other texts.

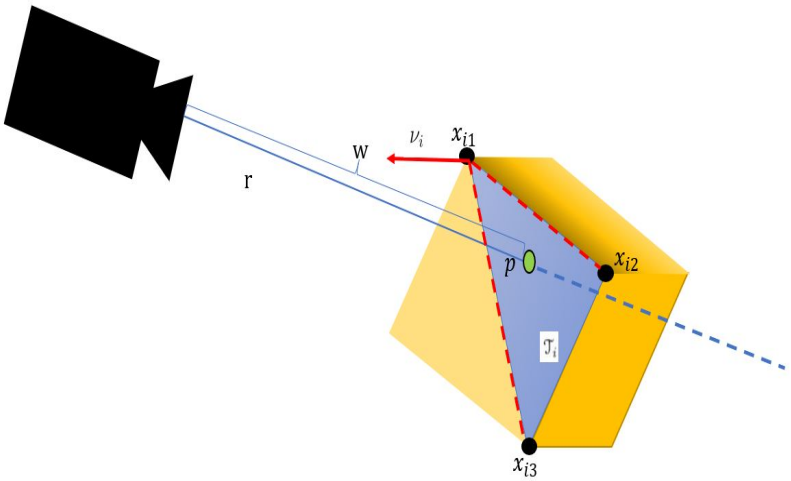


Figure B.3: Point to triangle ray tracing

Given the eye gaze for the point p on the model, the point was projected back to the image that textured the model. A mapping between the eye gaze and the source images could be displayed, despite the transformation differences. Following the calculation of the coordinate p, a texture coordinate could be retrieved using the transformations used to project the texture initially. This can be seen by treating the point p, projected from the view camera as the p, in the texture coordinate calculation in Equation 2.3.

This approach worked well, given that the hallway was composed of 10 triangles (2 triangles per face of an open-ended cube). Following further testing, promising results were found using a video sequence of the "Port Tunnel" projected onto the cube. Eye-tracked points could be retrieved from 3D and back to a single 2D coordinate on the image from the video. An issue with this approach is that faces/triangles will not scale well as models become larger, as seen with the large meshes created in Chapter 4. For extensive scenes, objects should be loaded from memory in smaller sections of the point cloud as required by the renderer and the tracer. This should speed up the rendering process and the retrieval of 3D eye gaze coordinates. Initially, this can be addressed by splitting the mesh into fragments to be loaded into the rendering pipeline based on a factor of Euclidean distance.

It is inefficient to do a ray calculation between the ray and all the points in the vertex buffer. A standard solution to this problem is to group many points and approximate their geometry using a shape that allows for fast calculation of intersections (like the plane intersection tests discussed earlier). The shapes used are typically bounding boxes, spheres, and cylinders. In most graphics engines, they are referred to as "Colliders". These are normally manually annotated for many 3D objects in game engines such as Blender or Unity. However, automatic approaches have been attempted in the past through clustering and random sampling-based (RANSAC) methods [153, 82]. Convolutional deep belief networks have been used to generate voxel representations of 3D data from depth maps, which also encode the distribution of points to aid in identifying hierarchies of sets of points among the voxels. These hierarchies can be used to identify larger regions where points/voxels within that set cannot be intersected if they are not intersected. Others have used CNN's to generate multiple cuboids to approximate 3D point clouds [172] and from depth images [203]. It would be recommended that future implementations of this system take similar approaches to these. However, this section has presented how to project eye-gaze and relate the projections to texture using projective mapping over simple models.

B.2 Ackermann Steering Model -Formalization

While developing the arc-ball motion model, it was proposed that another motion model be applied to give realistic motion to the movement of the rendering camera, the Ackermann model [149]. This Ackermann model was not implemented for the final driving simulation as it was a less critical factor for a demonstration of the final system. However, this section formalises the equations to indicate how it might be used. This has been developed to link the rendering camera transformation matrix and vehicle controls to the current state of the vehicle's wheels. The intention was that the presentation of the 3D reconstructed models would have the *feel* as though the person is driving a car through the scene rather than using some 3D viewer software.

The standard way of representing the transformation of the robotic platform about the global coordinate system will be held in Equation B.5. The trans-

formation between the vehicle's platform's global and local reference frames is considered a planar transformation. It is considered using three parameters, positions x, y, and angular difference between the reference frames θ. The subscript I denotes the global reference frame. This can be seen in Figure B.4. For simplicity, this can be analysed in a top-down view of ξ_I moving along the ground plane. Later ξ_I can be substituted for P_t, the camera's pose in 3D space as described in Chapter 4.

$$\xi_I = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} \tag{B.5}$$

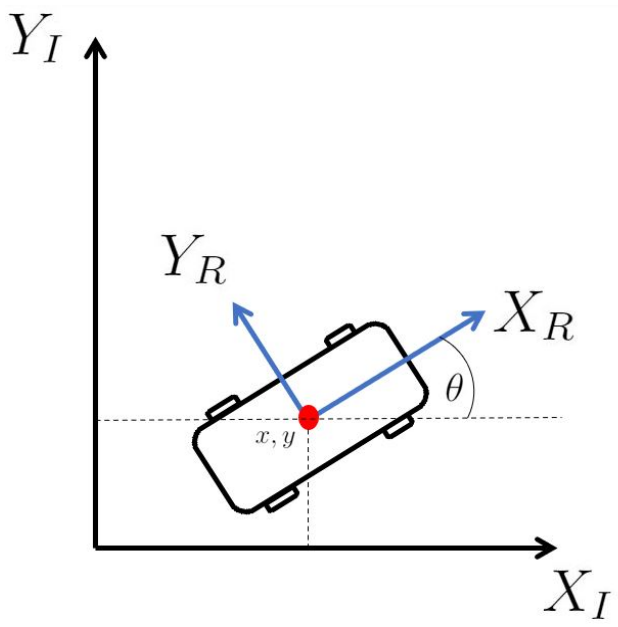


Figure B.4: Top down view of the vehicle platform and local reference (X_R, Y_R) relative to the global reference frame (X_I, Y_I).

On a typical rear wheeled drive car there are two steerable wheels at the front and two fixed motorised wheels at the rear. Each wheel will be described in its local coordinate system relative to some fixed point on the motion platform. This includes the wheels distance from the fixed-point l, this will be called the chassis for future reference. The angular difference between the centre of the wheel and

the fixed point on the x-axis is α. The angular difference between the chassis and the forward wheel plane is β. See Figure B.5 for an example of a single fixed wheel relative to the centre of the motion platform, where the center is also the origin of the axis. The radius of the wheel is r, which can spin over time and the rotational position of the wheel is a function of time $\phi(t)$. The rolling constraint of a fixed wheel can be summarised in Equation B.6. Where $\dot{\xi}_R$ is the instantaneous change in the motion of the vehicle platform within it's own local reference. This equation can be used to calculate the possible direction the vehicle can travel in with respect to a set of control inputs (the rate the wheels are spinning and the orientation of the wheels at that time). A steerable wheel has the same setup with exception to β, which can change over time in relation to the drivers input, and can be replaced with $\beta(t)$.

$$\begin{bmatrix} sin(\alpha+\beta) & cos(\alpha+\beta) & (-l)cos(\beta) \end{bmatrix} \dot{\xi}_R - r\varphi = 0 \tag{B.6}$$

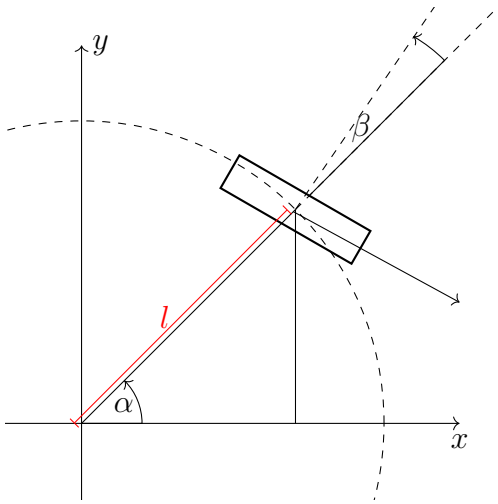


Figure B.5: Example of a standard wheel relative to the vehicle platform

To calculate the appropriate β for the wheel to be facing forward the following must be considered. When α is zero the assumed forward vector of the wheel is $\begin{bmatrix} 0 & -1 \end{bmatrix}^\intercal$, call this $\hat{\mathbf{d}}_1$[1]. Following a rotation of α on a 2D plane the new forward vector, $\hat{\mathbf{d}}_\mathbf{2}$, of the wheel can be calculated using Equation B.8. To calculate the

[1]The caret symbol is used in this context to mean the vectors are unit length.

angle β required for the wheel to be facing forward consider the difference in angle between the wheels forward motion line, $\hat{\mathbf{d}}_2$, and the line parallel to the x-axis (call it $\hat{\mathbf{d}}_3$). Note it is possible to consider the lines perpendicular to the wheel but this is essentially the same calculation. It is desired for $\hat{\mathbf{d}}_\mathbf{3}$ to point in the direction $\begin{bmatrix}1 & 0\end{bmatrix}^\intercal$. To find β the angular difference between $\hat{\mathbf{d}}_2$ and $\hat{\mathbf{d}_\mathbf{3}}$ is required (unit vectors are used for ease of calculation), this can be calculated using the dot product see Equation B.9. These forward vectors can be visualized in Figure B.6. This formalization can be extended to all wheels on the vehicle platform, see Figure B.7. In this example each wheel has it's rotation about the origin α_i, and a rotation about its own center β_i, where i denotes the wheel the angle belongs to[1]. The forward vectors described in Figure B.6 are extended to all wheels, $\hat{\mathbf{d}}_{ij}$, where the i associates the forward vector to a particular wheel and j is associated with the order of rotations in which the vectors have undergone.

$$R(\alpha)\hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_2 \tag{B.7}$$

$$\begin{bmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{bmatrix} \hat{\mathbf{d}}_1 = \hat{\mathbf{d}}_2 \tag{B.8}$$

$$\hat{\mathbf{d}}_2.\hat{\mathbf{d}}_3 = |\hat{\mathbf{d}}_2||\hat{\mathbf{d}}_3|cos(\beta) \tag{B.9}$$

The rolling constraints for the vehicle's fixed wheels can be contained within J_{1f} and the constraints for the steerable wheels are $J_{1s}(\beta_s)$, see Equation B.10. J_{1f} is a constant matrix as all the wheels are fixed . However, $J_{1s}(\beta_s)$ varies as it is a function of the steering angle of the wheels at that time. These rolling constraints can then be considered with the spin of the wheels themselves and the position of the vehicle platform globally, similar to Equation B.6, but expanded to include the rolling constraints of all wheels, see Equation B.11. J_2 is a diagonal matrix containing the radii of each wheel represented in $J_1(\beta_s)$ in the same row order, and $\dot{\varphi}$ is the instantaneous rolling motion of these respective wheels at time t. Given the current spin of the wheels, $\dot{\varphi}$ and the steering angle of the front wheels β_s the instantaneous motion of the vehicle $\dot{\xi}_R$ can be calculated at

[1] In some cases the smallest angle to rotate the wheel involved a positive or negative rotation about the axis.

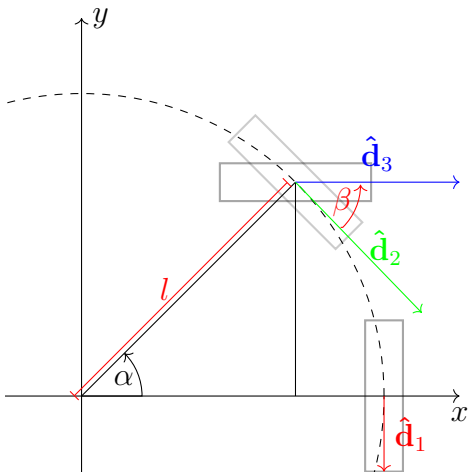


Figure B.6: Rotations of the wheel to ensure the wheels forward vector is forward relative to the vehicle platforms local coordinates.

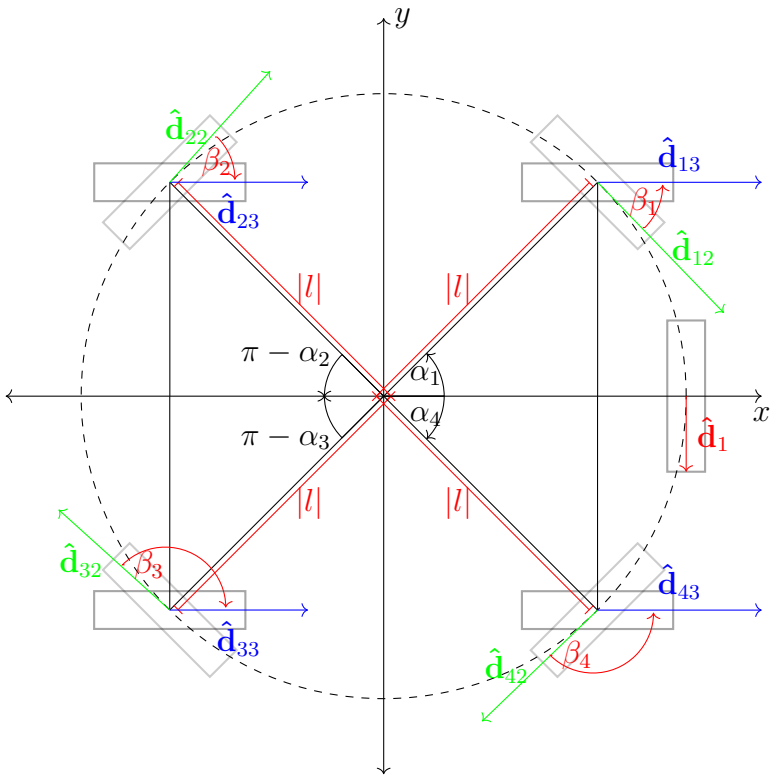


Figure B.7: All wheels rotated and aligned to the vehicle platform such that their forward vectors are parallel to the x-axis

any time given all the other wheel parameters are constant.

$$J_1(\beta_s) = \begin{bmatrix} J_{1f} \\ J_{1s}(\beta_s) \end{bmatrix} = \begin{bmatrix} sin(\alpha_2 + \beta_2) & -cos(\alpha_2 + \beta_2) & -lcos(\beta_2) \\ sin(\alpha_3 + \beta_3) & -cos(\alpha_3 + \beta_3) & -lcos(\beta_3) \\ sin(\alpha_1 + \beta_1(t)) & -cos(\alpha_1 + \beta_1(t)) & -lcos(\beta_1(t)) \\ sin(\alpha_4 + \beta_4(t)) & -cos(\alpha_4 + \beta_4(t)) & -lcos(\beta_4(t)) \end{bmatrix} \tag{B.10}$$

$$J_1(\beta_s)\dot{\xi}_R - J_2\dot{\varphi} = 0 \tag{B.11}$$

This calculation was to limit the vehicle's mobility in the driving simulation at different points in time to produce higher fidelity with real driving. An arc ball camera setup would allow the user to move the vehicle in any translation or rotation at any time. An extension of the rolling constraints was the description of the sliding constraints of the vehicle, $C_1(\beta_s)$. This describes the motion of the vehicle platform made unavailable with a specific orientation of the wheel angles α and β. Using these principles it should be possible to implement vehicle motion with the Ackermann model.

Appendix C

Elastic Fusion and PTM

C.1 Elastic Fusion and PTM

During the PhD there were commissioning experiments for the implementation of projective texture mapping (PTM). Before arriving at COLMAP as the final automated reconstruction tool, Elastic Fusion was used. What follows are the details of how that system was used, the dataset it was tested on, and how PTM was applied to enhance results.

Elastic fusion was used to automate 3D reconstruction given sensor measurements available from the real-world. A 3D model of an indoor scene was reconstructed to commission the Elastic fusion software [184]. While Elastic Fusion was not used in the final prototype, it was helpful, as it provided an accurate point cloud in real-time. It also provided a set of corresponding poses for each image taken of the model. The scene reconstruction was quick due to the availability of depth information in the input data. This provided an effective means of building knowledge about the alignment of textures and 3D data structures. The algorithm's input included camera intrinsic parameters and a set of RGB-D frames/images, see Figure C.1. In this case, the parameters were included as part of the open-source dataset. The outputs included a 3D model and a set of odometry measurements corresponding to the position and pose of the camera capturing the RGB-D images.

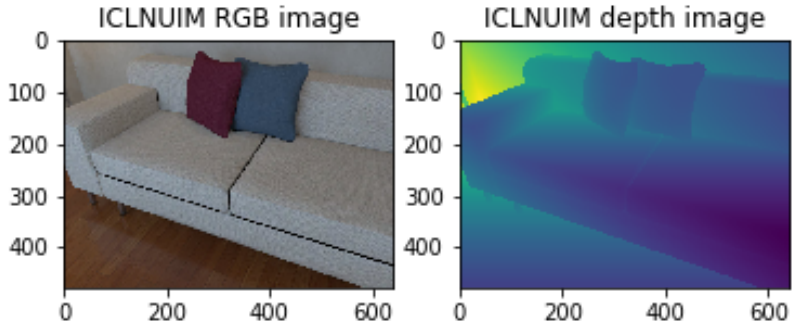


Figure C.1: RGBD image pair from the ICLNUIM dataset.

C.1.1 Formalization

Elastic fusion was used to create a map, $\mathcal{M}$. This map was composed of a set of vertices where each vertex has a position $\mathbf{p} \in \mathbb{R}^3$, and colour $\mathbf{c} \in \mathbb{N}^3$, The image space domain is defined as $\Omega \subset \mathbb{N}^2$, where an RGB-D image was used. Depth maps $\mathcal{D}$ of depth pixels $d : \Omega \to \mathbb{R}$ and the colour image C of colour pixels $\mathbf{c} : \Omega \to \mathbb{N}^3$. A set of transformations were computed to represent the poses of each of these images relative to the mapped vertices but was rewritten with a subscript t for each individual pose at a given time, see Equation 4.7. This used the same interpretation of 3D points described in the previous section while describing the rendering camera $\mathbf{C}$. An example of each of these poses can be seen in Figure C.2 where each camera pose is represented by a black frustum and the 3D model behind them shows the 3D mapped vertices. The camera intrinsic parameters are the same for every image and follow the same convention as the cameras described in Equation 2.1.

C.1.2 The Dataset: ICL-NUIM

Synthetic data from ICL-NUIM was used for RGBD measurements of a 3D model [68]. This dataset was chosen as Elastic fusion has been proven to perform accurately on this dataset. The dataset contained multiple materials, including diffuse, specular, and reflective lighting. These features can be lost during 3D reconstruction and will serve as an excellent example of the strength of PTM.

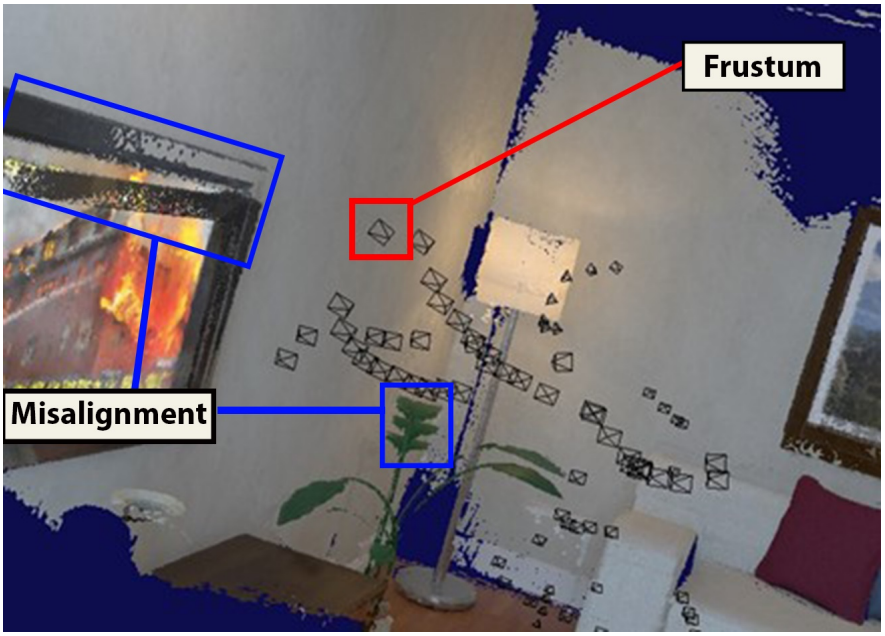


Figure C.2: Final 3D reconstruction and camera poses (frustums).

The tested sequence included the Living Room 'lr kt0' trajectory. Frame-to-frame tracking was enabled for Elastic fusion as it produced the lowest root mean squared error for predicted poses compared with the ground truth poses. The final map, containing a surfel representation, was generated along with predicted odometry. The surfel map was then processed by a Poisson Mesh reconstruction process using MeshLab, reducing the point count from $622,367$ to $253,088$ points and providing a model with a continuous surface rather than a point-based one, see Figure C.3 [83, 30]. Figure C.3 shows how the meshing process carried out hole filling and point decimation. The dense 3D model will provide a helpful surface to project onto, given its completed structure. The texture associated with occluding foreground objects such as the lampshade can appear onto other surfaces within the scene when viewed off-axis with the projector, as detailed in Everitt's implementation [53]. Solutions to this are developed in Section 4.2.4.

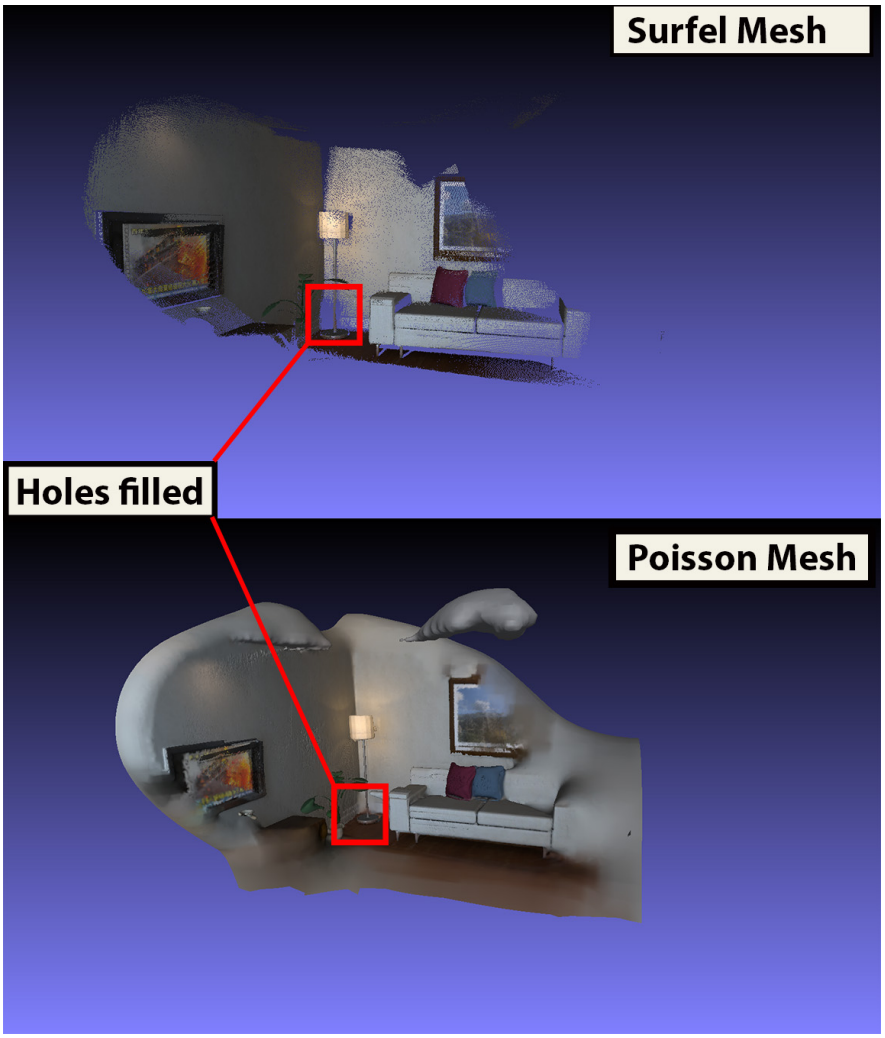


Figure C.3: ICLNUIM reconstruction surfel (top) and Poisson mesh (bottom) representation.

C.2 Image-Model Alignment

Projective texture mapping qualitatively appears photo-realistic and convincing to the human eye. However, it is sensitive to noise due to projection error. The error is more noticeable in this case because the projection of colour is not a continuous function over the 3D model. The estimation of colour on a 3D surface is usually estimated using a moving average over the vertices [39]. Examples of projection errors' effect on projective texture mapping can be seen in Figure C.4. When the projector was not aligned to the geometry found in the model, it resulted in a projection of the texture onto the wrong surfaces (geometry and texture do not align). This section will cover the approach used to rectify the projection errors. The image-model alignment was a solution to a specific problem related to the projection error of images following the use of Elastic Fusion on a 3D model. This was demonstrated using measurements from the ICL-NUIM living room benchmark.

C.2.1 Testing

The following approach was used to correct the projection error between the image and the 3D model. The model, $\mathcal{M}$, was treated as ground truth, and the transformation for each camera pose, $\mathbf{P}_t$ (the estimated camera pose from Elastic Fusion), was optimized relative to the model. The depth map corresponding to $\mathbf{P}_t$ of the model $\mathcal{M}$ was calculated, $\mathcal{P}_t$. This was done by rendering the 3D model in front of the camera but rendering depth values instead of colours. Each of these depth maps was compared against the sensor measurement depth map associated with the equivalent poses $\mathcal{D}_t$ (depth map from RGBD camera). The two depth maps were reprojected into 3D space as though their poses were set to $\mathbf{P}_t$. Poses were aligned using the iterative closest point (ICP) algorithm to solve for a transformation with the minimum error between the model and the predicted pose, $\mathbf{P}_t + \Delta\dot{\mathbf{P}}_t$. The most effective method was the point-to-plane implementation of ICP for the given geometry in the living room dataset [29]. The two depth maps were compared in 3D space. Following ICP, two depth maps $\mathcal{D}_t$, and $\mathcal{P}_t$, were compared using projection error and fitness, see Figure C.5. ICP had a higher root mean squared error than the initial poses, Figure C.5.

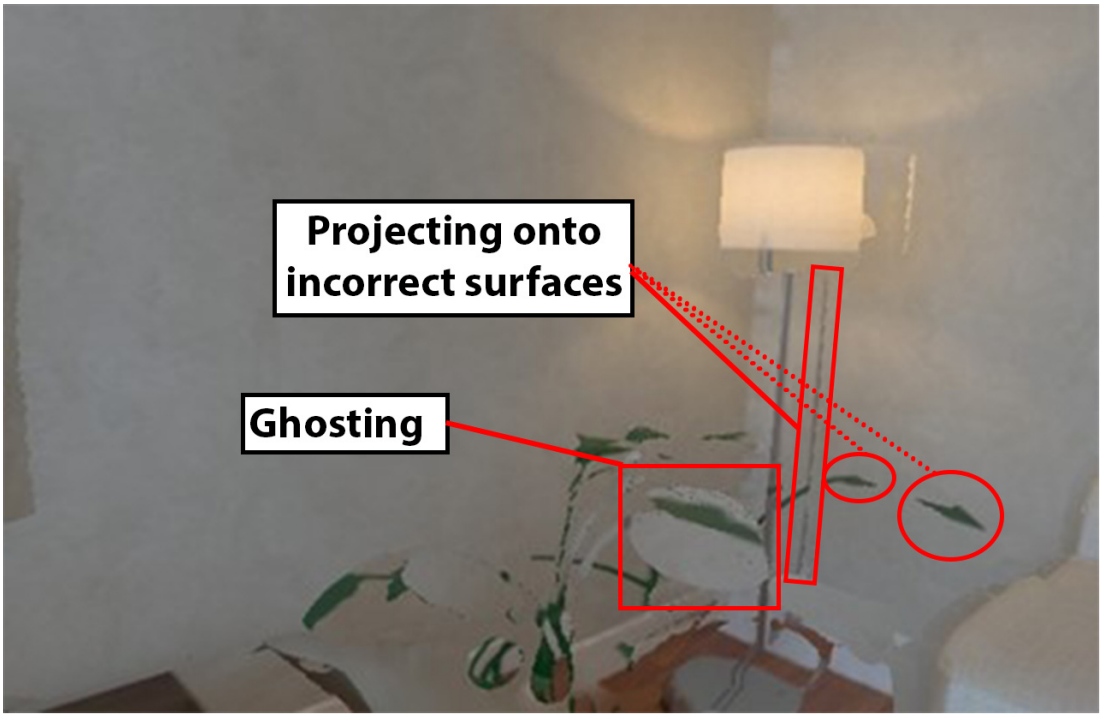
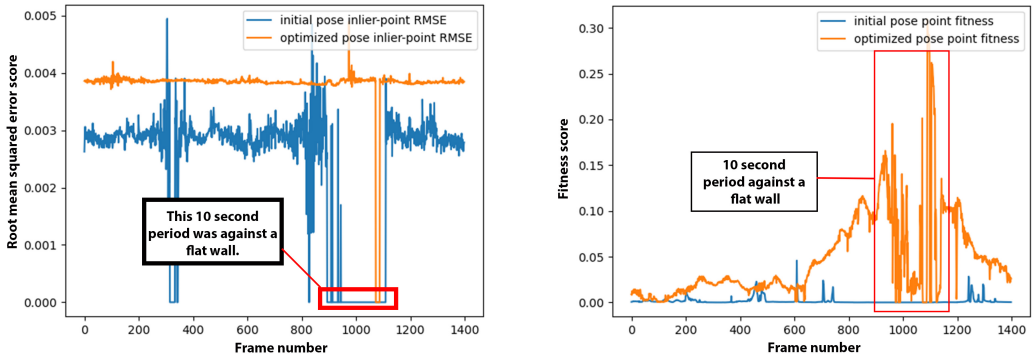


Figure C.4: Reprojection error and its effect on projective texture mapping.

The depth maps after ICP were matched to more points than the initial poses, as shown in the fitness measure, see Figure C.5b. An example of the projections side by side can be seen in Figure C.6. This included the depth map of the model at the estimated pose, $\mathcal{P}_t$, in yellow, and the depth sensor measurement for that pose, $\mathcal{D}_t$, in blue. The left image was the initial disparity between the point clouds. The right image was the projections after the poses were optimized with ICP. Images were created using Open3D [201].



(a) Point error measure.

(b) Point fitness measure.

Figure C.5: Point based optimization performance.

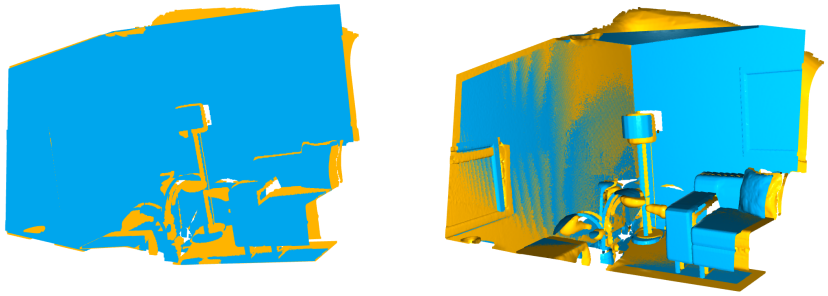
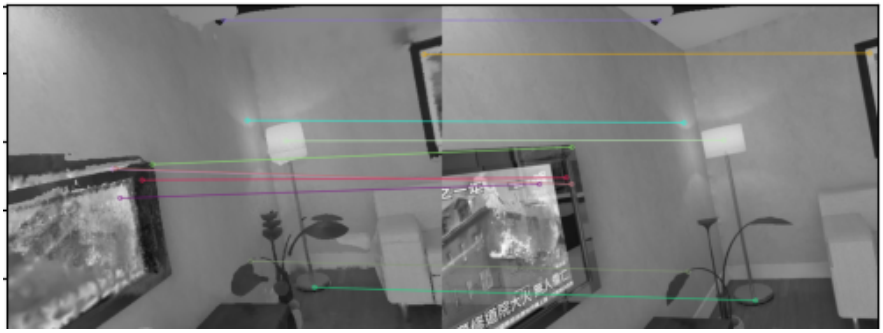


Figure C.6: Initial model/depth map (left) disparity compared to optimized (right). Orange was the 3D mesh rendered from the perspective of the Elastic Fusion pose $\mathbf{P}_t$ and blue was the depth map before (left) and after (right) ICP alignment.

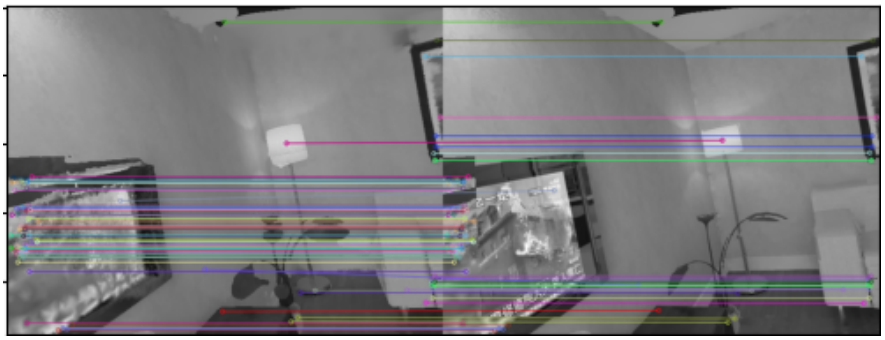
To illustrate the performance of ICP the misalignment was measured by computing SIFT features and matching them between the vertex colours image and the PTM while using the poses before and after ICP, see Figures C.7a and C.7b respectively [99]. The disparity between the features was measured as Euclidean distance on the image planes of each image and was averaged. This measure was reported for the full Living Room 'lr kt0' scene from the ICL-NUIM dataset, see Figure C.8a. In Figure C.7a the average error was a distance of 20 ± 4.33 pixels (standard deviation: 14) with a fitness measure (number of features matched to projection versus the total number of features in vertex colours image) of 0.018. For Figure C.7b the average error was 0.68 ± 0.33 pixels (standard deviation: 3) with a fitness of 0.15. The fitness for all other projections in the trajectory was also included, see Figure C.8b. In most cases, within this dataset, the reprojection error is lower after the optimization while also matching more features than the initial projection provided by Elastic Fusion.

C.2.2 Limitations and Benefits of Elastic fusion and RGBD Images

Elastic Fusion demonstrated the feasibility of using multiple images to reconstruct a scene. The technique works well with images providing different views of the same scene. The camera required a depth sensor. Active depth cameras do not typically work in daylight due to the over-saturation of the sensor from natural sunlight. For this reason, this approach cannot be directly carried over to actual roads. Elastic Fusion is a real-time system and has made sacrifices to keep it running at this rate. A structure-from-motion (SfM) library can be tuned to run until it reaches some quality threshold associated with the 3D model. SfM was the final algorithm investigated for the reconstruction of roads. Elastic Fusion has typically been used over small spaces with a slight transformational difference between video frames. Methods to extend dense 3D reconstruction over larger areas include Kintinuous [186]. A camera-mounted vehicle will typically traverse much larger distances between video frames and should be used with algorithms that accommodate large baseline differences between images. Changes may be required to use the Elastic fusion system in that environment.

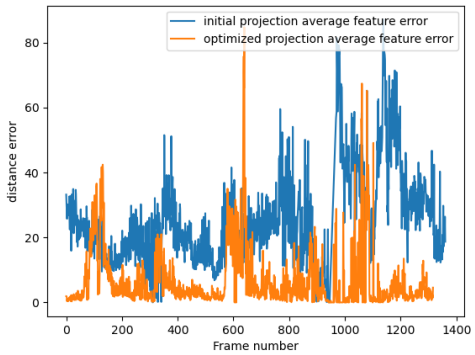


(a) Initial Projection.

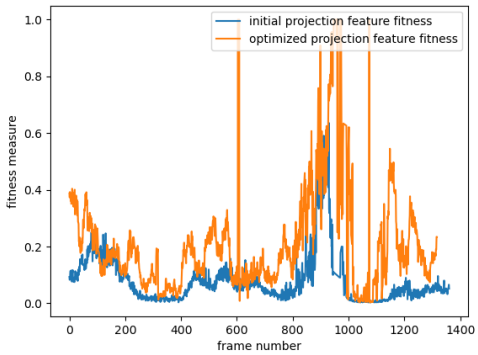


(b) Optimized Projection.

Figure C.7: Image feature matches (a) before ICP optimization and (b) after ICP.



(a) Feature error measure.



(b) Feature fitness measure.

Figure C.8: Performance of features matching across all image in 'lr kt0' sequence from ICL-NUIM [68]

The reconstruction using the Elastic Fusion algorithm with RGBD images provided an accurate 3D model and set of poses for each image. The poses for the images could later be used as transformations for the projectors. However, this method is not possible using a standard RGB dash camera without extra features being built to compute the MVS or predictive depth estimation [89]. It was intended that a road engineer could use this system to reconstruct real roads using only a standard camera instead of a specialized vehicle. RGBD cameras, although cheap, do not typically work in daylight, and Elastic Fusion has not been developed with the appropriate parameters for wide baseline differences in video sequences. This setup worked well for commissioning the integration of the outputs of a 3D reconstruction system, the 3D mesh model, and a set of image poses with PTM.

C.3 Conclusions

The ICP optimization had a focus on improving view-dependent visual effects. Lighting variability from each view was only considered peripherally as specular reflections were only observed from the projection of other images. A discrete projected image was aligned with the 3D model. There was no continuous representation of a light radiance field for each 3D point. Later in the same year (2020), researchers in machine vision released a paper on Neural Radiance Fields (NERF), which carried out a per-point optimization over the 3D model and attempted to estimate the direction the light radiance reflected for some set of views [111]. This work supports that lighting and colour should be optimized based on view-dependent instances instead of treating 3D surfaces under Lambertian world assumptions. Using Lambertian world assumptions has proven to be an excellent approach to reconstructing 3D scenes geometrically. However, it is the belief of the author of this thesis that this should be a temporary solution used only for reconstructing the geometry. View-dependent shading should be used to reconstruct appearance/colour in these virtual environments. The solution to fitting images to a complete 3D model improved the overall reprojection error. Bundle adjustment techniques optimize the reprojection error for the entire set of images and the triangulated 3D points. For this reason, 3D reconstruction systems such

as COLMAP, which use bundle adjustment, are used in Elastic Fusion's place.

Appendix D

Driving Simulation Runtime Features

Chapters 4 and 5 described the reconstruction of the 3D environment and rendering methods for use in the simulator developed. The following sections describe the components required to implement real-time projective texture mapping in a driving simulator. It also describes how user input was used to adjust the vehicle's motion and the driver's view. The following are the contributions of this chapter:

1. Given image labels for each semantically labelled image a tracking algorithm was applied to improve the robustness of retrieving vehicles.

2. A view selection method developed to find images and their pose matrix for projective texture mapping.

3. Projective texture mapping (PTM), inpainting, foreground removal/background subtraction, and shadow mapping can be computationally expensive. Instead of considering these as a stream-based process, some processing steps were completed in batch style before simulation. This required the creation of specific storage and data types to hold the intermediary data produced by these batch processes for use during run-time.

4. There is a requirement to read 3D models into memory from disk in real

time; to date, it has been assumed that the model has always been present[1] for rendering. The approach to reading these models from disk is discussed here.

5. The motion of the camera and vehicle between rendered views has not previously been considered and is explained here.

This chapter will discuss in detail how each of these was addressed. The driving simulation will be limited to specific paths. The rendering camera will begin with an orientation that aligns with the set of images used during the data capture of the 3D model. It has been shown in previous experiments that photo-consistency metrics for projective texture mapping perform best when the rendering camera is aligned to the extrinsic parameters/view matrix of the projectors, see Chapter 4. It is recommended that the driver remains on a specific path via instructions provided by the experimenters. The driver must progress along the road in the same direction as the recording camera. The playback view is limited to $\pm 45^{\circ}$ concerning the forward direction of the recording camera. The driver is required to drive on the same side of the road as the recording camera.

3D points of the reconstructed model can be textured with an image of a real road to produce photorealistic views using projective texture mapping. The optimum projector needs to be identified for each position the driver occupies. The poses of the projectors relative to the 3D model were compared to the rendering camera. The pose with the minimum difference was selected for projective texture mapping. The difference metric used was the Euclidean distance between the pose of the projector and the rendering camera. Finding local minima between these two sets was computationally expensive at run-time. A linear search for the minimum Euclidean distance would result in order O(n) search times [36]. The run-time should be fast enough for the simulator to remain responsive to the change in the pose of the rendering camera. The simulator should maintain >15 frames per second run-time. Using a Monte Carlo style test, a variety of search methods for the projector search problem was compared. This involved creating a synthetic dataset of random poses and a random query pose for each test. The

[1]It is not being said that the model still needs to be reconstructed. The read from disk time has previously been assumed to be completed.

time and accuracy were recorded for each test. Previously it was described that projections should only be used on surfaces that are front facing towards the projector. Shadow mapping was used to achieve this. This process was computationally prohibitive. During development, projective texture mapping could achieve ≥ 60 frames per second. However, shadow mapping required depth maps from the projector-camera pose to be stored in a frame buffer and read later. This process increased the computation, and the frame rate dropped below 30fps. Pre-baking depth maps ahead of simulation removed this processing time. Frame rate consequently returned to 60fps. Similarly, inpainting/background subtraction was required for instances where the image contained vehicles or pedestrians, but the final 3D model or mesh did not. Inpainting was carried out in a batch before the simulation. The inpainting workflows described in Chapter 5 required greater run times and were not implemented into a single stream process. The resulting images were stored on disk and read with their respective 3D model/mesh.

3D models and meshes can be read from the disk at the start of each simulation. However, there is limited memory capacity for both RAM and GPU. Visual Studio 2022 often has a limit on the virtual memory at 2GB. However, it can be expanded to 4GB [176]. Creating a 3D model/mesh of an entire city would quickly exceed this memory limit. This was before considering textures, depth maps, or pose data that would also have to be read. The 3D model/mesh had to be read in segments to prevent scalability from being an issue. The segments must be small enough to fit in memory but read fast enough not to detract from the immersion of the simulation. The 3D model should be read in the background, and the driver should be unaware of this. The time taken to read models during simulation should avoid impacting the game logic and rendering. To load all data relevant to a specific segment of road, the 3D mesh, textures, and pose meta-data were stored under shared folders. This folder hierarchy allowed the simulator to load all appropriate content for a given road based on the position of the rendering camera.

A motion model was constructed to enable steering and forward motion within the simulator. A vehicle's steering has been identified as a key factor for measuring human driver responses because steerable simulation supplies higher fidelity. Most driving simulations include this ability, for this reason, [182, 49, 55, 157].

An arc ball steering model was used, and a decaying acceleration model was used for the vehicle's forward motion.

D.1 Vehicle Tracking

Chapter 5 discussed the removal of foreground objects, specifically vehicles. To remove these objects from video they must be localized in each image. Semantic segmentation was used to identify where these vehicles were. However the segmentation network does not have 100% accuracy [199]. Given the input data is based on video and there exists a spatial consistency over most frames of video, machine vision techniques were applied to counter false positives and false negatives identified by this network.

D.1.1 Tracking for 2D Inpainting

Section 5.2 presented an inpainting workflow that would work in most cases. However, the same surface observed from different perspectives or under different lighting conditions can provide variable photo-consistency and this had a significant effect on the performance of image template matching techniques. To overcome these variances, temporal and spatial consistency was used to find neighbouring regions containing the same image labels. For example, if image I_q^0 matched at an image location $(\hat{x}, \hat{y})$ in image $I_{\hat{q}}^0$ for the image template located to the left of the vehicle cluster, then it would be likely that the next image I_q^1 would contain a match in $I_{\hat{q}}^1$ within proximity $(\hat{x} \pm \mathcal{A}, \hat{y} \pm \mathcal{B})$. This would be based on some threshold that limits the degree to which these templates can shift frame-to-frame within the video. Other sanity checks can be applied including ensuring that the top, left, right, and bottom templates maintained the same orientation within the image.

Semantic segmentation identifies each pixel of an image. It would later be a requirement to identify instances of vehicles to enable the tracking of these objects between images within a video. Rather than developing a network for tracking, the existing segmentation network was used. Classic machine vision techniques were applied to the masks provided by this network. To localize the

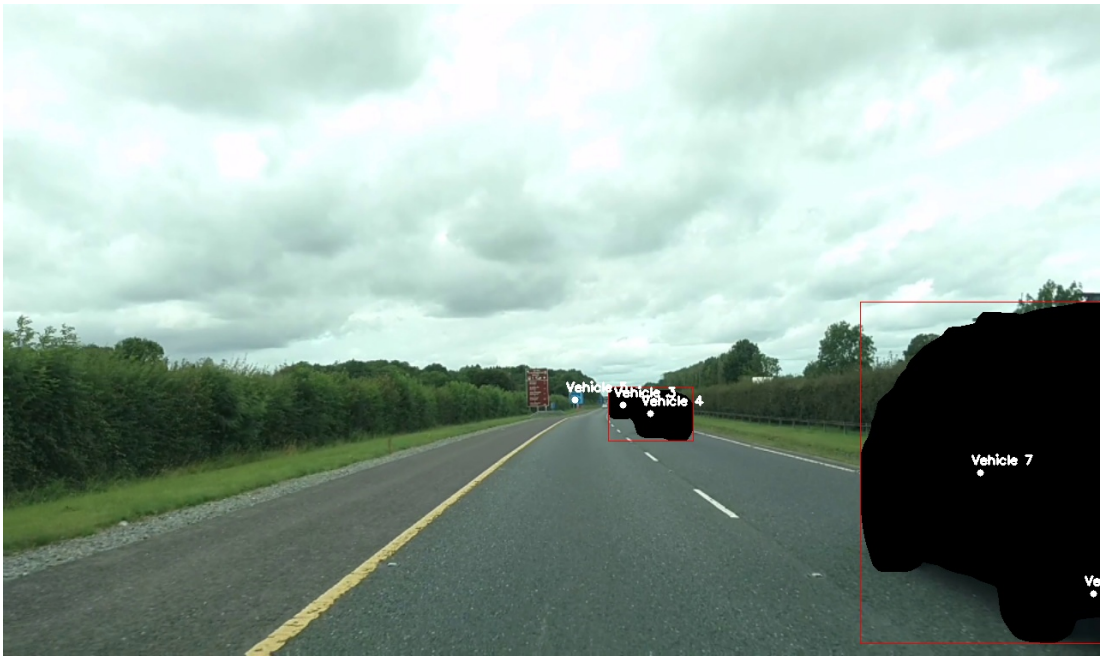


Figure D.1: Cluster centre tracking.

centroid (the average position of the set of pixels labelled as a single vehicle) of the labels the images were converted to binary images and a contour finding method (**findContours()**) was used through OpenCV [12, 159]. A bounding rectangle was calculated for each set of contours. This was identified by acquiring the minimum and maximum horizontal, and vertical coordinates corresponding to each instance of some vehicle. Each centroid was contained within a list, φ^t, where t corresponded to the index of the frame number for a given video sequence. Each centroid within the list had a set of attributes including size, center-point, and visibility. The list φ^t was updated after each new labelled image was provided. If a centroid did not reappear after some set number of images (60 images), the centroid was removed from the list and not queried again. The image labels were classified into clusters using a centroid tracking algorithm based on this article [135]. The labelling of centroids between images was based on a nearest neighbour's algorithm using Euclidean distance within the image plane. Figure D.1 shows an example of these regions being tacked within bounding boxes. Instances where some images were not labelled as vehicles due to misclassification,

could be rectified by applying this tracking algorithm. For example, if the labels were detected in image I_p^n but lost between $I_q^{[n+1...m]}$ and then detected again in image I_q^{m+1} the tracking of these labels could be interpolated, assuming the gap between m and n was not large. The original tracking algorithm described in [135] was responsible for identifying whether centroids were related to each other and the interpolation between images was an extension we added to this during implementation. These interpolation-based methods were applied after the centroid tracking algorithm was finished. It followed a similar approach to eye-tracking and fixation labelling [117].

D.1.2 Tracking for 3D Inpainting

To apply approach to inpainting described in Section 5.3, the following improvements were considered. A tracking algorithm was developed to take the labelled videos and make them robust to single-frame misclassification. Misclassifications in terms of a vehicle attributes were not corrected previously. Each semantically segmented blob was characterized to account for these misclassification instances. The characterization provided a unique identifier for each label. These characterizations were tracked across successive frames of video to correct for outliers.

Image Labelling in Video: Semantic segmentation in the previous section was considered discretely per-image. It did not utilize the chronological advantages of video. Inference of image appearance can be gained from vertically and horizontally adjacent pixels and temporally. An assumption can be made that pixels within an image, $I_p^t(x, y)$, labelled as a vehicle, are more likely to contain a vehicle pixel if the previous m images at the same position, $I_p^{[t-m,t-1]}(x, y)$, if the scene is static and the camera is not moving. Alternatively, suppose vehicle-labelled pixels are approaching some position $I_p^t(x, y)$ in the previous set of images. In that case, it may be likely that this current pixel within the image could be labelled as a vehicle. Optical flow has been a popular area in machine vision that would have previously dealt with scenarios similar to this [100, 102]. More recently, approaches using prior probability distributions for image labels while using an image labelling model could have been applied. This work is done with recurrent neural networks and long short-term memory (LSTM) [134]. Previous work has

shown that image labelling can be improved with more measurements, including labels over a 3D model, see Semantic Fusion [106].

There are some key differences between the problem presented here and what is normally done in the state-of-the-art in machine vision. The setup in this thesis for reconstructing real roads and removing foreground objects uses a fixed set of videos, not a continuous stream (batch processing vs. stream processing). There was no need to make a prediction based solely on prior information. For this reason, the act of tracking attributes related to semantic labels were treated in a similar fashion to eye tracking. The representation of each labelled vehicle was contained in a list of centroids, φ^t. These could be processed similarly to eye gaze tracked and classified into fixations, see Appendix A.1.4 for more details on this process. A labelled vehicle was often classified correctly; however, a small percentage of the labels were incorrect. This was noticeable to humans watching a video while rendering the tracked vehicle with a bounding box, but statistically, this would be a very low false negative rate. The model to attribute semantic labels to each image had a pixel accuracy percentage of 80.13% on the MIT ADE20K dataset [74, 197, 199]. This effect is apparent when vehicles suddenly re-appear in a simulation of an empty road. Assuming this did not happen often, signal processing techniques could be applied. Among those signal processing techniques, linear interpolation (from the previous section) was used to rectify issues when there was a total loss of tracking and Gaussian convolutions was to smooth the effects of misclassification of centroid attributes (size and location).

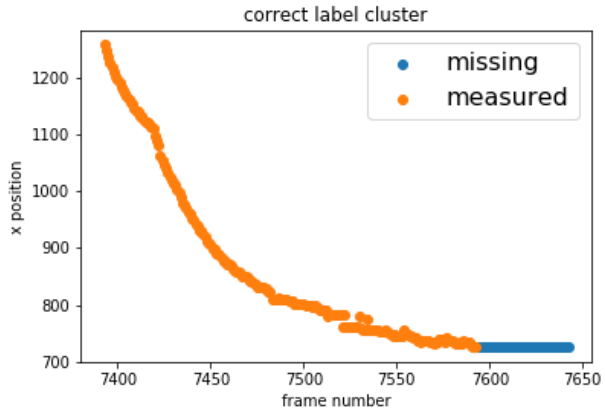
Other vehicle tracking factors to consider include; mislabelling of the vehicle pixels (false positives). False positives introduce unnecessary inpainting. Euclidean distance between each occurrence of a centroid over each image was compared against a threshold distance. Large jumps for centroids were not considered to be the same instance of a vehicle. Occlusions of vehicles could cause tracking to be lost, but this would be more critical if the instance of the vehicle mattered, which is not the case here. For this use case, centroids were no longer tracked if they had not been detected for more than 50 frames of video.

In practice, there were very few false positives. However, false positives associate themselves with the same centroid (a labelled blob whose characteristics are tracked over time). Figure D.2 observes three typical cases while labelling

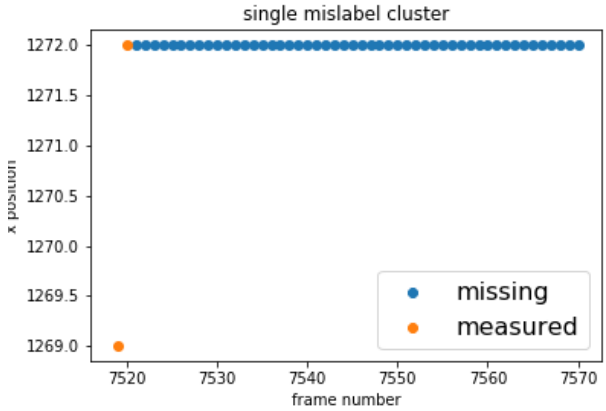
these sequences. The top graph represented the tracking of a correctly labelled cluster of vehicle pixels on an image plane, with the horizontal position of the vehicle on the y-axis and time on the x-axis. Orange points represented the positive detection of a centroid, and blue points represented the negative detection of the centroid. Figure D.2a highlights the presence of the vehicle vanishing to the edge of the image. This contained more positive instances of the centroid than negative instances, reinforcing the likelihood that this was a true positive. Figure D.2b highlighted a single false positive. This was easy to detect and was a likely occurrence given that the network labelling these images did not have 100% accuracy. Measuring the centroid's lifetime could be used to detect these instances. Figure D.2c, however, contained a challenging example where multiple mislabeled centroids were detected and associated with a single instance. The way to detect these instances was to compare the ratio of positive to negative instances of vehicle centroids.

D.2 View Selection

This section describes how the image pose data was queried at run-time to ensure that an appropriate image, or set of images, was used for projective texture mapping. The texture selection for projection was based on maximising the similarity between the camera's pose that captured that image relative to the reconstructed 3D model and the rendering camera within the driving simulator. The similarity was based on Euclidean (spatial) distance between the two poses. Three approaches were considered for searching for these matching poses. All tests were evaluated in Python. An initial optimized linear search approach was applied using the NumPy library. This was compared against a scratch-built search method that used a k-dimensional tree (KD-tree). This data structure depended on the pose similarity between images used for 3D reconstruction. These approaches were tested on a set of randomly generated poses surrounding the odometry for driving sequence obtained from KITTI [63]. There were $10,000$ random poses sampled, and as a result, there was a high likelihood that a suitable match existed for the driving sequence to query. The KD tree was tested and compared to a linear search. It was predicted that the linear search would have a lower



(a) Labelling true positive in video sequence.



(b) Single false positive in video sequence.

(c) Multiple false positive in video sequence.

Figure D.2: Examples of centroid tracking for common true positive and false positive instances.

residual error while choosing a projector pose to align with but would take more time to query than the KD-tree on average. The prediction was based on the Big O complexity of the search using these data structures. The NumPy linear search uses compressed array structures and vectorized methods. The underlying C-code was pre-compiled and did not require the iterative interpretation of byte code for each comparison associated with using loops and lists in Python. The KD-tree, however, did not use this vectorized library and was developed from scratch.

D.2.1 View Selection Algorithms

The images were associated with a KD-tree data structure to assist in selecting the appropriate image for perspective projection. This tree was computed before the simulation run-time and parsed with the image data at initialization. In Figure D.3, the left graph shows the pose of a camera given an aerial view (just the x-y coordinates). These poses were contained within a pose tree data structure. Each node grouped all the poses within some boundary. The resolution of each node increased with the depth of the tree. This can be seen in the right graph. Each bounding box for the first two layers was linked to a node in the tree. This tree stored every image name with its respective pose. The meta-data of all the underlying poses was stored in nodes closer to the root. Poses within the same node were more like each other than nodes in other parts of the tree, based on Euclidean distance. The tree's leaves contained image names and pose information. The tree was constructed by splitting the root nodes' poses into a grid of nine sections. Poses contained within a single area were associated with their respective node. The section was split into nine nodes if the child node had more than nine poses. This process was repeated recursively until less than nine poses were contained in a node. The higher-level hidden nodes had attributes associated with their children, including the average coordinate, standard deviation of coordinates, and the minimum and maximum positions of the images in the leaves of their sub-trees.

This structure allowed for faster search times for the images with close alignment to the rendering camera. A comparison of the rendered pose against the

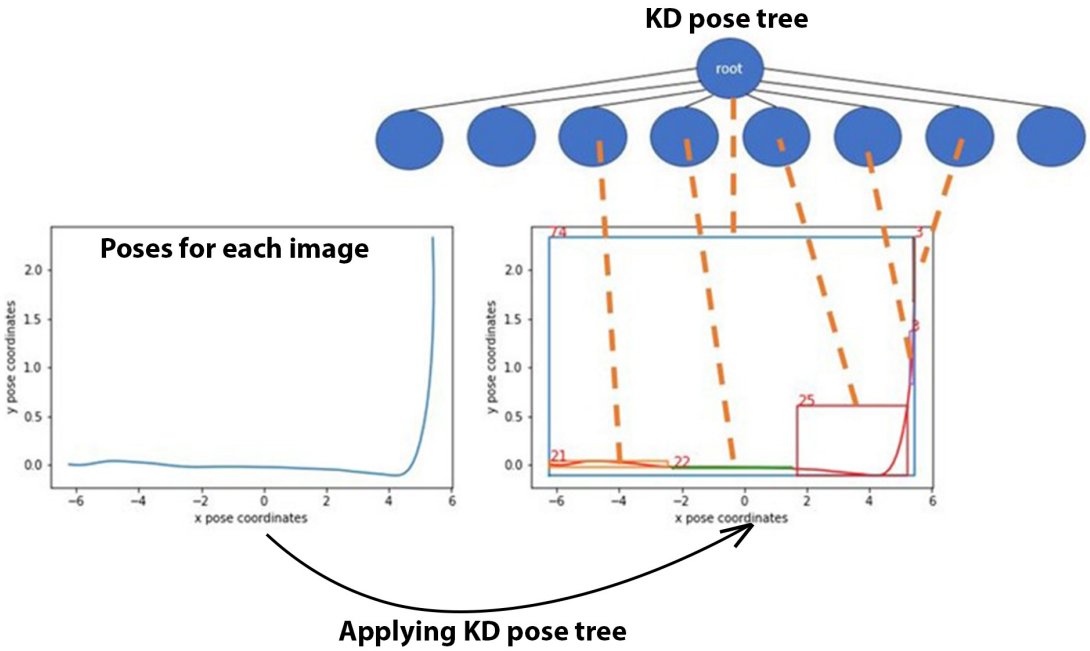


Figure D.3: 1 Video sequence odometry (left) and how it was distributed across a KD tree (right). Each root node of the tree is connected to a boundary of the map the image poses are located associated with.

nodes in the tree, starting from the root node, ensured the search space for the image was $\log_k(n)$, where k was the dimension of the maximum number of children for each node in the tree (k=9), and n was the number of images in the database being queried. Following the first search within this tree, images were quickly found by searching neighbouring nodes in the sub-tree and leaving the sub-tree when some threshold distance was elapsed, see Equation D.1. The variance and mean of the poses within the node and coordinate of the rendering camera were represented by σ, μ, and p, respectively. The mean and variance at each sub-tree were pre-computed and stored in the tree data structure. These values were compared against the current pose. $2\log_k(n)$ was the worst-case run-time for this improvement on the search algorithm. The worst case often occurred once per sequence when the vehicle travelled from one side of the 3D model to the other. If the search did not have to leave the occupied sub-tree from the previous search, the run-time was only $k(9)$ comparisons.

$$2\sigma - |\mu - p|_{L2} < 0 \tag{D.1}$$

D.2.2 Testing View Selection

To test these search times a sequence of poses was compared against a randomly distributed synthetic poses surrounding a neighbourhood of the original driving sequence from the KITTI dataset. This was done to demonstrate the scalability of this approach while showing the data structure did not have to be dependent on video sequences, like binary search, and could accommodate sequence spatial overlap. A linear search method for comparing the distances of every point in the dataset against each incoming query pose was considered, see Figure D.4. The top image highlighted the path of a camera from the KITTI dataset. The blue dot was the query pose to the KITTI sequence. The bottom graph showed the Euclidean distance between the query and all other poses in the sequence above. There was a local minimum over the shared x-coordinate matching the query pose. Linear search was compared to the KD-tree on the same randomly distributed dataset. The KD-tree was tested with both the smart-cache search strategy and the naive strategy. The smart-cache search used the previously

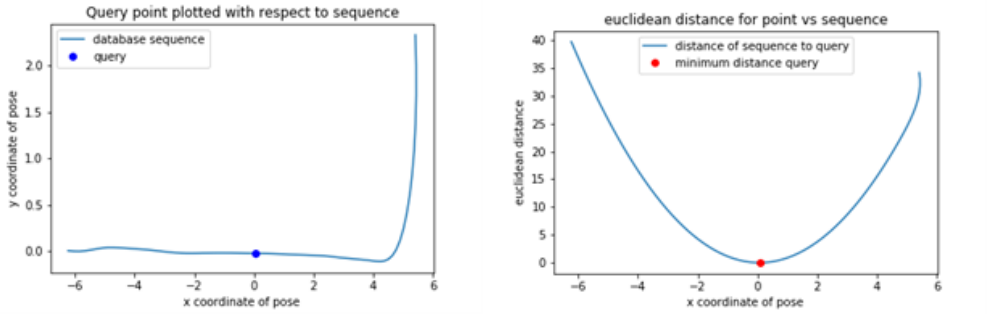


Figure D.4: Comparison of top-down camera poses (left) and the relative distance between some points in the camera pose sequence and all other points (right).

Method	Average Search Time (seconds)	Standard deviation search time (seconds)	Average Euclidean Distance (model scale units)
Numpy linear search	290×10^{-6}	167×10^{-6}	41×47^{-3}
Root KD-Tree	382×10^{-6}	120×10^{-6}	49×52^{-3}
Smart KD-Tree	189×10^{-6}	105×10^{-6}	25×23^{-2}

Table D.1: Run-times for each camera pose search strategy

acquired pose nodes to begin their search for the next closest node. The naive search required each search to begin from the root node down to the solution in every case. All cases computed the Euclidean distance between the query and points in the database. The search times were highlighted in Table D.1 and Figure D.5.

D.2.3 Summary

Linear search always returns the pose with the minimum residual error. The KD-tree trades-off minor pose residuals for shorter search times. While considering which nodes to enter or leave, the KD tree could change its search strategy to compare the boundaries of the tree instead of the average. This trade-off may provide better solutions for uniformly distributed data. However, there would be no guarantee that poses searched near the boundaries would provide a solution closely matching the query. In most cases, comparing the query pose against the

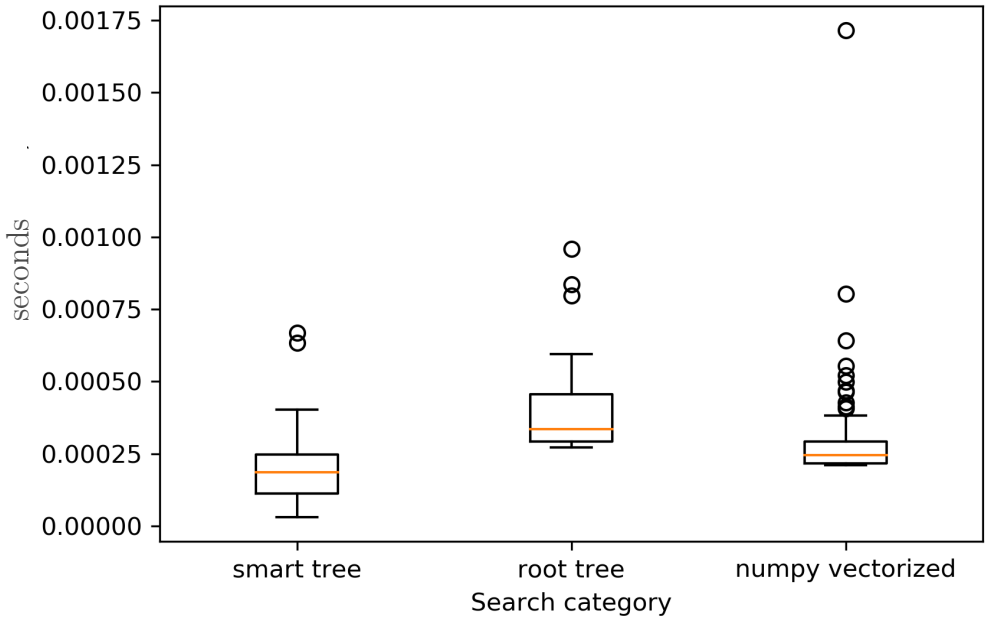


Figure D.5: The search time for each respective method.

average pose of the nodes in their subtree increases the likelihood of finding a well-aligned pose. In most cases, a sparse set of poses will be used for reconstruction, as seen in the data collection sections in Chapters 4 and 5. The density in the number of poses would be higher near the mean pose, given that they were collected in a sequence. The KD-tree was the most appropriate method for run-time within these tests. NumPy uses densely packed arrays of the same type and often carries out operations in parallel. Numpy operations and methods on their arrays typically outperform Python lists in terms of run-time. The tree searches tested in this instance contained nodes, and within those nodes, Python lists. The smart tree was faster despite not using parallel programming or densely packed memory optimizations. This was shown based on the average search time and standard deviation search time columns in Table D.1 respectively.

D.3 Asset Loading

All items[1] included in a graphics pipeline must be read from disk in some fashion, processed by the CPU, and passed to the graphics card for rendering. Specialized data structures, storage types, and specialized algorithms were employed to do this efficiently. There were three asset types loaded for this driving simulation. These included the 3D model(s)/mesh, images used for projective texture mapping, and the camera pose and model metadata. The rationale for using the assets will be described. The process of separating the sections of the road into fragments and labelling them to later bring them back together will be described. The storage of these assets will be described, and how they are brought from disk to computer memory will be detailed.

D.3.1 Partitioning the Road

It is possible to read the 3D model from disk in one chunk, but this is not scalable. If 3D reconstructions were developed that spanned a large enough space, the 3D model would not fit in RAM and GPU memory. It was proposed that the 3D model be read in smaller fragments. Within the 3D model, there was a set of poses, as described in the previous chapters. Each pose corresponds to a translation and rotation of the captured image. A 2D map was constructed based on each pose's vertical and horizontal coordinates as observed from a top-down view, see Figure D.6. These poses were estimated by creating a sparse road reconstruction using ORB-SLAM2 [116]. This provided an overview of the structure of the road before the full-scale dense 3D reconstructions. The fragments of the reconstruction were based on the segmentation of this 2D map, forming a topological map. Each fragment corresponded to a different segment of the road. In Figure D.6, each segment was labelled with a number, and each neighbouring road segment was given a different colour to highlight each road's beginning and endpoints. Each section was labelled manually based on correspondence to a separate junction or a straight road. During development, it was noted that if the reconstruction ended at a turn or junction, the final mesh often contained many

[1] 3D models, textures, shaders, and any file descriptors influencing rendering or system logic

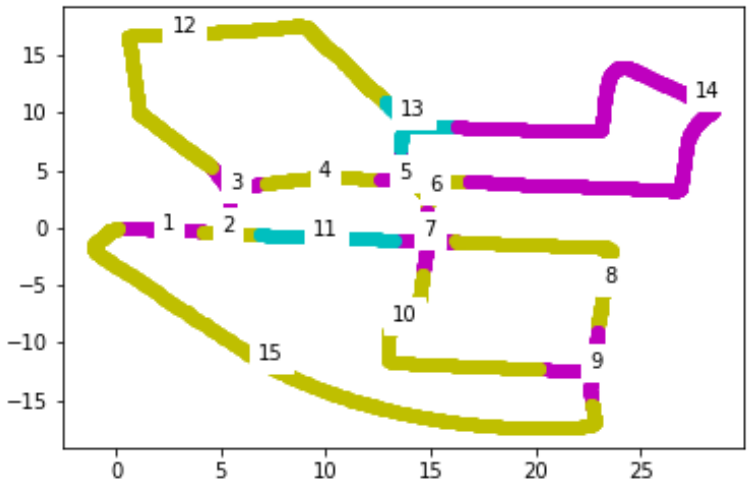


Figure D.6: KITTI routes segmented to create a topological map (sequence 0).

outliers that were geometrically inconsistent with the ground truth of the scene. The fragments were separated in this way for this reason. 3D reconstructions of each road were made using COLMAP, and their corresponding textures were stored within the same file hierarchy as the 3D model/mesh.

The 3D reconstruction process for these roads was carried out using COLMAP, see Chapter 4. However, some modifications were made to this process to ensure the models had consistent attributes. These attributes included consistent scale, continuity of image poses, and consistent coordinates and orientation of shared 3D points for each fragment. The image poses from ORBSLAM2 were used as priors for the triangulation step in COLMAP to provide consistency between the 3D fragments. Given these seeded priors, it was hypothesised that COLMAP would likely fall into the same local minima as ORBSLAM2 while carrying out triangulation and bundle adjustment. The primary appeal of this approach was that these priors would likely result in uniform scale across the final 3D meshes, similar to stereo cameras with a known baseline. The second benefit of using these priors was that each of the image poses and 3D points for each fragment would be contained within the same coordinate system. This spared further processing to compute the transformations required for moving from one fragment coordi-

nate system to the next while rendering these models. Notably, the incorporation of seeded priors for the pose of each image from ORBSLAM2 resulted in fewer iterations of bundle adjustment during fusion and more detailed meshes in the final 3D reconstruction during multiple view stereopsis. ORBSLAM2 was specifically built for solving the localization and sparse mapping problems, whereas COLMAP focussed on providing an entire structure from motion and multiple view stereopsis pipeline [139]. ORBSLAM2 was also initially benchmarked on the KITTI dataset. ORBSLAM2 performance on this data and other roads would likely perform better than in other environments.

The poses/odometry from ORBSLAM2 did not port directly into the COLMAP pipeline. There were multiple items to address before this could be fully implemented. ORBSLAM2 provided odometry following the completion of a sequence in a keyframes file. A subset of poses was localized from the overall set of images. These were referred to as keyframes. This localization occurred when enough new information entered the view of the reconstruction system based on the images provided and the features they contained. Two possible solutions for this were 1) to edit the ORBSLAM2 codebase to localize all images regardless of the information criteria or 2) to interpolate between the keyframes to find the approximate position and orientation of all the images between neighbouring keyframes. The latter option was used. The format of the position and orientation of each keyframe from ORBSLAM2 was $[t_x, t_y, t_z, q_x, q_y, q_z, q_w]_{ORB}$, where t was the position and q was the quaternion of the pose. These were provided in world coordinates. However, COLMAP expected these poses in the format of $[t_x, t_y, t_z, q_w, q_x, q_y, q_z]_{COLMAP}$, and these poses were given in camera view coordinates rather than in world coordinates. Since these two mapping systems operated using different coordinate systems, it was necessary to create a conversion between them, see Equation D.2. R_C and R_O were the COLMAP and ORBSLAM2 rotation matrices, respectively. They were converted from quaternions to rotation matrices prior to this. The translation of each pose was represented by t_C, and t_O for COLMAP and ORBSLAM poses, respectively. This equation highlighted how ORBSLAM2 pose coordinates could be directly converted to COLMAP pose

coordinates.

$$\begin{bmatrix} R_C & t_C \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_O^\intercal & -R_O^\intercal t_O \\ 0 & 1 \end{bmatrix} \tag{D.2}$$

The pose data was labelled and saved to a text file labelled, "images.txt", and a database on a table labelled "images" that would be used for reference by COLMAP throughout the SfM and MVS processes. Camera intrinsic parameters were saved to a file labelled as "cameras.txt" and committed to the database under the table labelled cameras. A blank file labelled "points3D.txt" and a table labelled points3D were also created. Given the new set of poses from ORBSLAM2, COLMAP was then used to extract image features (SIFT), match those features, triangulate the image features, and use bundle adjustment to optimize the poses of each image relative to the 3D features. COLMAP used the ORBSLAM poses as priors before starting the triangulation step. The ORBSLAM poses contained residual errors between the ground truth poses of each image relative to each other because of the previous interpolation step. Further iterations of bundle adjustment and triangulation could optimize the set of poses and final 3D model. The images were undistorted and then mapped into depth, and normal maps for each image using the patch match algorithm [140]. These depth and normal maps were fused to produce a single mesh (PLY) Polygonal/Stanford file format [10] using Poisson mesh reconstruction.

D.3.2 File Structure

The workspace used for the driving simulation is outlined in Figure D.7. Within this hierarchy, the top folders contained Models and Shaders. The Models folder contained all data relating to assets used during the simulation, including 3D models/meshes, textures for the 3D models, and depth maps. These depth maps were created based on the final 3D mesh instead of the depth maps estimated from the patch match algorithm. Pose data was used for the projection of each image. Camera parameters were used to describe the intrinsic parameters of the camera used to capture the images and, consequently, its projector. The shaders folder contained the GLSL code that would be compiled and run on the

graphics card. These shaders described how 3D points should be interpreted and rendered in Chapters 4 and 5. Within the Models folder, there were several sequences, each corresponding to a different KITTI video sequence. Within those folders, the sequences were broken into several parts/fragments. The numbering of these fragments corresponded to the numbering used for either the graph or queue used during runtime while loading each fragment. The fragment folder was a PLY file labelled "meshed-poisson.ply". This was the 3D model that was loaded during runtime. Shader programs read each 3D point in the mesh during runtime. There were two folders within this directory, "images" and "sparse". "images" was used to store files that would be used for texturing. The "textures" folder contained images from the reconstruction, some of which had undergone inpainting (depending on the presence of foreground vehicles that did not present in the final 3D model/mesh). The "depthMaps" folder contained depth map images computed from the depths observed from each camera pose relating to each image in the "textures" folder. These were used for computing shadow projections for each image in the "textures" folder. Within the "sparse" folder, there were two files: "images.txt" and "cameras.txt". These were the duplicate files used during the 3D reconstruction for COLMAP. "images.txt" listed each pose for each image found in the "textures" folder. This was read during runtime and was used to orient the projector during projective texture mapping. "cameras.txt" contained the camera intrinsic parameters. These were read once at the simulation's start and applied to all matrices related to the projectors used for projective texture mapping.

D.3.3 Reading from Disk

Each fragment file was read into CPU memory during the simulation and transferred to GPU buffers. The selection for each fragment to load depended on the rendering camera's location during the simulation. Euclidean distance was used to determine which of the fifteen fragments was closest to the rendering camera. When a fragment was read from the disk, its neighbours were also read into memory. The load times for these models and their corresponding textures could take longer than the time it took for the driver to complete a section of the

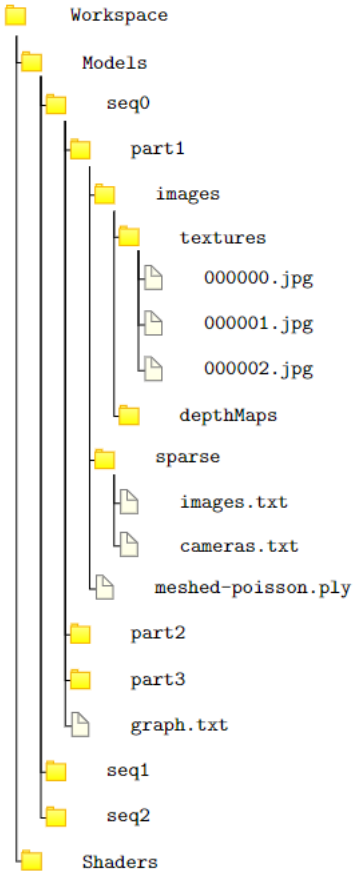


Figure D.7: File structure for simulator workspace.

road. Knowing ahead of time which models to load required a topological map to inform the model loader which roads were connected. The topological map had two possible representations: a queue or a graph. The queue introduced the constraint that the 3D map would be read in a specific order. Most likely related to the original trajectory taken during data capture. A graph allowed the user to take alternative routes while in the simulation. Roads were loaded based on their adjacency and connectedness conveyed in the topological map data structures.

For each 3D model, the set of corresponding textures was loaded. It was assumed there would be a single disk that contained all assets (3D models and textures), with the file structure in Figure D.7. The images were read from the disk after the model was parsed, and images were stored in chronological order. A separate folder was used for each texture type, including a folder for the colour images and their corresponding depth maps. Before the simulation, inpainting was completed as a batch, removing the need to do inpainting/background subtraction with multiple images in real-time. In memory, the depth map and colour images were stored in a 2D array of textures that could be passed as uniforms to OpenGL (the GPU).

A maximum of five models were in memory at any one time. Fifteen models were created following the creation of the 3D model of a road from sequence 0 from the KITTI dataset. Each disk read required a minimum of three roads (3 meshes): a straight road with two neighbouring roads, one ahead and another behind, giving the option to reverse. The maximum number of disk reads required five roads: a junction with a road ahead, behind, to the left, and the right. Examining the sections of the road in Figure D.6 that each section of the road has between 2 and 4 neighbours. Reading neighbouring models and the model the rendering camera currently occupied ensured the next required model for rendering was available. When the rendering camera transitioned from one road to a neighbouring road, the other roads were released from memory, and the neighbours of the following road were read from the disk. The last section of the road and the next road would not have to be reread. As a result, after initialization, the maximum number of roads that would have to be read at any one time would be three. This is because the road the vehicle is coming from and the road it is going to are already in memory.

Node	N1	P1	N2	P2	N3	P3	N4	P4
1	2	101	15	3011	-	-	-	-
2	1	90	3	200	4	505	5	1011
3	2	190	12	300	11	400	-	-

Table D.2: Example format of Graph.txt. The N and P followed by a number in each column represents neighbour and pose respectively.

The reading order of the 3D models was determined by parsing a "Graph.txt" file. See Table D.2 for an example of the structure of this file. Each row of this file corresponds to the node's name in the graph (a 3D model from the sequence). The first column was the name of this 3D model. The following columns corresponded to the names of neighbouring nodes and the label of the pose that should be queried for checking the Euclidean distance between the rendering camera and the next model to render. The node's name and its respective query pose were in ordered pairs. Each pose for each neighbouring node acted as a dynamic weight for the edges between nodes. The distance between the rendering camera and each neighbouring node was compared, and the rendering node was swapped for a neighbour when a threshold distance was met. The neighbours of the neighbour would be read into memory while the neighbours of the previous node would be de-referenced and deleted unless they were the current node or the last current node.

Reading the models from disk was completed with a separate thread from the main rendering loop. A background thread read these 3D models and textures from the disk while the rendered scene was occupied. Transferring the model data to the GPU was completed as part of the main rendering loop; otherwise, multiple OpenGL contexts would have to be set for each thread [187]. The vertex buffer and index buffer transfer to the GPU would have to be processed in small segments to allow the rendering loop to continue to steam feedback to the driver. The simulator's current implementation does not allow for this. As a result, the final demonstration reads the first 3D mesh and its neighbours exclusively. After reaching the end of the neighbouring mesh, the simulation resets. This creates the effect of continuous forward motion using a limited data set. It also demonstrated the dynamic model switching required if more extensive models were used.

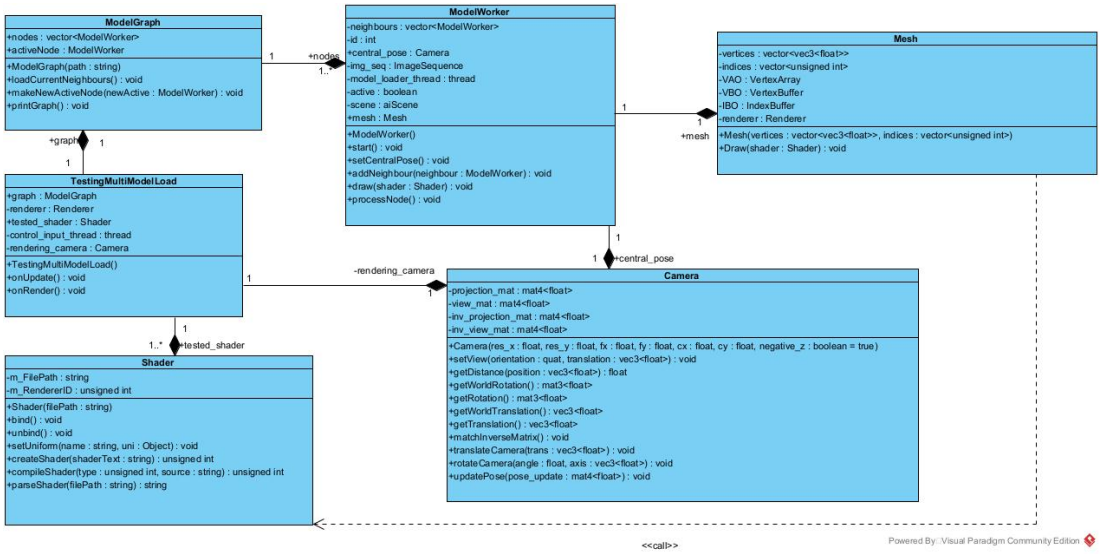


Figure D.8: Model reading and rendering class diagram.

The above observations were abstracted into classes; see Figure D.8. The main rendering loop was tested in a class named "TestingMultiModelLoad". All other classes in this diagram had a dependency on this class. This class created instances of the Shader to process the graphics instructions for rendering and projection. See Chapter 4 for details on these shaders. A camera object was created to enable updates to the perspective of the rendering camera as the 3D models were viewed. A "ModelGraph" was initialized. The "ModelGraph" read "Graph.txt" and a "ModelWorker" was initialized for each node it read from this file. The "ModelWorker" created a pointer to a mesh that reads from the disk when instructed by the "ModelGraph". Each "ModelWorker" retained a record of their neighbours using a vector of pointers to other instances of "ModelWorker". Within the "TestingModelLoad" main thread, the "ModelGraph" was queried every render cycle to check if the currently rendered model should be swapped for a neighbour. If this was true, they were swapped, and the neighbour was rendered immediately while a background thread fetched their neighbours' models from the disk.

Summary: the processing involved in reading 3D models and textures/images from disk memory was detailed. The solution to this is scalable where the limiting factor is dependent on the number of models saved to disk. The use of graphs

and queues has helped to inform which models are connected and in what order they should be read from the disk. Multi-threading and carefully chosen data structures have been used to minimize observable delays during run-time.

D.4 Vehicle Motion

The driving simulator's rendering accepts poses for the rendering camera and uses projective texture mapping over a reconstructed 3D model to provide a photorealistic stimulus for the driver. Allowing a driver to control the vehicle requires an implementation of the camera movement about the model. The vehicle's motion is simulated using an arc-ball model where the camera can translate and rotate about its axis in any direction. There is total manoeuvrability possible in the environment. Through vehicle controls, however, the motion of the rendering camera is limited to the forward and backward vectors and steering about the yaw axis. The control inputs of the vehicle were handled by a separate thread that polls the actions of steering wheel and pedal controllers. The arc-ball camera provides a practical driving demonstration, forward motion, translation, and steering. The Ackermann model would provide an extension by capturing some pose constraints associated with front wheel steering. This is discussed in Chapter 6.

D.4.1 Camera Movement: Arc Ball

Camera movement can be calculated using a series of matrix multiplications and dot products. There are two major components to camera motion, these include orientation and translation. Each of these has been discussed, while setting a fixed orientation for a camera or projector. This section will discuss the implementation for providing an update to the current pose of the rendering camera, when provided with a state update to the current transformation.

The movement of the rendering camera in 3D space was controlled using a view matrix, $\mathbf{P} \in \mathbb{R}^{4\times4}$. This view matrix contained the orientation, $\mathbf{R} \in \mathbb{R}^{3\times3}$, and location, $\mathbf{t} \in \mathbb{R}^3$, of the camera. Similar to the definition of a camera pose in previous sections. When updated using user input, the update to the orientation was transformed via the matrix $\dot{\mathbf{R}} \in \mathbb{R}^{3\times3}$ or a quaternion $\dot{\mathbf{q}}$ composed of a vector

$[Q_x, Q_y, Q_z] \in \mathbb{C}^3$ and a scalar $Q_w \in \mathbb{R}$. An additional rotation was calculated by getting the inner product of the current transform and the additional rotation matrix. The change in the translation was computed with an addition of the final column of the current pose, $\mathbf{t} \in \mathbb{R}^3$, and the translational update, $\dot{\mathbf{t}} \in \mathbb{R}^3$. While both the rotation and the translation were required, these were calculated by getting the inner product of the current pose $\mathbf{P}$ and the motion update $\dot{\mathbf{P}}$.

$$\dot{\mathbf{P}} = \begin{bmatrix} \dot{\mathbf{R}} & \dot{\mathbf{t}} \\ 0 & 1 \end{bmatrix} \tag{D.3}$$

$$\hat{\mathbf{P}} = \dot{\mathbf{P}}\mathbf{P} \tag{D.4}$$

D.4.2 Vehicle Controls

Equations D.3 and D.4 aid in the description of discrete updates to the pose. However, the driver would expect that the state update vector would update in response to user controls. As opposed to the controls directly interfacing with the camera pose. For example, while a driver presses on the accelerator with zero steering angle it is expected that the forward motion of the vehicle increases in speed. The accelerator should not control the absolute position of the vehicle. For this reason, the forward motion speed was updated in response to the angle of the accelerator pedal. Steering wheel, accelerator and brake were polled from a thread outside of the game loop. The angular difference of the accelerator pedal from its neutral position was reflected as $\Delta\dot{\mathbf{t}}$, in Equation D.5. The rotational difference of the steering wheel from its neutral position directly transformed via the rotation update of the pose matrix on the yaw axis, $\dot{\mathbf{R}}$. This rotational update was done using an axis angle representation as this resulted in more readable code (others could include quaternion, or Euler angle). Equation D.5 was used in place of Equation D.3 while using these controls. The vehicle position was updated after every render cycle. The amount of time was recorded since the last update, this was called Δdt. The forward and angular motion was scaled according to Δdt. This ensured the vehicle would progress at the same speed with respect to time,

regardless of the frame rate for that render cycle.

$$\dot{\mathbf{P}} = \begin{bmatrix} \dot{\mathbf{R}} & \dot{\mathbf{t}} + \Delta\dot{\mathbf{t}} \\ 0 & 1 \end{bmatrix} \tag{D.5}$$

The rendering camera's pose update was computed with a single matrix product. The control inputs from the user come from a steering wheel to instruct the camera's rotation. The accelerator instructed the forward translational motion to the rendering camera. The viewing camera was facing forward on the negative z-axis; therefore, forward acceleration transformed the camera forward in this axis direction. This setup implemented an arc-ball camera where the centre of rotation was set about the camera centre. This was an essential first step in introducing vehicle motion in the simulator. It provides a template for future development into higher fidelity vehicle motion that could incorporate a variety of kinematic models (Ackermann, bicycle, swiss wheels, etc.) [149].

D.5 The System in Review

The following features have been included in the final driving simulator:

1. Vehicle tracking algorithms have been developed to enable the removal of vehicles from video and inpainting the region they previously occupied, as described in Chapter 5.

2. Rendering: Complete rendering procedures required for the driving simulation, including processing vertex and index buffers and using projective texture mapping with textures, have been developed.

3. View selection: Compared the drivers' render camera pose with the set of projector poses and selected the set of projectors that synthesize an optimal projection.

4. Planning and Loading: Compare the drivers' position against the 3D model that should be used for the current time. If the model is close enough to a new model, then a new 3D model is read and used to render. See Figure D.9 for an example of multiple models being rendered depending on

the rendering camera position. Vertex colours were used in this example to highlight that this is using the 3D reconstruction and not video frames. Note the driver sees mainly the projected view and the vertex colours shown in this figure are used off axis corrections to perspective texture mapping,

5. User Input: Poll the drivers' user input and introduce observable motion (steering, acceleration, and braking) of the rendering camera with respect to the 3D model.

Multi-threading would be necessary to deliver these processes concurrently at a high framerate. Currently, the driving simulator can select camera poses based on Euclidean distance for projective texture mapping, rendering the projections onto the 3D models, and reading the 3D models from the disk. Each is on separate threads. However, transferring the 3D model's vertex data into GPU memory must occur on the main thread, and this can lead to significant drops in the framerate. As a result, the current simulator operates over four meshes read into RAM and GPU memory at initialization. When the driver reaches the end of this mesh, they are transported to the initialization point again. The framerate is maintained at 60fps, and the driver is made aware that they have completed the circuit. User input is taken from a separate thread and updates the rendering camera at each render cycle using the motion models described above. This chapter demonstrates the feasibility of using real image data to reconstruct a 3D model of a road for driving simulator experiments.

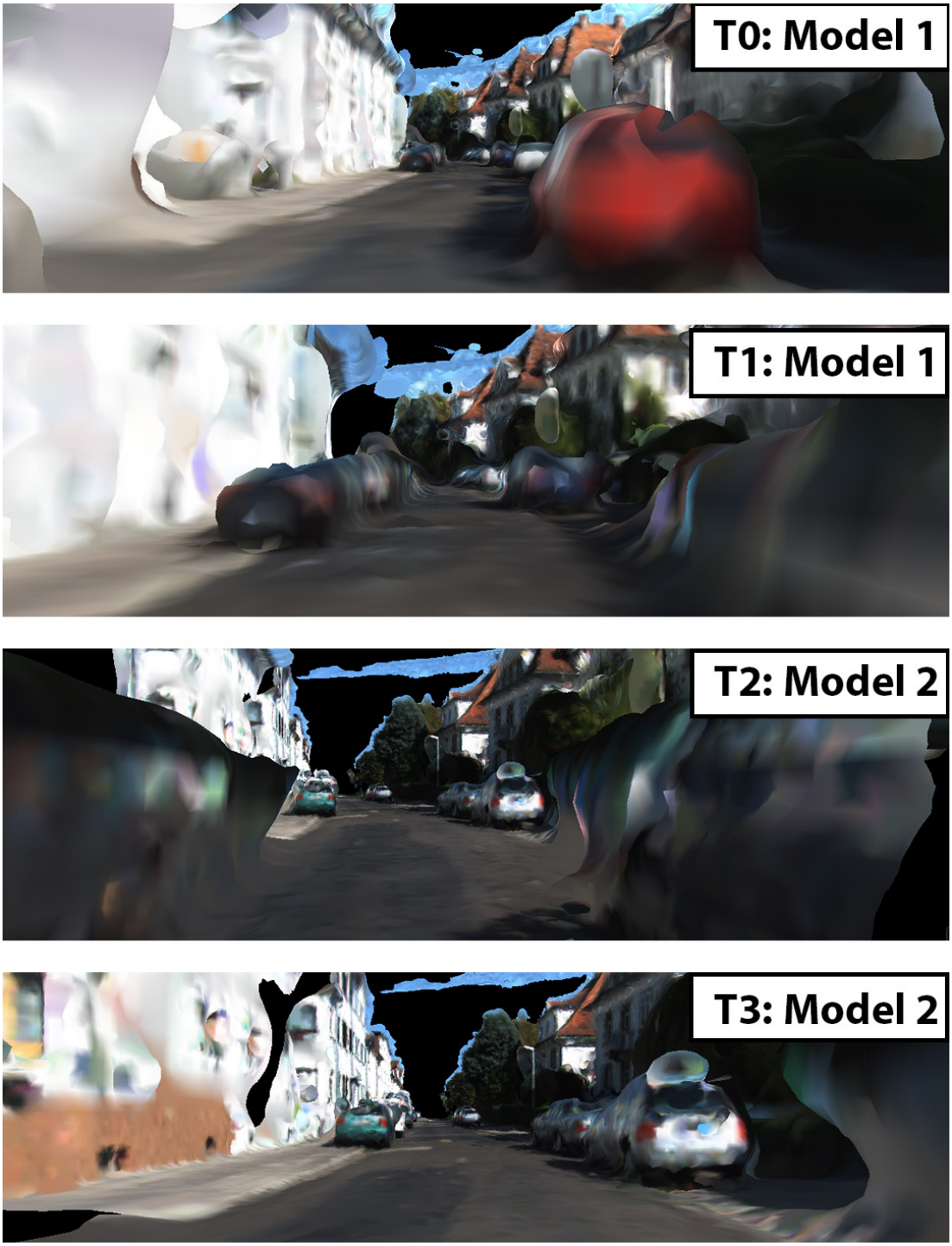


Figure D.9: Renderings of the 3D reconstructed models being loaded into the graphics pipeline based on the driving simulator camera position. This is shown for two frames in Model 1 (T0 and T1), and two frames in Model 2 (T2 and T3).

Bibliography

[1] Sameer Agarwal et al. "Building rome in a day". In: *Communications of the ACM* 54.10 (2011), pp. 105–112.

[2] Song Ho Ahn. *OpenGL Projection Matrix*. 2018. URL: `https://www.songho.ca/opengl/gl_projectionmatrix.html`.

[3] Alexander Amini et al. "VISTA 2.0: An Open, Data-driven Simulator for Multimodal Sensing and Policy Learning for Autonomous Vehicles". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022.

[4] Geerd Anders. "Pilot's attention allocation during approach and landing-Eye-and head-tracking research in an A 330 full flight simulator". In: *Focusing Attention on Aviation Safety* (2001).

[5] Christian Beder and Richard Steffen. "Determining an initial image pair for fixing the scale of a 3d reconstruction from an image sequence". In: *Joint Pattern Recognition Symposium*. Springer. 2006, pp. 657–666.

[6] Marcelo Bertalmio, Andrea L Bertozzi, and Guillermo Sapiro. "Navier-stokes, fluid dynamics, and image and video inpainting". In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*. Vol. 1. IEEE. 2001, pp. I–I.

[7] Sai Bi, Nima Khademi Kalantari, and Ravi Ramamoorthi. "Patch-based optimization for image-based texture mapping." In: *ACM Trans. Graph.* 36.4 (2017), pp. 106–1.

[8] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. "Evaluating the performance of structure from motion pipelines". In: *Journal of Imaging* 4.8 (2018), p. 98.

[9] Domenico D Bloisi et al. "Multi-modal background model initialization". In: *International conference on image analysis and processing*. Springer. 2015, pp. 485–492.

[10] Paul Bourke. *PLY - Polygon File Format Also known as the Stanford Triangle Format*. URL: `http://paulbourke.net/dataformats/ply/`.

[11] Thierry Bouwmans, Lucia Maddalena, and Alfredo Petrosino. "Scene background initialization: A taxonomy". In: *Pattern Recognition Letters* 96 (2017), pp. 3–11.

[12] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[13] Michael Brogan et al. "Automatic Generation and Population of a Graphics-Based Driving Simulator: Use of Mobile Mapping Data for Behavioral Testing of Drivers". In: *Transportation Research Record* 2434.1 (2014), pp. 95–102. DOI: `10.3141/2434-12`. eprint: `https://doi.org/10.3141/2434-12`. URL: `https://doi.org/10.3141/2434-12`.

[14] Michael Brogan et al. "Steering in Video-Based Driving Simulation with Stereo Depth Maps: Dynamic Perspective Corrections". In: *Transportation Research Record* 2518.1 (2015), pp. 104–112.

[15] Brent Burley. "Extending the Disney BRDF to a BSDF with integrated subsurface scattering". In: *SIGGRAPH Course: Physically Based Shading in Theory and Practice. ACM, New York, NY* 19 (2015).

[16] Brent Burley and Walt Disney Animation Studios. "Physically-based shading at disney". In: *ACM SIGGRAPH*. Vol. 2012. vol. 2012. 2012, pp. 1–7.

[17] Brent Burley and Walt Disney Animation Studios. "Physically-based shading at disney". In: *ACM SIGGRAPH*. Vol. 2012. vol. 2012. 2012, pp. 1–7.

[18] V Cavallo, E Dumont, and G Gallée. "Experimental validation of extended fog simulation techniques". In: *Etudes et recherches des Laboratoires des ponts et chaussées. Série Chaussées* (2007), pp. 89–100.

[19] Dan Cernea. "OpenMVS: Multi-View Stereo Reconstruction Library". 2020. URL: `https://cdcseacave.github.io/openMVS`.

[20] John Challen. "Reality Bytes: Toyota's Ultimate Analysis Tool". In: *Traffic Technology International* (2008).

[21] Ya-Liang Chang et al. "Free-form Video Inpainting with 3D Gated Convolution and Temporal PatchGAN". In: *In Proceedings of the International Conference on Computer Vision (ICCV)* (2019).

[22] Peter Chapman, Geoffrey Underwood, and Katharine Roberts. "Visual search patterns in trained and untrained novice drivers". In: *Transportation Research Part F: Traffic Psychology and Behaviour* 5.2 (2002), pp. 157–167.

[23] Chih-Fan Chen, Mark Bolas, and Evan Suma Rosenberg. "Rapid creation of photorealistic virtual reality content with consumer depth cameras". In: *2017 IEEE Virtual Reality (VR)*. IEEE. 2017, pp. 473–474.

[24] Chih-Fan Chen, Mark Bolas, and Evan Suma Rosenberg. "View-dependent virtual reality content from RGB-D images". In: *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2017, pp. 2931–2935.

[25] Chih-Fan Chen, Mark Bolas, and Evan Suma. "Real-time 3D rendering using depth-based geometry reconstruction and view-dependent texture mapping". In: *ACM SIGGRAPH 2016 Posters*. 2016, pp. 1–2.

[26] Chih-Fan Chen and Evan Suma Rosenberg. "Automatic Generation of Dynamically Relightable Virtual Objects with Consumer-Grade Depth Cameras". In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. 2019, pp. 1295–1296. DOI: `10.1109/VR.2019.8798037`.

[27] Chih-Fan Chen and Evan Suma Rosenberg. "Dynamic omnidirectional texture synthesis for photorealistic virtual content creation". In: *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. IEEE. 2018, pp. 85–90.

[28] Chih-Fan Chen and Evan Suma Rosenberg. "Capture to Rendering Pipeline for Generating Dynamically Relightable Virtual Objects with Handheld RGB-D Cameras". In: *Proceedings of the 26th ACM Symposium on Virtual Reality Software and Technology*. 2020, pp. 1–11.

[29] Yang Chen and Gérard G Medioni. "Object modeling by registration of multiple range images." In: *Image Vis. Comput.* 10.3 (1992), pp. 145–155.

[30] Paolo Cignoni et al. "Meshlab: an open-source mesh processing tool." In: *Eurographics Italian chapter conference*. 2008, pp. 129–136.

[31] William Clifford, Catherine Deegan, and Charles Markham. "High speed reconstruction of a scene implemented through projective texture mapping". In: *Irish Machine Vision and Image Processing Conference Proceedings*. The Irish Pattern Recognition & Classification Society. 2017, pp. 171–177.

[32] William Clifford and Charles Markham. "Ghost Towns: Semantically Labelled Object Removal From Video". In: *Irish Machine Vision and Image Processing Conference Proceedings*. The Irish Pattern Recognition & Classification Society. 2019, pp. 124–131.

[33] William Clifford and Charles Markham. "Projective Texture Mapping on Reconstructed Scenes". In: *Irish Machine Vision and Image Processing Conference Proceedings*. The Irish Pattern Recognition & Classification Society. 2020, pp. 101–104.

[34] William Clifford, Charles Markham, and Catherine Deegan. "Smart Detection of Driver Distraction Events". In: *European Conference of Eye Movement*. European Group of Scientists active in Eye Movement Research. 2017.

[35] William Clifford et al. "Method to assess driver behaviour following distractions external to the vehicle". In: *7th International Conference on Driver Distraction and Inattention*. SAFER Vehicle, Traffic Safety Centre at Chalmers in Sweden, the Université Gustave Eiffel, France, and the University of New South Wales, Australia. 2021, pp. 48–50.

[36] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2022.

[37] Mechanical Simulation Corporation. *Carsim mechanical simulation.* Version 2021.1. Aug. 31, 2021. URL: `https://www.carsim.com/products/carsim/index.php`.

[38] Thomas Scott Crow. "Evolution of the graphical processing unit". In: *A professional paper submitted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science, University of Nevada, Reno* (2004).

[39] Brian Curless and Marc Levoy. "A volumetric method for building complex models from range images". In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 1996, pp. 303–312.

[40] Angela Dai et al. "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration". In: *ACM Transactions on Graphics 2017 (TOG)* (2017).

[41] Davison. "Real-time simultaneous localisation and mapping with a single camera". In: *Proceedings Ninth IEEE International Conference on Computer Vision*. 2003, 1403–1410 vol.2. DOI: `10.1109/ICCV.2003.1238654`.

[42] Andrew J Davison et al. "MonoSLAM: Real-time single camera SLAM". In: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), pp. 1052–1067.

[43] Paul Debevec, Yizhou Yu, and George Borshukov. "Efficient view-dependent image-based rendering with projective texture-mapping". In: *Eurographics Workshop on Rendering Techniques*. Springer. 1998, pp. 105–116.

[44] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[45] Konstantinos G Derpanis. "The harris corner detector". In: *York University* 2 (2004).

[46] Edmund Donges. "A two-level model of driver steering behavior". In: *Human factors* 20.6 (1978), pp. 691–707.

[47] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". In: *Conference on robot learning*. PMLR. 2017, pp. 1–16.

[48] George Downs. *Why driverless cars are taking 'hyper-realistic' virtual driving lessons*. 2022. URL: https://www.wsj.com/video/series/george-downs/why-driverless-cars-are-taking-hyper-realistic-virtual-driving-lessons/394A6E9F-CAA7-4B49-8C07-7EBE960355E9.

[49] Johannes Drosdol and Ferdinand Panik. "The Daimler-Benz driving simulator a tool for vehicle development". In: *SAE Transactions* (1985), pp. 981–997.

[50] Andrew T Duchowski. "Eye tracking methodology". In: *Theory and practice* 328.614 (2007), pp. 2–3.

[51] "Editorial Board". In: *Transportation Research Part F: Traffic Psychology and Behaviour* 77 (2021), p. ii. DOI: 10.1016/s1369-8478(21)00045-0.

[52] Epic Games. *Unreal Engine*. Version 4.22.1. Apr. 25, 2019. URL: https://www.unrealengine.com.

[53] Cass Everitt. "Projective texture mapping". In: *White paper, NVidia Corporation* 4.3 (2001).

[54] Martin A Fischler and Robert C Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[55] Donald L Fisher et al. *Handbook of driving simulation for engineering, medicine, and psychology*. CRC press, 2011.

[56] Bernard H Fox. "Engineering and psychological uses of a driving simulator". In: *Highway Research Board Bulletin* 261 (1960), pp. 14–37.

[57] Jan-Michael Frahm et al. "Building rome on a cloudless day". In: *European conference on computer vision*. Springer. 2010, pp. 368–381.

[58] Simon Fuhrmann, Fabian Langguth, and Michael Goesele. "MVE-A Multi-View Reconstruction Environment." In: *GCH*. Citeseer. 2014, pp. 11–18.

[59] Yasutaka Furukawa and Carlos Hernández. "Multi-view stereo: A tutorial". In: *Foundations and Trends® in Computer Graphics and Vision* 9.1-2 (2015), pp. 1–148.

[60] Yasutaka Furukawa and Jean Ponce. "Accurate, Dense, and Robust Multi-View Stereopsis". In: *IEEE Trans. on Pattern Analysis and Machine Intelligence* 32.8 (2010), pp. 1362–1376.

[61] Clément Galko, Romain Rossi, and Xavier Savatier. "Vehicle-Hardware-In-The-Loop system for ADAS prototyping and validation". In: *2014 International conference on embedded computer systems: Architectures, Modeling, and Simulation (SAMOS XIV)*. IEEE. 2014, pp. 329–334.

[62] Dorian Gálvez-López and J. D. Tardós. "Bags of Binary Words for Fast Place Recognition in Image Sequences". In: *IEEE Transactions on Robotics* 28.5 (2012), pp. 1188–1197. ISSN: 1552-3098. DOI: 10.1109/TRO.2012.2197158.

[63] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[64] *Application of Computer Generated Imagery to Driver Training*. Vol. 8. North Carolina Symposium on Highway Safety. Chapel Hill, NC: University of North Carolina Highway Safety Research Center, 1973.

[65] Andrew S Glassner. *An introduction to ray tracing*. Elsevier, 1989.

[66] Hans Godthelp, Paul Milgram, and Gerard J Blaauw. "The development of a time-related measure to describe driving strategy". In: *Human factors* 26.3 (1984), pp. 257–268.

[67] Google. *Google Maps directions from Kill East Co Kildare to L2007, Ballyhays, Co. Kildare*. Retrieved July 11th 2023. n.d. URL: https://goo.gl/maps/jiMidTpG7jxWFGheA.

[68] A. Handa et al. "A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM". In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. Hong Kong, China, 2014.

[69] Joanne L Harbluk et al. "An on-road assessment of cognitive distraction: Impacts on drivers' visual behavior and braking performance". In: *Accident Analysis & Prevention* 39.2 (2007), pp. 372–379.

[70] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[71] Richard I Hartley and Peter Sturm. "Triangulation". In: *Computer vision and image understanding* 68.2 (1997), pp. 146–157.

[72] Mary M Hayhoe, David G Bensinger, and Dana H Ballard. "Task constraints in visual working memory". In: *Vision research* 38.1 (1998), pp. 125–137.

[73] Jibo He et al. "Mind wandering behind the wheel: Performance and oculomotor correlates". In: *Human factors* 53.1 (2011), pp. 13–21.

[74] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[75] Jia-Bin Huang et al. "Temporally coherent completion of dynamic video". In: *ACM Transactions on Graphics (TOG)* 35.6 (2016), pp. 1–11.

[76] Jingwei Huang et al. "3Dlite: towards commodity 3D scanning for content creation." In: *ACM Trans. Graph.* 36.6 (2017), pp. 203–1.

[77] Charles H Hutchinson. *Automobile driving simulator feasibility study*. Cornell Aeronautical Laboratory, Incorporated of Cornell University, 1958.

[78] iRacing.com Motorsport Simulations, iRacing. *iRacing*. URL: `https://www.iracing.com/`.

[79] Shahram Izadi et al. "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera". In: *Proceedings of the 24th annual ACM symposium on User interface software and technology*. 2011, pp. 559–568.

[80] J Johnsson and R Matos. "Accuracy and precision test method for remote eye trackers". In: *Tobii Technology* (2011).

[81] David Kaneswaran. "Assessment of the Effects of Road Geometry on Irish Accident Rates and Driving Behaviour". PhD thesis. Maynooth University 2014., 2014.

[82] Zhizhong Kang and Zhen Li. "Primitive fitting based on the efficient multiBaySAC algorithm". In: *PloS one* 10.3 (2015), e0117341.

[83] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. "Poisson surface reconstruction". In: *Proceedings of the fourth Eurographics symposium on Geometry processing*. Vol. 7. 2006.

[84] Sheila G Klauer et al. "The impact of driver inattention on near-crash/crash risk: An analysis using the 100-car naturalistic driving study data". In: (2006).

[85] Georg Klein and David Murray. "Parallel tracking and mapping for small AR workspaces". In: *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE. 2007, pp. 225–234.

[86] C Kothe, D Medine, and M Grivich. "Lab Streaming Layer (2014)". In: *URL: https://github. com/sccn/labstreaminglayer (visited on 02/01/2019)* (2018).

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[88] Rainer Kümmerle et al. "g 2 o: A general framework for graph optimization". In: *2011 IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 3607–3613.

[89] Iro Laina et al. "Deeper depth prediction with fully convolutional residual networks". In: *2016 Fourth international conference on 3D vision (3DV)*. IEEE. 2016, pp. 239–248.

[90] Michael F Land and David N Lee. "Where we look when we steer". In: *Nature* 369.6483 (1994), pp. 742–744.

[91] Benjamin Laugraud, Sébastien Piérard, and Marc Van Droogenbroeck. "LaBGen: A method based on motion detection for generating the background of a scene". In: *Pattern Recognition Letters* 96 (2017), pp. 12–21.

[92] Benjamin Laugraud, Sébastien Piérard, and Marc Van Droogenbroeck. "LaBGen-P: A pixel-level stationary background generation method based on LaBGen". In: *2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE. 2016, pp. 107–113.

[93] R John Leigh and David S Zee. *The neurology of eye movements*. OUP USA, 2015.

[94] Yulan Liang, Michelle L Reyes, and John D Lee. "Real-time detection of driver cognitive distraction using support vector machines". In: *IEEE transactions on intelligent transportation systems* 8.2 (2007), pp. 340–350.

[95] Miao Liao et al. "DVI: Depth Guided Video Inpainting for Autonomous Driving". In: *European Conference on Computer Vision*. Springer. 2020, pp. 1–17.

[96] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.

[97] Guilin Liu et al. "Image Inpainting for Irregular Holes Using Partial Convolutions". In: *The European Conference on Computer Vision (ECCV)*. 2018.

[98] H Christopher Longuet-Higgins. "A computer algorithm for reconstructing a scene from two projections". In: *Nature* 293.5828 (1981), pp. 133–135.

[99] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[100] Bruce David Lucas. *Generalized image matching by the method of differences*. Carnegie Mellon University, 1985.

[101] *Lumen global illumination and reflections*. URL: `https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/`.

[102] Yi Ma et al. *An invitation to 3-d vision: from images to geometric models*. Vol. 26. Springer Science & Business Media, 2012.

[103] Andrew K Mackenzie and Julie M Harris. "A link between attentional function, effective eye movements, and driving ability." In: *Journal of experimental psychology: human perception and performance* 43.2 (2017), p. 381.

[104] Lucia Maddalena and Alfredo Petrosino. "The SOBS algorithm: What are the limits?" In: *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE. 2012, pp. 21–26.

[105] Ricardo Martin-Brualla et al. "Nerf in the wild: Neural radiance fields for unconstrained photo collections". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 7210–7219.

[106] John McCormac et al. "Semanticfusion: Dense 3d semantic mapping with convolutional neural networks". In: *2017 IEEE International Conference on Robotics and automation (ICRA)*. IEEE. 2017, pp. 4628–4635.

[107] A James McKnight and A Scott McKnight. "The effect of cellular phone use upon driver attention". In: *Accident Analysis & Prevention* 25.3 (1993), pp. 259–265.

[108] Natasha Merat et al. "The "Out-of-the-Loop" concept in automated driving: proposed definition, measures and implications". In: *Cognition, Technology & Work* 21.1 (2019), pp. 87–98.

[109] Coleman Merenda et al. "Effects of vehicle simulation visual fidelity on assessing driver performance and behavior". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2019, pp. 1679–1686.

[110] Despina Michael et al. "Impact of immersion and realism in driving simulator studies". In: *International Journal of Interdisciplinary Telecommunications and Networking (IJITN)* 6.1 (2014), pp. 10–25.

[111] Ben Mildenhall et al. "Nerf: Representing scenes as neural radiance fields for view synthesis". In: *European Conference on Computer Vision*. Springer. 2020, pp. 405–421.

[112] Lionel Moisan, Pierre Moulon, and Pascal Monasse. "Automatic homographic registration of a pair of images, with a contrario elimination of outliers". In: *Image Processing On Line* 2 (2012), pp. 56–73.

[113] Pierre Moulon and Pascal Monasse. "Unordered feature tracking made fast and easy". In: *CVMP 2012*. 2012, p. 1.

[114] Pierre Moulon, Pascal Monasse, and Renaud Marlet. "Adaptive Structure from Motion with a Contrario Model Estimation". In: *Proceedings of the Asian Computer Vision Conference (ACCV 2012)*. Springer Berlin Heidelberg, 2012, pp. 257–270. DOI: 10.1007/978-3-642-37447-0_20.

[115] Pierre Moulon et al. "Openmvg: Open multiple view geometry". In: *International Workshop on Reproducible Research in Pattern Recognition*. Springer. 2016, pp. 60–74.

[116] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system". In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[117] Anneli Olsen. "The Tobii I-VT fixation filter". In: *Tobii Technology* (2012), pp. 1–21.

[118] Shintaro Ono et al. "A photo-realistic driving simulation system for mixed-reality traffic experiment space". In: *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.* IEEE. 2005, pp. 747–752.

[119] *Parallel domains website*. 2023. URL: https://paralleldomain.com/.

[120] Evangelos Paschalidis, Charisma F Choudhury, and Stephane Hess. "Combining driving simulator and physiological sensor data in a latent variable model to incorporate the effect of stress in car-following behaviour". In: *Analytic methods in accident research* 22 (2019), p. 100089.

[121] Deepak Pathak et al. "Context encoders: Feature learning by inpainting". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 2536–2544.

[122] *Performing a calibration and validation in pro lab*. 2019. URL: https://www.tobiipro.com/learn-and-support/learn/steps-in-an-eye-tracking-study/run/performing-a-calibration-and-validation-in-pro-lab/.

[123] Josselin Petit and Roland Brémond. "A high dynamic range rendering pipeline for interactive applications". In: *The Visual Computer* 26.6 (2010), pp. 533–542.

[124] Bui Tuong Phong. "Illumination for computer generated pictures". In: *Communications of the ACM* 18.6 (1975), pp. 311–317.

[125] Jordi Pont-Tuset et al. "The 2017 davis challenge on video object segmentation". In: *arXiv preprint arXiv:1704.00675* (2017).

[126] *Proceedings of the Sixth International Conference on Driver Distraction and Inattention, Gothenburg, Sweden, October 15-17, 2018*. Gothenburg, Sweden: Sweden MEETX AB, 2018.

[127] Miguel A Recarte and Luis M Nunes. "Effects of verbal and spatial-imagery tasks on eye fixations while driving." In: *Journal of experimental psychology: Applied* 6.1 (2000), p. 31.

[128] Miguel A Recarte and Luis M Nunes. "Mental workload while driving: effects on visual search, discrimination, and decision making." In: *Journal of experimental psychology: Applied* 9.2 (2003), p. 119.

[129] Michael A Regan, Charlene Hallett, and Craig P Gordon. "Driver distraction and driver inattention: Definition, relationship and taxonomy". In: *Accident Analysis & Prevention* 43.5 (2011), pp. 1771–1781.

[130] SR Research. *EyeLink 1000 User's Manual, Version 1.5. 2*. 2010.

[131] GL RICHARD et al. "Compensation for transport delays produced by computer image generation systems[Final Report, Nov. 1976- May 1978]". In: (1978).

[132] Gernot Riegler and Vladlen Koltun. "Free view synthesis". In: *European Conference on Computer Vision*. Springer. 2020, pp. 623–640.

[133] Gernot Riegler and Vladlen Koltun. "Stable view synthesis". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 12216–12225.

[134] Bernardino Romera-Paredes and Philip Hilaire Sean Torr. "Recurrent instance segmentation". In: *European conference on computer vision*. Springer. 2016, pp. 312–329.

[135] Adrian Rosebrock. *Simple object tracking with OpenCV*. 2020. URL: `https://www.pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/`.

[136] Susana Ruano and Aljosa Smolic. "A Benchmark for 3D Reconstruction from Aerial Imagery in an Urban Environment." In: *VISIGRAPP (5: VISAPP)*. 2021, pp. 732–741.

[137] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF." In: *ICCV*. Vol. 11. 1. Citeseer. 2011, p. 2.

[138] Manolis Savva et al. "Habitat: A Platform for Embodied AI Research". In: *CoRR* abs/1904.01201 (2019). arXiv: `1904.01201`. URL: `http://arxiv.org/abs/1904.01201`.

[139] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[140] Johannes Lutz Schönberger et al. "Pixelwise View Selection for Unstructured Multi-View Stereo". In: *European Conference on Computer Vision (ECCV)*. 2016.

[141] Thomas Schulze et al. "Open asset import library (assimp)". In: *Computer Software, URL: https://github. com/assimp/assimp* (2021).

[142] Chris Schwarz, Tad Gates, and Yiannis Papelis. "Motion characteristics of the national advanced driving simulator". In: *Proceedings of the Driving Simulation Conference*. 2003.

[143] SCS Software. *Euro Truck Simulator 2*. URL: `https://eurotrucksimulator2.com/`.

[144] Mark Segal et al. "Fast shadows and lighting effects using texture mapping". In: *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. 1992, pp. 249–252.

[145] Graham Sellers, Richard S Wright Jr, and Nicholas Haemel. *OpenGL superBible: comprehensive tutorial and reference*. Addison-Wesley, 2013.

[146] Charlotte Sennersten. "Gameplay (3D Game engine+ Ray tracing= Visual attention through eye tracking)". PhD thesis. Blekinge Institute of Technology, 2008.

[147] Shital Shah et al. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles". In: *Field and Service Robotics*. 2017. eprint: arXiv:1705.05065. URL: https://arxiv.org/abs/1705.05065.

[148] WL Shebilske and DF Fisher. *Understanding extended discourse through the eyes: How and why*. 1983.

[149] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[150] Kyle Simek. *Calibrated Cameras in OpenGL without glFrustum*. 2013. URL: http://ksimek.github.io/2013/06/03/calibrated_cameras_in_opengl.

[151] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[152] Noah Snavely, Steven M Seitz, and Richard Szeliski. "Photo tourism: exploring photo collections in 3D". In: *ACM Siggraph 2006 Papers*. 2006, pp. 835–846.

[153] Chloe Snyder et al. "Approximating 3D poly mesh virtual objects with primitive shape colliders". In: (2019).

[154] Open Source. *OpenSceneGraph*. Version 3.6.0. Aug. 31, 2021. URL: http://www.openscenegraph.org/.

[155] Elisavet Konstantina Stathopoulou and Fabio Remondino. "Open-source image-based 3D reconstruction pipelines: Review, comparison and evaluation". In: *6th International Workshop LowCost 3D–Sensors, Algorithms, Applications*. 2019, pp. 331–338.

[156] Hauke Strasdat, J Montiel, and Andrew J Davison. "Scale drift-aware large scale monocular SLAM". In: *Robotics: Science and Systems VI* 2.3 (2010), p. 7.

[157] David L Strayer, Frank A Drews, and William A Johnston. "Cell phone-induced failures of visual attention during simulated driving." In: *Journal of experimental psychology: Applied* 9.1 (2003), p. 23.

[158] Jane C Stutts et al. "The role of driver distraction in traffic crashes". In: (2001).

[159] Satoshi Suzuki et al. "Topological structural analysis of digitized binary images by border following". In: *Computer vision, graphics, and image processing* 30.1 (1985), pp. 32–46.

[160] Chris Sweeney. *Theia multiview geometry library: Tutorial & reference*. 2015.

[161] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[162] Amy Tabb. *Converting OpenCV cameras to OpenGL cameras*. 2019. URL: `https://amytabb.com/ts/2019_06_28`.

[163] Takuji Takahashi et al. "Arbitrary view position and direction rendering for large-scale scenes". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*. Vol. 2. IEEE. 2000, pp. 296–303.

[164] Unity Technologies. *Unity Engine Manual*. accessed: 26/09/2019. 2019. URL: `docs.unity3d.com`.

[165] Alexandru Telea. "An image inpainting technique based on the fast marching method". In: *Journal of graphics tools* 9.1 (2004), pp. 23–34.

[166] Sebastian Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[167] Transport Infrastructure Ireland T.I.I. *Reports and Accounts*. 2020. URL: `https://www.tii.ie/tii-library/reports-accounts/`.

[168] Jan EB Törnros and Anne K Bolling. "Mobile phone use—effects of handheld and handsfree phones on driving performance". In: *Accident Analysis & Prevention* 37.5 (2005), pp. 902–909.

[169] P.H.S. Torr. "An assessment of information criteria for motion model selection". In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 1997, pp. 47–52. DOI: 10.1109/CVPR.1997.609296.

[170] John R Treat. "A study of precrash factors involved in traffic accidents." In: *HSRI Research review* (1980).

[171] Bill Triggs et al. "Bundle adjustment—a modern synthesis". In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[172] Shubham Tulsiani et al. "Learning shape abstractions by assembling volumetric primitives". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017, pp. 2635–2643.

[173] Turn 10 Studios, Playground Games. *Forza Horizon 4*. URL: https://forzamotorsport.net/en-US/games/fh4.

[174] Luiz Velho and Jonas Sossai Jr. "Projective texture atlas construction for 3D photography". In: *The Visual Computer* 23.9-11 (2007), pp. 621–629.

[175] Trent W Victor, Joanne L Harbluk, and Johan A Engström. "Sensitivity of eye-movement measures to in-vehicle task difficulty". In: *Transportation Research Part F: Traffic Psychology and Behaviour* 8.2 (2005), pp. 167–190.

[176] *Visual Studio performance tips and tricks*. 2022. URL: https://docs.microsoft.com/en-us/visualstudio/ide/visual-studio-performance-tips-and-tricks?view=vs-2022.

[177] Huy V Vo, Ngoc QK Duong, and Patrick Pérez. "Structural inpainting". In: *Proceedings of the 26th ACM international conference on Multimedia*. 2018, pp. 1948–1956.

[178] Joey de Vries. *Welcome to OpenGL*. URL: https://learnopengl.com/.

[179] Michael Waechter, Nils Moehrle, and Michael Goesele. "Let there be color! Large-scale texturing of 3D reconstructions". In: *European conference on computer vision*. Springer. 2014, pp. 836–850.

[180] Yongxiang Wang et al. "Examination of driver visual and cognitive responses to billboard elicited passive distraction using eye-fixation related potential". In: *Sensors* 21.4 (2021), p. 1471.

[181] Thomas Weber and Christian Plattfaut. *Virtual night drive*. Tech. rep. SAE Technical Paper, 2001.

[182] David H Weir and Charles K Wojcik. *SIMULATOR STUDIES OF THE DRIVER'S DYNAMIC RESPONSE IN STEERING CONTROL TASKS-WITH DISCUSSION AND CLOSURE*. Tech. rep. 1971.

[183] *What happens during the eye tracker calibration?*. 2022. URL: `https://www.tobiipro.com/learn-and-support/learn/eye-tracking-essentials/what-happens-during-the-eye-tracker-calibration`.

[184] Thomas Whelan et al. "ElasticFusion: Dense SLAM without a pose graph". In: Robotics: Science and Systems. 2015.

[185] Thomas Whelan et al. "ElasticFusion: Real-time dense SLAM and light source estimation". In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1697–1716.

[186] Thomas Whelan et al. "Kintinuous: Spatially extended kinectfusion". In: (2012).

[187] OpenGL Wiki. *OpenGL and multithreading*. Online; straffsed 27-September-2022. 2018. URL: `https://www.khronos.org/opengl/wiki/OpenGL_and_multithreading`.

[188] Changchang Wu. "Towards linear-time incremental structure from motion". In: *2013 International Conference on 3D Vision-3DV 2013*. IEEE. 2013, pp. 127–134.

[189] Changchang Wu et al. "VisualSFM: A visual structure from motion system". In: (2011).

[190] Fei Xia et al. *Gibson Env: Real-World Perception for Embodied Agents*. 2018. arXiv: `1808.10654 [cs.AI]`.

[191] Ning Xu et al. "Youtube-vos: A large-scale video object segmentation benchmark". In: *arXiv preprint arXiv:1809.03327* (2018).

[192] Rui Xu et al. "Deep flow-guided video inpainting". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 3723–3732.

[193] Alfred L. Yarbus. "Eye Movements During Perception of Complex Objects". In: *Eye Movements and Vision*. Boston, MA: Springer US, 1967, pp. 171–211. ISBN: 978-1-4899-5379-7. DOI: 10.1007/978-1-4899-5379-7_8. URL: https://doi.org/10.1007/978-1-4899-5379-7_8.

[194] Kristie Young, Michael Regan, and M Hammer. "Driver distraction: A review of the literature". In: *Distracted driving* 2007 (2007), pp. 379–405.

[195] Kai Zhang et al. "Nerf++: Analyzing and improving neural radiance fields". In: *arXiv preprint arXiv:2010.07492* (2020).

[196] Zhengyou Zhang. "A flexible new technique for camera calibration". In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), pp. 1330–1334.

[197] Hengshuang Zhao et al. "Pyramid scene parsing network". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2881–2890.

[198] Bolei Zhou et al. "Scene Parsing through ADE20K Dataset". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2017.

[199] Bolei Zhou et al. "Semantic understanding of scenes through the ade20k dataset". In: *International Journal on Computer Vision* (2018).

[200] Qian-Yi Zhou and Vladlen Koltun. "Color map optimization for 3D reconstruction with consumer depth cameras". In: *ACM Transactions on Graphics (TOG)* 33.4 (2014), pp. 1–10.

[201] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. *Open3D: A Modern Library for 3D Data Processing*. cite arxiv:1801.09847Comment: http://www.open3d.org. 2018. URL: http://arxiv.org/abs/1801.09847.

[202] SM Zolanvari et al. "DublinCity: Annotated LiDAR point cloud and its applications". In: *arXiv preprint arXiv:1909.03613* (2019).

[203] Chuhang Zou et al. "3d-prnn: Generating shape primitives with recurrent neural networks". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 900–909.