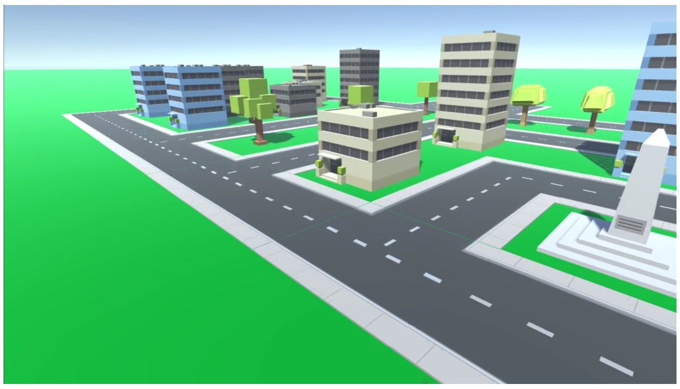


SmartMob-VR: Virtual Reality platform for the development and validation of intelligent mobility systems

André Ponce Álvares Vieira

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. Alfredo Manuel Dos Santos Ferreira Júnior
Prof. Teresa Maria Sá Ferreira Vazão Vasques

December 2019

Acknowledgments

My university life has been a long journey and I could not have done it without all the people that were with me and that I've met throughout the years. My family, my friends, teachers and colleagues. I would like to thank them all for allowing and helping me to achieve what I did. My family that was always there to back me up from the start to end. The childhood friends and high school friends that made the transition from high school life to university life easier and friends I made along the way. Not only university friends that gave extra motivation and laughs daily but foreign friends as well. Especially, the foreign friend, for always being there for anything. Lastly, but definitely not least, I want to thank a very special person to me, my girlfriend, for always being there giving extra support, having patience, motivating me in a numerous amount of ways, bringing joy and love throughout the days we've been together.

Thank you all.

Resumo

O SmartMob é um projeto de um curso do campus Taguspark do IST. Este projeto permite que os alunos estudem a comunicação por rede com carros que podem ser controlados com scripts num Raspberry pi. Estes carros são físicos tendo baterias que são drenadas muito rapidamente. Os carros são usados com uma quantidade limitada de estradas físicas semelhantes a carpetes para modelar cidades. O SmartMobVR coloca este projeto em Realidade Virtual, motivando os alunos e mitigando alguns problemas. O SmartMobVR tem duas componentes principais: modelação e visualização. Modelação é o programa que permite aos alunos desenvolver uma cidade. Há uma grade bidimensional onde se podem colocar estradas, prédios, árvores, estátuas, semáforos e lâmpadas para modelar a cidade desejada e depois exportar para um arquivo XML. A visualização, depois, importa o arquivo XML e constrói a cidade implementada anteriormente em três dimensões que pode ser vista em Realidade Virtual. Os carros reais ainda podem comunicar-se entre si e, também, com a visualização para que o utilizador possa ver qual movimento que os carros estão a fazer. Os utilizadores podem navegar pela cidade e selecionar os carros que desejam. Tendo um mundo dinâmico e objetos dinâmicos, estudamos técnicas de seleção para ver qual funciona melhor para o nosso projeto.

Palavras-chave: Realidade Virtual, SmartMob, Técnicas de seleção, Modelação, Dinâmico

Abstract

SmartMob is a project from a IST's Taguspark campus's course. This project allows students to study network communication with cars that can be controlled with scripts on a Raspberry pi. These are physical cars with batteries that are drained very quickly. The cars are used with a limited amount of physical carpet-like roads to model cities. SmartMobVR puts this project in Virtual Reality, motivating students and mitigating some problems. SmartMob VR has two main components: modelling and visualisation. Modelling is the program that allows the students to develop a city. There is a two dimensional grid where they can place roads, buildings, trees, statues, traffic lights and lamps to model the desired city to then export into an XML file. The visualisation, then, imports the XML file and builds the previously implemented city in three dimensions that can be seen in Virtual Reality. The real cars can still communicate with each other and communicate with the visualisation so the user can see what movement the cars are doing. Users can navigate through the city and select the cars they want to. Having a dynamic world and dynamic objects we study selection techniques to see which one works best for our project.

Keywords: Virtual Reality, SmartMob, Selection Techniques, Modelling, Dynamic

Contents

List of Tables

List of Figures

Chapter 1

Introduction

SmartMob is a project developed in academic environment that allows students to study and develop intelligent mobility systems with infra-structure elements and vehicles that work in a cooperative and connected way. The project consists in teams of students proposing solutions for challenges such as road safety, traffic management or entertainment. These solutions, developed in practical classes, will be later applied in the real world.

Currently, the modelling of cities is done by creating road maps using flat rectangular objects, similar to carpets, and assembling straight roads or intersections, by placing them consecutively. After modelling the city's street network, vehicles and infrastructure elements that function as smart objects can be introduced into the city. Students can control vehicles remotely through a computer, by either applying single real-time instructions or by running an algorithm that controls their movement. Once the modelling of the city's road map is completed, and the algorithms that control the intelligent objects are implemented, it's possible to test the overall solution, and evaluate its behaviour and effectiveness in solving the proposed problem.

However, solution validation is always limited, given the impossibility to represent, in a miniature world, the complex scenarios that occur in a real environment.

Virtual Reality opens great technological doors, allowing users to immerse themselves in new worlds and have a level of interaction with virtual environments never before achieved. VR creates new opportunities for broader and more complex tests and experiments, not only allowing greater environment control and new interaction possibilities, but also presenting less limitations, like the ones imposed in the real world.

We want users to be able to individually select any object in our virtual scenes, in order to get specific information about it. Interaction and selection of objects are very important when it comes to virtual worlds; however, choosing how to accomplish them is also one of its greatest challenges. So much so that it has being studied over the years with no definitive answer on which technique is overall better, when compared to others. Most papers study the usage of selection techniques in static environments with static objects, but in this study we chose to analyse different selection techniques and evaluate which one demonstrates a better performance when executing a given set of tasks.

1.1 Motivation

Tests and experiments carried out in the real world allow us to obtain results that more accurately demonstrate how a test subject will behave after deployment, especially when the testing environment is the same, or as close as possible to the intended working environment. However, this type of experimenting can create limits in the broadness and complexity of test cases, since the environment of testing is not entirely manipulable; for instance, when we need to recreate specific external conditions or large amounts of resources. In these situations, we can observe reduction in the number of tests (which may lead to inaccurate results), delays or slow down in the test phases.

Concerning the study of traffic management and traffic patterns, these types of issues are crucial, since they can greatly impact the efficiency and organisation of urban areas, especially the ones densely populated - considering that almost every person uses at least one type of road transportation daily. In this sense, by improving the effectiveness and efficiency of planning and testing traffic solutions, we can more easily improve the organisation and quality of life in urban areas. The transition of the current SmartMob to a virtual environment would mitigate testing limits and issues as the ones referenced before, since the process of creating and experimenting can be performed repetitively and unlimitedly. In this sense, Computer Science allows us to have a degree of freedom of manipulation which can help us in situations such as this. Real environments can be replicated and recreated very closely to reality, allowing us to perform both realistic tests (where the environment factors are very little, or not at all, changed) and controlled tests (where certain environment aspects can be manipulated, changed or even removed). Controlled test can be especially helpful since some factors that exist in the real world may compromise and/or limit the tests in question. Space, energy and other physical resources are examples of elements that can be manipulated when transitioning from the real world to the virtual world, making it able for users to simulate scenarios or behaviours multiple times without having to worry about, for instance, financial spending or environmental impact.

Moreover, virtual reality offers further ways to visualise data when compared to other simulators, while also simultaneously allowing the collection of data. It also has the ”new technology” factor, being fairly recent to users but constantly evolving and improving. VR is increasingly present and available for more people, being very appealing to the users due to its interactivity, responsiveness and immersion. By using VR technology, we're also able to foster a more immersive working environment, and ease the interpretation of the data that students are intended to collect and analyse. For instance, it's possible to observe, in real time, information about an agent as it moves around the scene - such as the speed of a moving car.

As mentioned before, Virtual Reality presents the challenge of selecting objects, since multiple techniques are available but without clear knowledge of which is best. This project resolves around a dynamic environment with dynamic objects because users can move around freely throughout the different scenes with different angles to the same cities and the objects that will be selected are moving cars with velocities and routes through the city. Therefore, part of this project will involve the analysis of different selection techniques in order to choose the most adequate, efficient and effective one for the target environment of this study, taking into account the type of activities students will perform using SmartMob VR.

1.2 Problem

The Redes Veiculares, or Vehicular Networks, course unit, operating in the Taguspark campus, has a set of equipment that supports learning activities for its students.

Among that equipment we highlight the miniature vehicles, 1:24 scale, which have been transformed into intelligent objects, in order to serve as a basis for the test of intelligent mobility systems, in which it is necessary to study their movement and interaction with others in a small town.

Cars may or may not be controlled remotely and have two engines: one to control the speed and another one to control the steering. Each car has a detachable body and is coupled with a computational platform consisting of a Raspberry Pi Zero which is attached to the car's chassis. The Raspberry Pi runs implemented scripts. These scripts control the car's motors, its movement, the communication and interaction between the other cars and smart objects in the environment.

Using the normal operating mode, in the laboratory environment, the batteries in the car essentially serve to control the motors. The Raspberry Pi Zero is powered by a mini-USB port that can be connected to the PC or to a powerbank (placed on an acrylic carrier in the vehicle). To connect the powerbank to the car, the car's body is detached and the powerbank is placed on top of the acrylic carrier, making it heavier and therefore slower. While in demo mode, the whole system is powered by Lithium batteries, and the car's body doesn't need to be detached. In these circumstances, the test's duration is short since the engines and the Raspberry Pi require a considerable amount of energy but, on the other side, the car is lighter.

By observing and analysing the current testing environment, we are able to identify common problems that may arise, such as interference in communication channels, propagation of problems and access to the available environment, as well as the difficulty to control and coordination of the vehicles' movement. Considering that the multiple groups of students are sharing the same physical resources, the class' schedule is limited and even if there are multiple students using different equipment at the same time, there is the possibility of interference on the communication.

Additionally, as previously mentioned, it is essential that these test can be carried out on a larger scale, with conditions more compatible and more similar to the ones taking placing in the real world.

In the past, the course organised an activity at the end of the semester, where it was possible to test scenarios using multiple cars at once. However, this required a very effective and complex coordination of the work of each team of students and the sharing of the available equipment among the different teams - said coordination wasn't always compatible with the pressure and additional amount of work that the students were subjected to and have to balance at the end of the semester. In addition, the testing using multiple cars required additional physical space to set the roads in order to create larger and more complex road maps and scenarios.

A drawback of using Virtual Reality is that, given that its setup represents a relatively large expense, there may not be enough equipment for all students to perform their activities at the same time, requiring some sort of sharing or alternating system. If this case happens, it is also possible to find more accessible alternatives.

Another challenge to overcome is the selection of agents and objects whilst using SmartMob VR. Since the environment will contain multiple dynamic objects, while being itself dynamic, choosing which is the overall better selection technique can be tricky. The decision will heavily depend on what type of interactions the user will perform, and what type of behaviour the dynamic objects will have.

1.3 Objectives

With this project we intend to, not only, mitigate usability issues and limitations described previously, but also to make the usage and study of smart cars and other related components (such as the roads they circulate on and the buildings around them) easier and more appealing.

The core idea is to implement and test a tool that can offer the ability to create a virtual city, whilst providing enough freedom and flexibility that the students are able to replicate real models into virtual ones. Furthermore, the usage of technology will make the process of modelling more appealing to them, and provide freedom to work outside their classrooms or laboratories, where the current equipment is stored.

After the world of SmartMob VR is implemented, we need to decide on which selection technique to choose. Since the environment used will be dynamic, containing a good amount of dynamic objects in it, such as cars, it's crucial that the users of SmartMob VR are able to select objects with the least amount of effort and with a steep learning curve - meaning that the user quickly grasps how to perform activities using the technique and demonstrates a quick rate of skill acquirement when compared with the amount of experience it has using said technique. This analysis will require testing with users to understand which technique is better suited to the type of activities students will perform.

1.4 Thesis Outline

Chapter 2 talks about the related work regarding two topics: Virtual cities and Virtual Reality. For the first topic, it sub divides into two smaller topics: traffic and cities. Traffic is referred to data collection and interaction between the different traffic components. Cities talks about the representation and model of diverse digital cities. For Virtual Reality we look into navigation and selection techniques that are important for this work. The third chapter is about the solution made for this project. The architecture and the User Interface used for each component as the modelling and visualisation. While the project was being developed, integration tests were made for the modelling, visualisation and simulation. This is discussed in chapter 4. Tests were run in order to see the picked selection techniques' performances when put into the different scenarios This is discussed on chapter 5 with how it was achieved and the test's data. The document ends with chapter 6, concluding the work and talking about possible future work related.

Chapter 2

Related Work

In this study the related work's focus will be divided into two greater topics: ”Traffic and Virtual Cities” and ”Interaction in Virtual Reality”. Each topic will then be subdivided into two smaller ones.

The first one consists of know-how on how virtual cities are built and work, how to solve the issues of user movement in the environment, the integration of smart cars, but also knowledge on traffic itself - interactions between the agents in traffic, such as cars, changing lanes, traffic lights and pedestrians - and how to visualise it.

The second topic targets Virtual Reality. Two subtopics are approached: navigation and selection techniques, more specifically which techniques were studied and which was the one that fit best in the selection of dynamic objects in a dynamic virtual world.

2.1 Traffic and Virtual Cities

With the help of computers, where it's possible to accomplish huge calculations and complex simulations, the study of traffic, that is, the interaction between road agents and their components, has been deepened and diversified. An important factor to consider is the interaction between the cars studied and the other agents that take action in the same environment, such as the roads, traffic lights, traffic signs and other cars that circulate around the city. In this section we will look at a few works that can help you understand what to keep in mind when building traffic simulators.

Virtual reality is a tool that has been more and more used over the years, not only because it offers new ways of visualising the scene, but also can help in interpreting data in a new, intuitive and unique way. We can see in Figure 2.1, from a paper by João Miguel Santos, for his Master thesis [1], where there is a new way of visualising and interacting with, in this case, protein molecules.

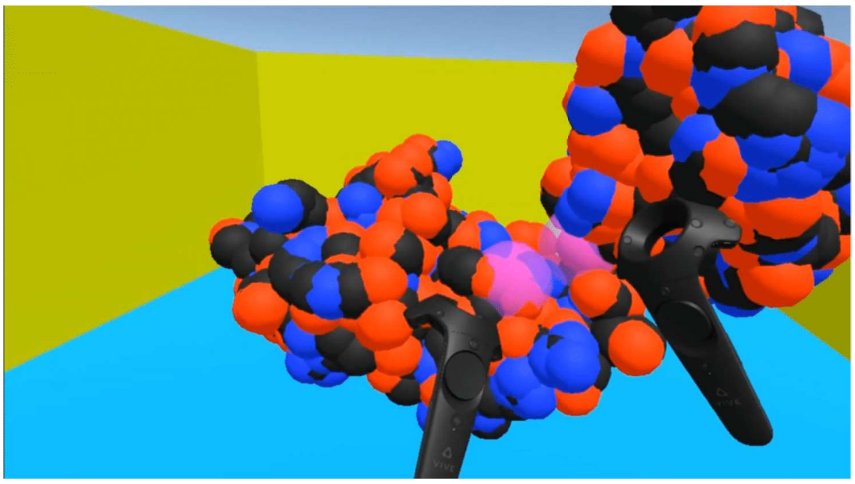


Figure 2.1: HTC Vive being used.

2.1.1 Traffic

In a given crash, a tool created by CATT Lab[2] can project temporary data into a timeline. Actions taken, such as changing lanes before the accident happens, are one of the examples of the data that this tool can provide. Since all the information is shown on a screen in a simple and intuitive manner, it makes it easier to detect critical information in order understand what events and factors led to the accident and what might be relevant to prevent future accidents from happening.

The CATT Lab also developed another 4D tool (Figure 2.2) to be able to perceive the state of the traffic in a more regional way (this means, information is located in areas of the map) and thus facilitating the interpretation of information. It is an interactive tool that simulates as if the user were in a virtual helicopter, flying over the intended city, whilst providing an infinite number of angles and distances from which he can visualise the scene. This tool is especially helpful in the management of the people involved in the accidents, the emergency staff - such as ambulances, and the general public that may be affected. The prototype also interacts with real-time databases of traffic and weather conditions. The aforementioned data is all stored in a system and then observed with further care and attention. Visualisation tools are quickly being requested by companies working with this sort of information since it is not easy to perceive and understand such large volume data. One of the tools that was also studied in this article was the Incident Cluster Explorer (ICE) that easily lets users analyse information from Virginia, D.C. and Maryland roads. ICE is very complex and tries to combine several widgets that then work together in order to help detecting problems to be solved; it also provides several different visualisation maps, such as histograms, 2D plots, parallel-coordinate plots and circular dependency graphs.

In addition to the above mentioned tools, CATT Lab has also developed a virtual, three-dimensional, multi-person virtual world - Virtual Incident Management (Figure 2.3), as a way of trying to train citizens and workers. This can facilitate not only the achievement of a much more effective accident prevention, but also a more efficient aid in these events.

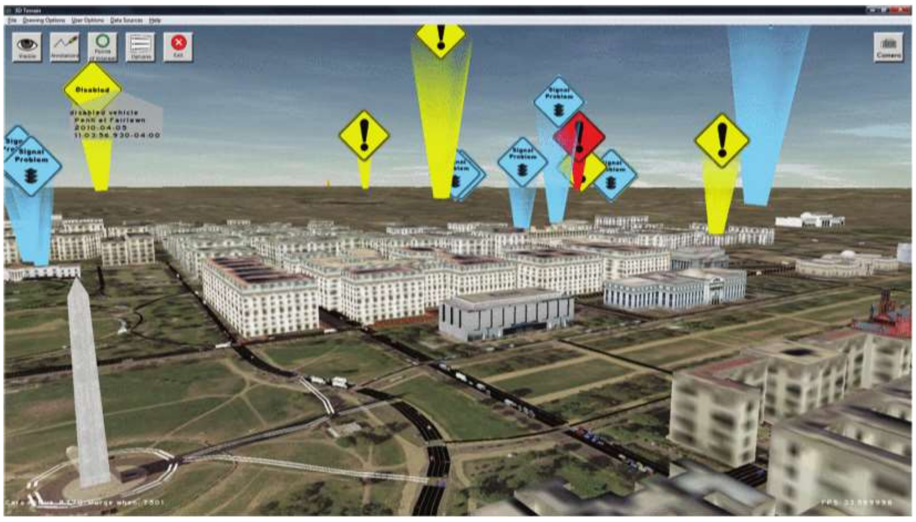


Figure 2.2: Real time visualisation of information (Source [2]).

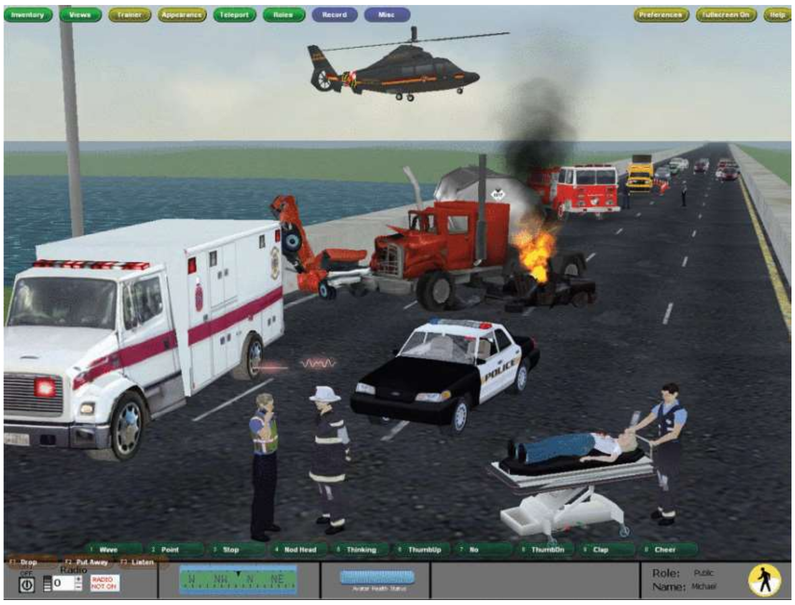


Figure 2.3: Screenshot of Virtual Incident Management (Source [2]).

A study that was performed [3], focusing on traffic management and analysis in virtual cities, demonstrated how efficient the use of virtual reality is when it comes to traffic simulation, for instance, when the goal is to determine the best possible alternative for a specific scenario. East Pilgrim Street is the area studied in this document, as it is considered a great area of opportunity for regeneration by the town hall, defined in 2006 by the Newcastle City Local Development Framework (LDF). This area is located near a main road and also has the potential to connect the South and East of the province. The objective of this work is to help the city council to make the best decision among three options. The goal is to improve the circulation, creating future hypothetical realities for each option. This is accomplished with the help of two tools: VISSIM, a simulation model that can analyse traffic operations of vehicles, traffic, pedestrians and bicycles under restrictions, such as configuration of road use, composition, traffic signals and traffic stops; and VR4MAX (Figure 2.4), an interactive three-dimensional simulation and visualisation tool used to create professional virtual reality applications and employed in this case mainly to analyse current and future traffic. The first step of the mentioned study would be the construction of a model that would reflect the transit conditions, with emphasis on the congestion that would occur. After the collection of data on traffic, and with great cooperation from the city council, they were able to start drawing the virtual reality system. This drawing had mainly two important steps: to create the virtual environment, the model, in which the simulation would be running and to create the simulation itself, running in the created model. The 3D Studio Max and VR4MAX tools were used for the modelling, while VISSIM was used for the programming of the traffic. After validation, to show the reliability of the virtual environment compared to the real environment, studies were done on traffic at different times of the day.

The VR4MAX is used to create professional applications in VR, for example, interactive design visualisations with third party applications (like VISSIM used in the previous project). This program is a plug-in that can turn rendered 3D models into a real-time interactive experience. User can navigate through it and gather information about the world around it. The project described above used MaxScripts to incorporate both programs together and giving valuable information about the possible outcomes when constructing the city. VR4MAX looks intuitive and a complete program, giving plenty of tools to worl around with, but to use its price can go from 1100 euros to 9900 euros.

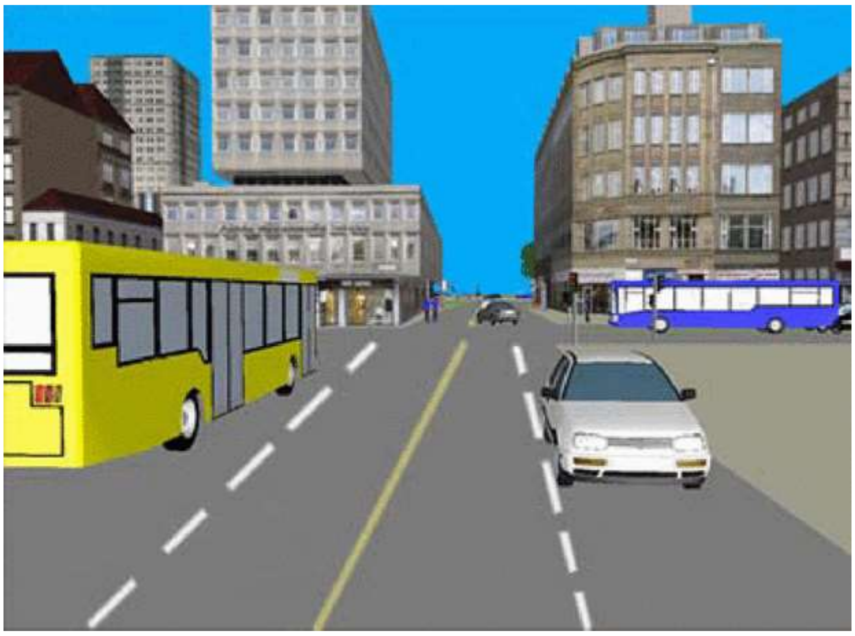


Figure 2.4: Screenshot taken inside the VR4MAX simulation (Extracted from [3]).

2.1.2 Virtual Cities

Virtual cities are, as the name implies, cities created and interacted with, within a virtual environment. To build environments of this sort it is important to know certain aspects such as how to do it, what dimensions to use, what properties to keep from the real world and which to discard. All these and other essentials can be acquired and studied through the analysis of different projects from various articles, and the understanding of how they approached their problems and motivations, and created urbanised areas according to the different objectives.

The process of creating virtual cities, if it is something to be done manually, can be a arduous job; it can take quite a few hours, and become very repetitive. There is a long process of creating every detail of the environment in question, such as the buildings, the streets, sidewalks and decoration like vegetation, lamps and traffic signs. Due to the fact that manual labour during the data acquisition process is a time-consuming process, a lot of the steps can be accelerated through automatic or semi-automatic processes. To obtain geometric data [4] a two-dimensional Geographic Information Systems, colour images and digital surface models can be used (Figure 2.5). The images for the Digital Surface Model are obtained through aerial laser scanners seeing as they offer homogeneous and high quality data of urban areas in great detail, not only of the buildings but also of the surrounding space. The overall project works with Geo Information Systems (GSI), colour images and Digital Surface Models (DSM). Ground plans, road routes and vegetation are obtained from GSI. This data is combined with DSM that automatically detects objects of interest such as buildings and trees to collect their structure, as height and texture. All of this is combined into a more complex model with real photo textures applied to the faces of the buildings thus creating a greater level of realism about the virtual objects.

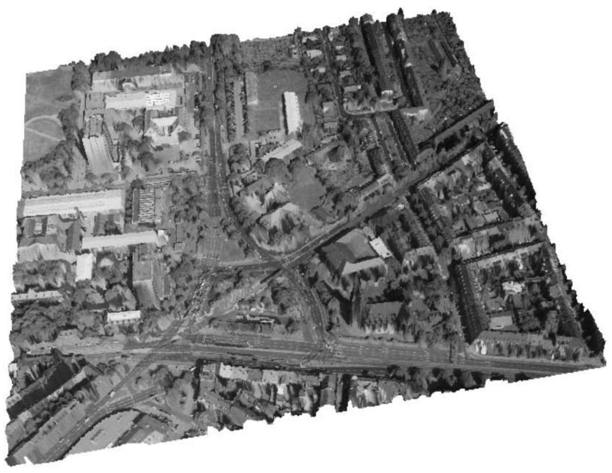


Figure 2.5: Three-dimensional visualisation of a digital surface model in an orthogonal image [4].

But city designs can also be done in a different manner. There is 2011 tool, well-known not only for its usefulness but also for its ease of manipulation and integration with other tools, called Simulation of Urban MObility [5], or SUMO for short. SUMO is a tool that allows you to draw city maps, more specifically, roads, and including, for instance, traffic lights run by algorithms. Simulation of Urban MObility also allows you to simulate communication between vehicles that navigate inside the city. It is a framework used in different projects to simulate self-driving cars or traffic management strategies (Figure 2.6). The user who is creating the map using this tool simply needs to draw the roads as one would do with brush strokes on a canvas, while also being able to edit their axes and place the traffic lights where they are needed and desired. After that, a simulation can be run by entering the number of cars on the road, extracted and later imported into Unity 3D as a three-dimensional map. Said map is automatically decorated, but can be edited later according to the liking of the user. In Unity 3D, a car can be commanded to test the roads that were created and to observe the interaction between the car and the others vehicles navigating autonomously.

There are several track design tools with different goals, ways of operating and user interfaces, spread across the Internet[1]. Scalextric Track Designer, for instance, is a tool with a very simple design, as shown in the figure 2.7.

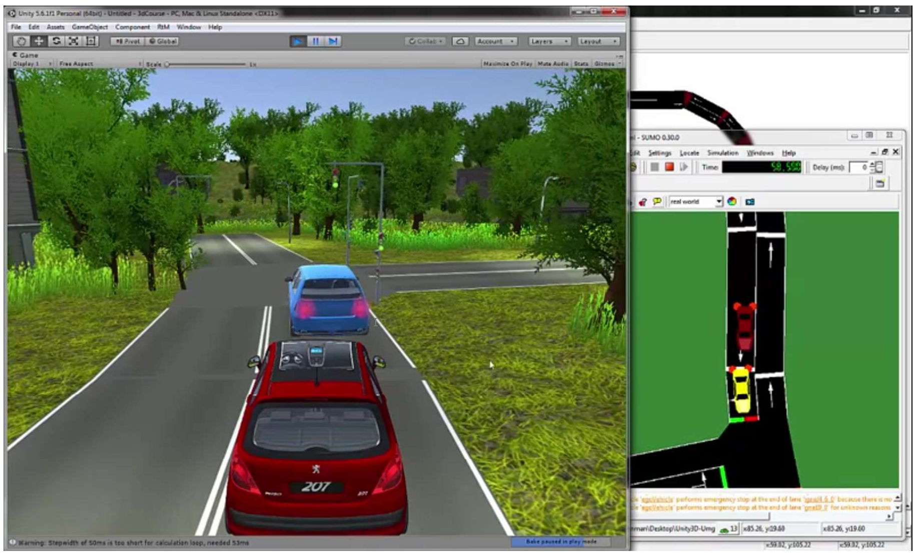


Figure 2.6: Car driven in a city generated by SUMO (Extracted from [5]).

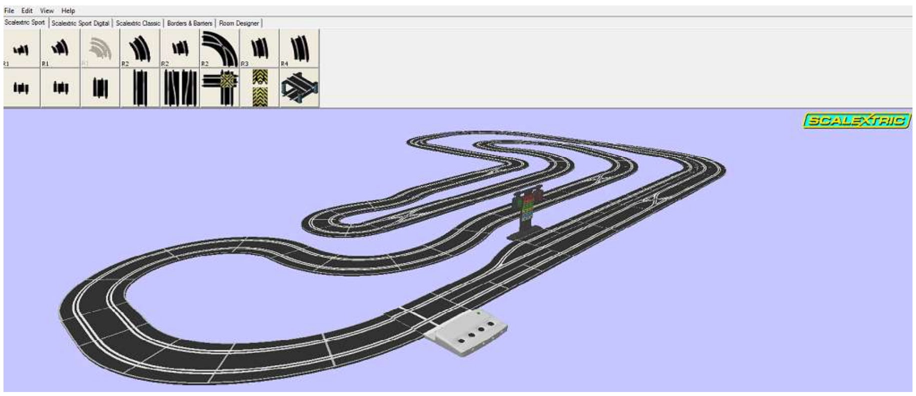


Figure 2.7: Creating a road with Scalextric Track Designer

In this tool, the program starts with an empty environment, except for a main piece of road that is used as a starting point for the road map modelling. The top of the program screen layout contains several lists of objects that are available for use, such as straight roads, curves, intersections, physical boundaries like barriers, and even imaginary boundary lines that can help you build roads in accordance with certain limitations (like the space available for building). The track being constructed can be seen in three dimensions and the camera can be easily manipulated just by holding down the left mouse key (in a space where there are no road pieces) and releasing it after moving the mouse to the desired position. When a component in the scene is selected, a user can simply click on another component in the top panel, and this new clicked part will be inserted immediately and automatically connected to the unit that was selected previously. Additionally, the new added component will be the new selected part to which the next component will be attached to, thus creating a track with just a few clicks. In case the user

[1]Track Design Software, slotcarwiki. https://slotcarwiki.org/tiki-index.php?page=Track+Design+Software

desires to edit an already constructed track, for instance, change its shape, he can simply select the road piece he wishes to change using the left mouse button, and holding it down as he drags the piece around the scene and into the new desired position. When the mouse button is release, the piece is set to its new place. If, as the object is placed, there are other parts adjacent, it will automatically attach itself to them in order to maintain the road map's consistency. Even though this feature is, for the most part, a good addition that makes the modelling easier, there can be situations where you do not want the pieces to attach, such as when you create two parallel roads next to each other - in theses cases, this involuntary fit can make editing a little frustrating and inefficient. Concerning the position of the pieces, the user can also press the space bar to rotate certain components to the right or left.

2.1.3 Virtual Cities in Virtual reality

Currently, there are also tools that allow you to build cities from within virtual reality. These programs, designed to unleash the creativity of the wearer, with glasses on a new environment, being able to be created and manipulated in real time.

CloudVR[2] is a good example of this. By using both remotes, the user can insert roads (Figure 2.8) and various buildings in the environment that surrounds him. The CloudVR application additionally offers a feature that shows the population of the city. The user can insert buildings which can be residential, commercial, industrial, offices, components of industries, and can also insert decoration, such as trees and lights. Because the customisation is as easy as picking up buildings and placing them where you want them, it can be a useful tool to easily preview a city that is being built in real time.

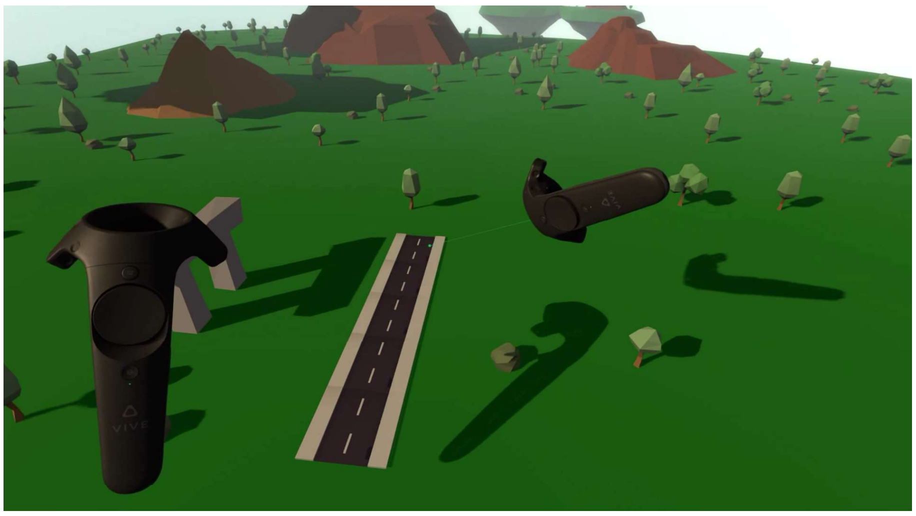


Figure 2.8: Building a road in CloudCityVR

2.1.4 Discussion

Concerning this project, we aim to achieve something similar to VR4MAX [3], where the user can see the city in three dimensions and have some possibilities of interaction with it. Although there are ways

[2]F. P. Interactive, CloudCity VR from Steam. https://store.steampowered.com/app/662950/CloudCity_VR/

to model complex and realistic cities, like we saw with GSI and DSM, the process and equipment to achieve it is too expensive and overwhelming for SmartMobVR. Building a city in virtual reality can be an interesting and fun activity, but a drawback of doing this is that users can take a few hours to complete the modelling. Modelling in a VR environment with precision and attention to detail might be harder to perform when compared with a two dimensional application. In the latter case, users can create and edit an environment in any computer without requiring any special equipment, but also without needing to worry some side effects of virtual reality such as motion sickness, spatial disorientation, vertigo or headaches. By taking this into account, we decided that to build the city we want something similar to the Scalextric Track Designer[3]. however, excluding its the automatic connection to adjacent roads or the three-dimensional view of the modelling. We want to make sure users can add and remove road pieces wherever they want on the grid and not just adjacent to other road blocks.

2.2 Interaction in Virtual Reality

Virtual Reality is a technology that exists for a while now but it has been getting more popularity and accessibility for more people. With this and the progress of technology there are more ways to utilise it. There isn't a global best way to navigate through different virtual worlds and not a global way to select different type of objects in those environments. This sub chapter talks about possible ways to navigate and select objects for our project.

2.2.1 Navigation

After our virtual city has been completed we will need to apply a way of navigating through the created environment or how to view it in it. Controlling a car is something that can be done through some of the equipment described above and by adding a few more components like a steering wheel and pedals, the user could immerse himself inside a virtual car in a virtual city.

In the context of virtual reality, there are restrictions for different equipment, whether it is the space of the room where the user is, the presence or absence of controls on the equipment or even limitations on some helmets. This means that there is no universal way of moving in virtual worlds. Even today tests are being conducted on what will be the best type of tool for exploring these environments.

The Virtual Reality Experiment Framework[4] is a framework that is freely available on the steam platform for academic purposes, which aims to study the movement in VR. The study focuses on three types of movement: Joystick Movement, Standard Teleportation, and Reference Teleportation (Figure 2.9). The first movement is the use of a track-pad and analog stick controller. The second is a transport method that allows the user to move from point A to point B instantly. Finally Reference Teleportation is a new technique that allows users to be in two places at once and giving them the ability to change their view from one to the other very quickly. For example, the user is standing in point A and points to

[3]Track Design Software, slotcarwiki. https://slotcarwiki.org/tiki-index.php?page=Track+Design+Software

[4]S. Howie, Virtual Reality Experiment Framework, Steam. https://store.steampowered.com/app/847650/Virtual_Reality_Experiment_Framework/

another point in the environment becoming this point B. Now on the user's hand there is a small square with the viewpoint of point B. The user can move this square, which is on his hand, and if the user puts the view totally in front of him, so he would immerse himself into that view, the teleportation would be complete. Scattered across the map are several stars that can be collected by users in the game. Each team is entitled to only one type of movement and the goal is to catch as many stars as possible in the shortest possible time. With some tests you can see what will be the best type of movement through the results obtained by different teams.

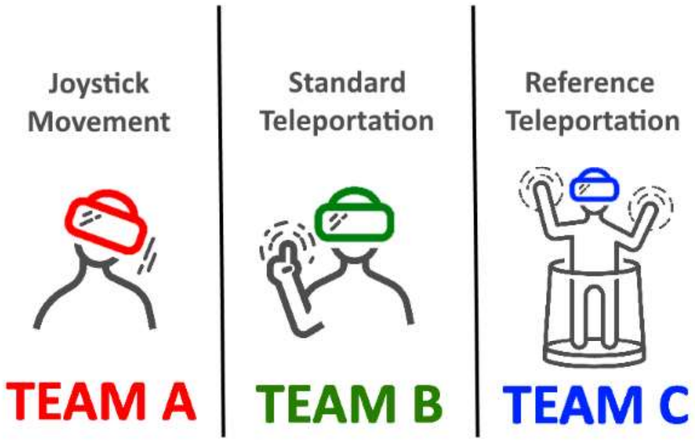


Figure 2.9: Different navigation types used in Virtual Reality Experiment Framework

In some countries driving tests are available, to get the driver's license or even for people with special care, with virtual reality equipment in conjunction with steering wheel, pedals, a seat and car interior accessories. Teen Distracted Reality and Interactive Virtual Education or D.R.I.V.E. Teen Drive[5] is a program that teaches young drivers about the risks associated with inexperienced driving. The simulator attempts to reproduce driving with distractions or situations where judgement may be dubious (Figure 2.10). Teen Distracted Reality and Interactive Virtual Education was a suburban Chevrolet with two simulators in the backseat with computer screens, steering wheel, pedals and seat belts. Using a virtual reality helmet and the aforementioned equipment, users can navigate through the virtual cities placed in the scenarios in question.

Navigating in Virtual reality can also be done by moving the user progressively as if he was flying. There are several techniques to do this, but the hand technique looks to be the most efficient[6]. This study has three different techniques, Elevator+Steering technique, gaze technique and hand technique (Figure 2.11). The most efficient technique from this study was the hand technique. This technique resolves around the user using his hand to define which direction to move to. If the hand is pointing upwards, the direction will follow that way. This way users can look around while pointing to the desired destination.

[5]E.Network@WRIcities.Using Virtual Reality to Create Safer Drivers in 2017. https://www.smartcitiesdive.com/ex/sustainablecitiescollective/friday-fun-using-virtual-reality-create-safer-drivers/1090332/

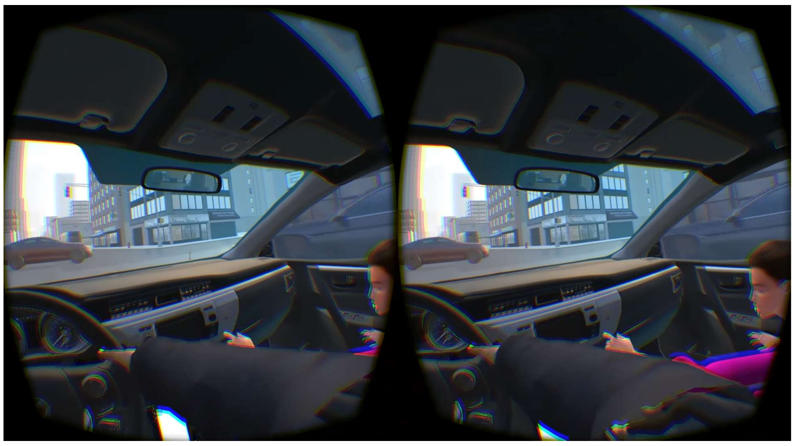


Figure 2.10: User driving a virtual car in a virtual city interacting with virtual people

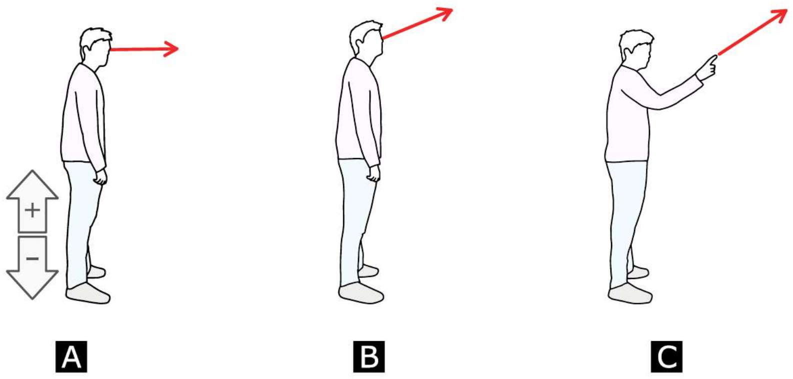


Figure 2.11: Three different navigation techniques used in Virtual Reality. (A) Elevator+Steering technique (B) gaze technique (C) hand technique (Extracted from [6])

2.2.2 Selection Techniques

There are a lot of studies and papers regarding selecting static objects in virtual reality. All of which try to fix jittering or occlusion problems. In our scenario we want to select dynamic objects in a dynamic world. This type of techniques are still something not studied in much depth.

A simple ray being shot from the user's finger may be a straightforward approach to selecting objects, but like most techniques in some cases the hand and/or device's jitter may compromise the selection of the desired object. Another big issue is occlusion, a simple ray can't select an object that is behind another or that is covered with multiple other objects from the scene, since the ray will simply select the closest object to the ray.

To select an object that is behind another or a group of others we could use iSith[7] which is a technique that uses two rays coming out of each hand, selecting then the closest object to the intersection of both these rays (Figure 2.12). This looks like a simple and effective solution but aside from the jitter for a big and complex city in a virtual world, with a lot of cars, people and environment objects a ray cast or an intersection of two, these techniques would mostly fail in a lot of selection exercises. Trying to select a distant object with iSith would be very hard since both rays would appear to be parallel and now we would have two hands that are jittering so the precision would be lower than a normal ray.

Some techniques like Go-go[8], Depth Ray[9], Lock Ray[9] and Flower ray[9] are very useful for

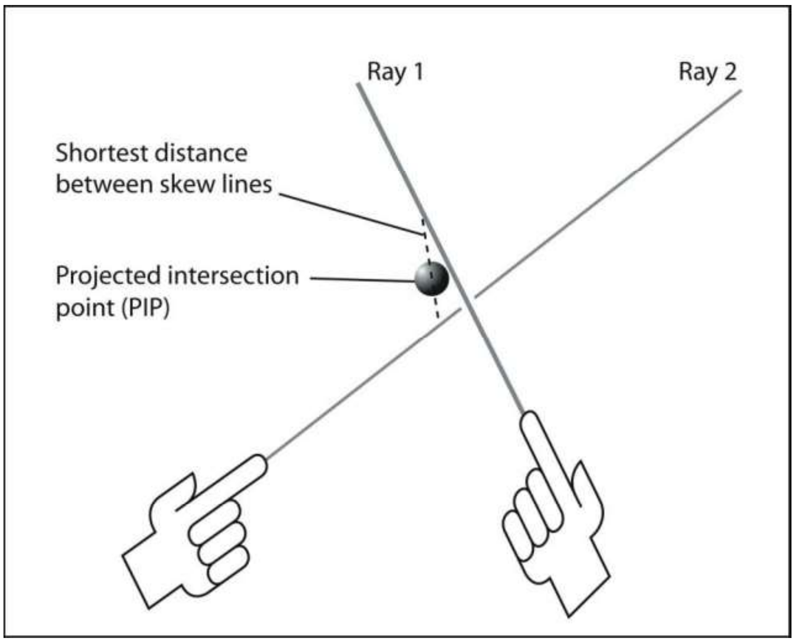

Ray 1
Ray 2
Shortest distance between skew lines
Projected intersection point (PIP)

Figure 2.12: iSith example (Source[7])

selecting occluded objects.

Go-go helps to give the immersion of the users still using an hand and being able to grab objects, the hand is extensive so the range to select an object is higher(Figure 2.13).

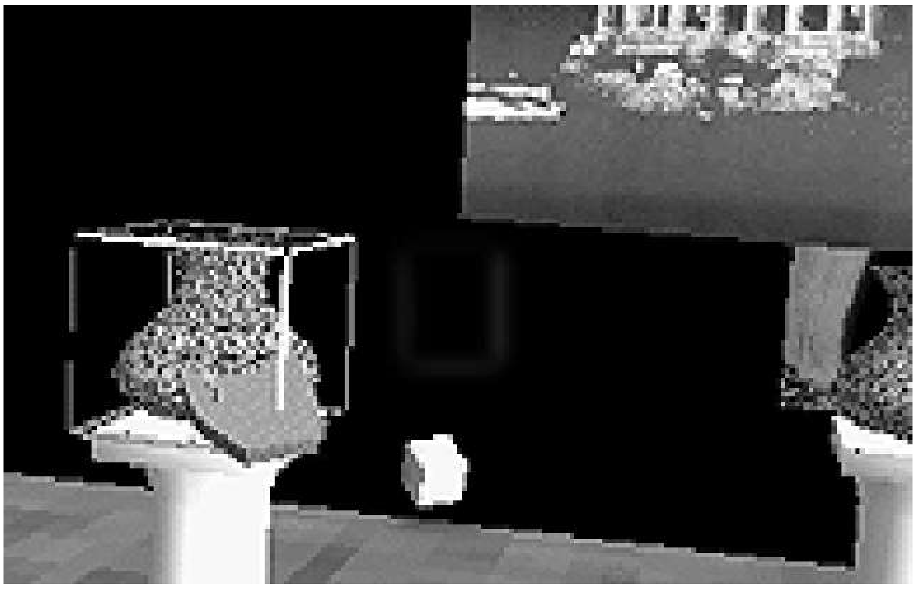


Figure 2.13: The Go-Go technique allows users to expand their reach. The white cube shows the real hand position.(Source[8])

Depth Ray is a normal selection ray coming out of the user's hand but this ray has a marker in it(Figure 2.14). The user can manipulate the marker so it moves around the length of the ray and selection is made when the marker is close enough to the desired object and the selection button is pressed.

Lock ray is a similar technique to Depth ray but this time there needs to be a phase where the ray

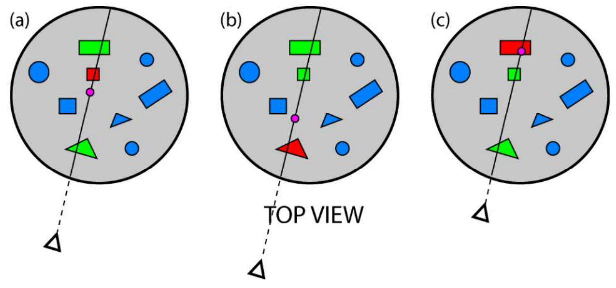


Figure 2.14: The depth ray. (a) A pink depth marker is used to select the closest intersected target. (b) Moving the input device backwards selects a closer object. (c) Moving the input device forwards selects the further target.(Source[9])

gets locked in place with a press of a button (Figure 2.15) and only after this can the user move the marker on the ray marking it easier to select occluded objects.

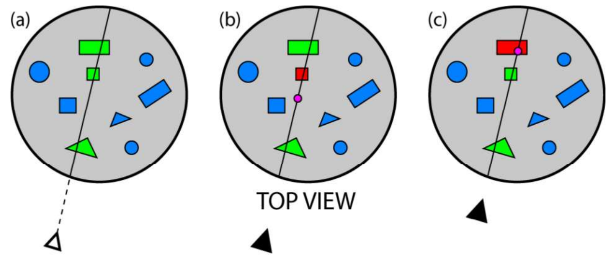


Figure 2.15: The lock ray. (a) All intersected targets are highlighted. (b) Holding the button down locks the ray and displays the depth marker at its center. (c) The input device is used to control the depth marker, selecting the closest intersected target. (Source[9])

Flower ray allows the user to shoot a ray and select as many objects as it intersects. Only these objects are then shown separately and remain static so the user can choose what object to select for his or her final selection (Figure 2.16).

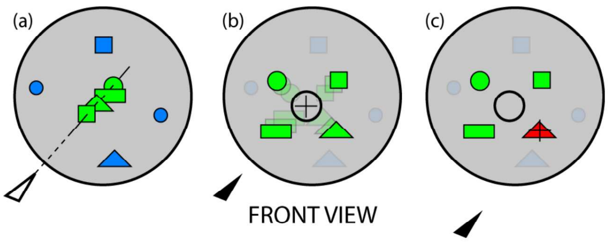


Figure 2.16: The flower ray. (a) All intersected targets are highlighted. (b) Holding the button down causes all intersected targets to flower out into a marking menu. (c) The input device is used to select the desired target from the marking menu.(Source[9])

Jitter is still an issue with the techniques mentioned in big environments, like the cities we will be working on. The techniques have good ways of selecting idle objects, which is not the case of our study. Lock ray, for example, not only breaks the immersion in a Virtual Reality World, it has to lock the ray in a position, and if we wanted to select a moving car, this would not be possible since the car would be in a different position while the ray was in a fixed position.

Still regarding distant objects, a study was made on a technique called Worlds in Miniature (WIM)[10], this would put a miniature version of the world on the user's non dominant hand so he could use the dominant hand to select objects from that miniature world (Figure 2.17). Objects that were selected and manipulated in the miniature world would also be manipulated at scale in the virtual world the user was on and vice versa. This way far away objects or occluded objects would, more easily, be manipulated in the environment. If the world is too big or has too many small objects, then the detail on the miniature world would be too small and would make it even harder to select the desired object.

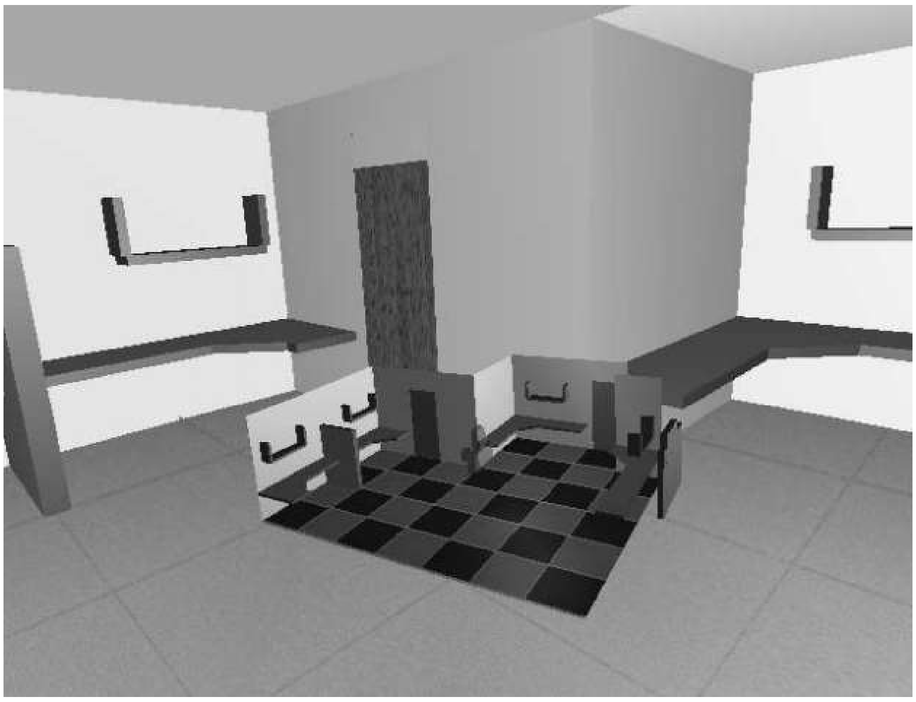


Figure 2.17: The World In Miniature (WIM) viewed against the background of a life-size virtual environment. (Source[10])

Continuous rays make for a great approach regarding dynamic and/or objects that are far away. Smart Ray[9] allows the user to have a continuous selection on objects and selecting the pretended one without much effort. This technique uses an algorithm based on target weights to determine which target should be selected when multiple targets are intersected. Target weights are continuously updated based on their proximity to the ray cursor. The closer the ray is to the center of the target, the larger the weight increase will be. When starting to select a target, a ray is shot and a green highlight will appear in every object intersected (Figure 2.18(a)). The intersected target with the highest weight is highlighted red, indicating that it can be selected by clicking a button. When the ray intersects multiple targets, the user can re position the ray so that its new position still intersects the intended target. Even if multiple targets are selected by the new ray position, the intended target will have the highest weight, as its weight has been continuously increasing.

Hook [11] is a similar technique to Smart Ray since both techniques have the continuous refinement of shooting a ray at an object for some time until it is selected. This technique was tested with dynamic objects, unlike smart ray, but with a two dimensional environment with a mouse, which is not the case for us.

Feedback is also very important when comes to the selection of objects, bubble ray[12] is a technique that has a bubble at the end of the ray and this bubble increases in size so it touches only the closest

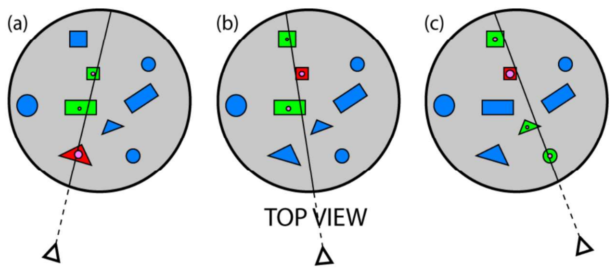


Figure 2.18: Using the smart ray to select the small square. (a) Target weights are based on the distance from the ray to the target, visualised as spheres in the center of each intersected target. The target with the highest weight can be selected. (b-c) The ray can be re positioned to select an occluded target, by continually increasing its weight.(Source[9])

target to the ray(Figure 2.19). The bubble only touches on one object and can even bend its form so it doesn't cover more than one at a time. Bubble rays make it hard to select occluded objects and sometimes dynamic ones as well

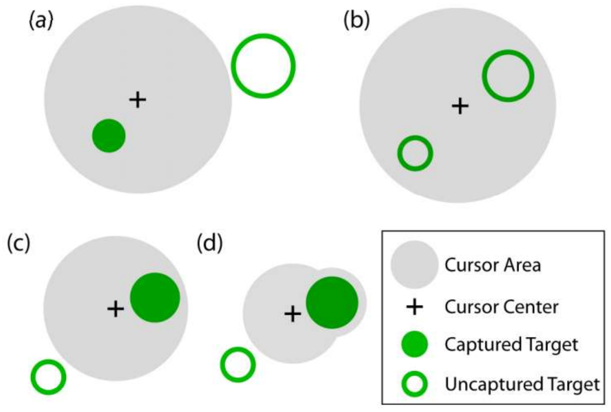
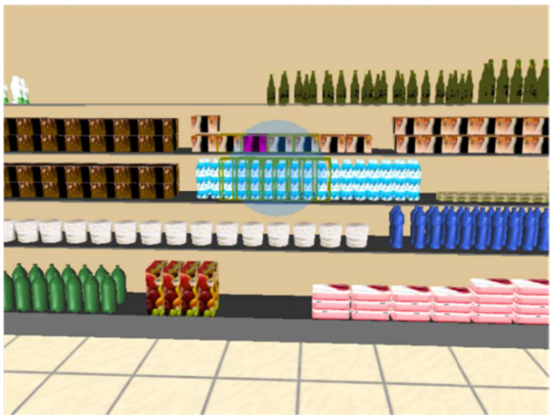


Figure 2.19: (a) Area cursors ease selection with larger hotspots than point cursors. (b) Isolating the intended target is difficult when the area cursor encompasses multiple possible targets. (c) The bubble cursor solves the problem in (b) by changing its size dynamically such that only the target closest to the cursor centre is selected. (d) The bubble cursor morphs to encompass a target when the basic circular cursor cannot completely do so without intersecting a neighbouring target. (Source[12])

Another technique with a bubble type ray is SQUAD[13]. SQUAD is a ray with a bubble at its tip, but this time, the bubble changes size based on how far the object is to the intersected ray. This way the bubble may select many objects at once(Figure 2.20a). After the first selection, a new screen will pop up divided in four, with the intersected objects distributed on those triangles and the user has to select the desired quadrant on which the pretended object is in(Figure 2.20b). This looks like a great approach but the new window or user interface breaks user immersion which is not desired for our Virtual World.

Other than rays many studies use cones for selecting objects, this way jittering is less of an issue, because this way the ray does not have to hit the pretended object. Occlusion is a problem with a simple cone technique so there are a few that have a better approach regarding this.

There are regular cone techniques where all objects inside the cone are selected. Other iterations of the regular cone is a Flashlight type of technique where the user still has a cone but there is a refinement. This refinement consists of selecting the object closest to the center of the cone and removing all other potential objects.


(a) Tip of SQUAD's ray forming a sphere and intersecting many objects

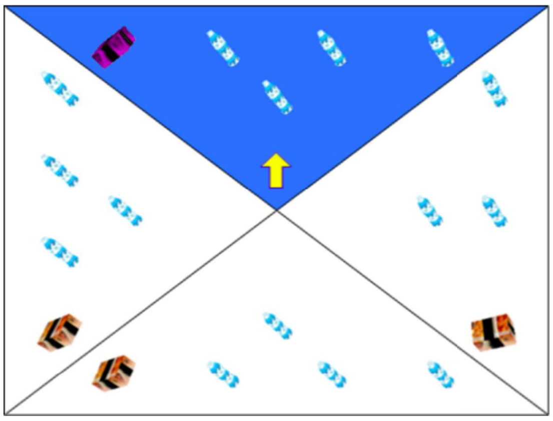

(b) SQUAD menu where the user selects the quadrant with the desired object

Figure 2.20: Example of using SQUAD

All the objects within Shadow Cone[14] are selected when the user presses the button, by holding it, and as the user moves their hand only objects that are always within the cone are selected when a button is released. When the user presses the button, the potential selection set is initialised with all objects that are within a target angle of the ray from the hand. On each frame any object that is no longer within the target angle is removed and it's deselected from the previous group. When the user releases the button the objects remaining in the potential selection set are then selected(Figure 2.21).

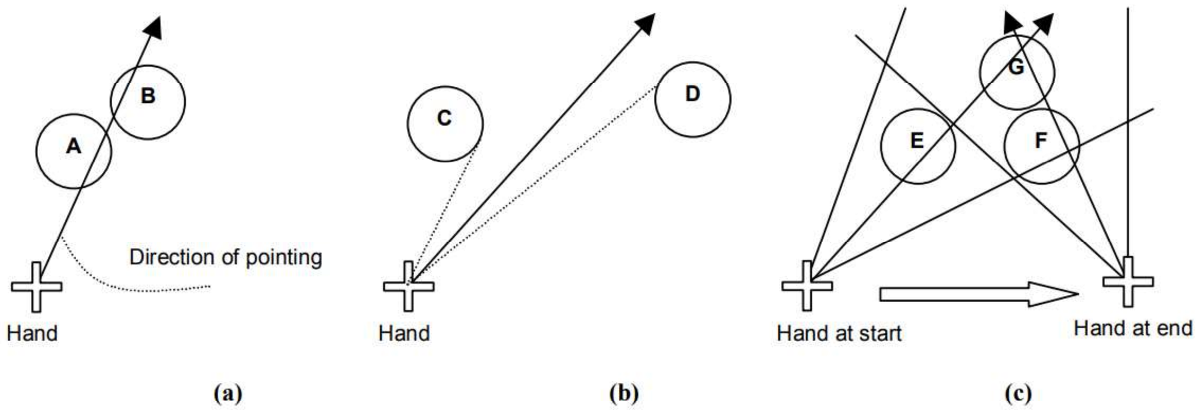


Figure 2.21: Shadow cone example (a) Ray-based selection is effected by choosing first object to intersect the ray from the hand (Object A) (b) Cone Selection is effected by choosing the object that is relatively closest to the line (Object D). This is indicated by the angle to the dotted line to Object D subtending a smaller angle to the direction of pointing than the line to Object C (c) Shadow Cone Selection is effected by choosing the object that is within all the cones projected from the hand (Object G). In this case both Objects E and G are highlighted at the start, but E is dropped as the hand moves to its end position and direction ([14]).

To ensure a seamless and effortless transition from ray casting selection behaviour, it is taken a simple ray-casting as a basis for the scoring metric. To get this, it is assigned the highest score of 1 if pointing is most accurate(Figure 2.22). That is, the ray is being pointed through the object's center point. It is also assigned the lowest score of 0 if pointing is least accurate, when the object's center point lies on the bounding surface of the conic volume. The scoring contribution is accumulated over time. Each object maintains its score over several time frames. That is, the score is not reset at every frame, but kept during a sequence of frames. So everytime the object is selected its score increases and when it stops, there is a decay. In the end of the selection ,the object with the highest score gets selected.[15]

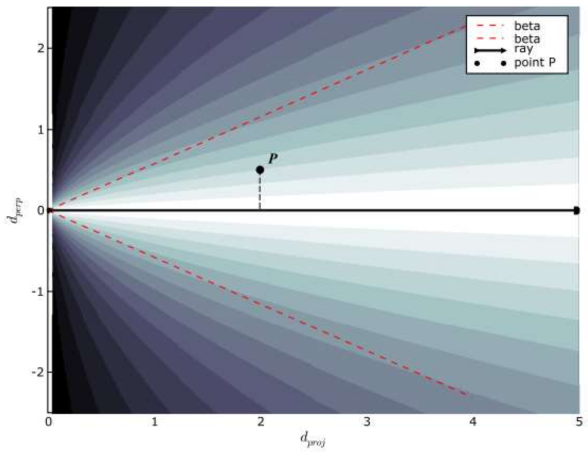


Figure 2.22: Object Scoring Function: Scoring value of point P inside the conic selection volume. (Source[15])

2.2.3 Discussion

For navigation purposes it will be used the Hand technique studied in [6] because it looks the most familiar and had great results in their testing.

Each technique is unique in its own way.

There's different selection volumes for each technique: Point, Ray, Bubble and Cone. Point techniques are the ones where there is an analogy on the user grabbing the objects. If we imagine a hand picking an object with the thumb and index finger, doing a pinch motion, we can see a point in space. If this point touches or intersects an object, this object is selected. The point created by the user doesn't have to be in the arms' reach of the user, for example in the Go-go hand technique, the user's hand is extensible and they can control it's reach reaching distances further than the user's hand, but still using the point analogy to pick objects up.

Ray and Cone techniques are self explanatory, first of all the object has to be intersecting either the ray or the cone when the user is trying to select it. Either a ray, which is a straight line, or the cone usually have their starting point on the user's tip of their index finger and has its direction is where the finger is pointing at.

Finally, there are bubble techniques. These techniques use rays as well when it comes to drawing a trajectory from the index finger to the desired location but the bubble appears at the ray's tip. The bubble may change in size and shape according to include objects in its area, or, like the SQUAD technique the bubble area changes according to the distance from the user.

When it comes to types it is important to separate single and multiple techniques. There are some techniques that are worth mentioning that they can select multiple objects on their final selection. This means the user may want to select one object but if other object is in the same conditions as the first one it will also be selected. This is what it means to be able to select multiple objects. The final result may be more than one.

Then we have different refinement methods. By refinement we mean how the algorithm chooses the object over another if there are more than one being intersected by the selection volume. There are continuous refinement and discrete refinement. The main difference between the two is the amount of steps or inputs from the user for the refinement to be completed. Continuous refinement resolves around constantly excluding objects from the desired one. For example, Smart ray is a technique that focuses on adding weight to objects that are selected, the refinement is always being made since the objects either keep getting more weight, then being intersected by the ray, or have their weights stable. Discrete refinement requires the user to give more inputs when trying to select the object. By this we mean that there are more than one step to select an object. It is an iterative process. SQUAD, for example, has a bubble at the end of the ray, which has the possibility of selecting more than one object from the first user's input. If this happens a new screen will pop up and show all the selected objects in a new User Interface, because the user has to point again and give another input, this refinement is considerable discrete.

Lastly, we wanted to show the amount of studies made, from the different techniques, in dynamic environments since our work will work around dynamic objects in a dynamic world.

Selection Techniques					
Name	Selection Volume	Type	Refinement		Dynamic tested
			Continuous	Discrete	
Ray	Ray	S			
iSith	Ray	S			
Go-go	Point	S			
Depth	Ray	S	X		
Lock	Ray	S		X	
Flower	Ray	S		X	
WIM	Point	S			
Smart	Ray	S	X		
Hook	Ray	S	X		X
Bubble	Bubble	S			
SQUAD	Bubble	S		X	
Cone	Cone	M			
Flashlight	Cone	S			
Shadow	Cone	M	X		
IntenSelect	Cone	S	X		

Table 2.1: Selection technique's table

We can clearly see there are not a lot of techniques being tested with dynamic objects nor in dynamic environments.

Chapter 3

SmartMob VR

In the process of creating this solution we wanted to develop tools that would allow students to test and run simulations in a more comfortable, affordable and appealing way.
The solution is divided into two distinct parts: The modelling of the city and the visualisation of the system containing the city itself, the simulation of the traffic, interaction of cars with the remaining road elements and the usage of selection techniques.
Modelling is the creation and manipulation of the city, which will begin as an empty project in a computer program. The city modelling software uses a mouse and keyboard to provide the modelling inputs and the process is viewed by a monitor, thus being the system's output a two-dimensional view of the city.
The second part of the system has the objective of giving a visualisation of the city that was created in the previous step and its components, in three dimensions, in virtual reality environment. The simulation's visualisation is in direct contact with the simulation and the users as this is where the information about the cars, like their location, and car controls come from (Figure 3.1).

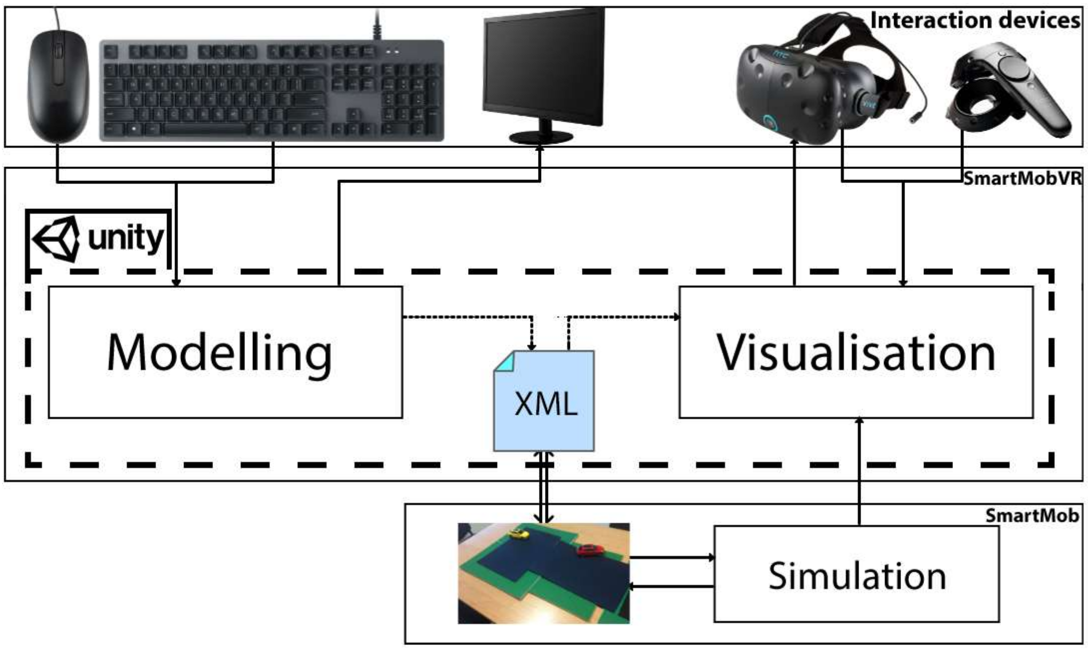


Figure 3.1: Solution architecture

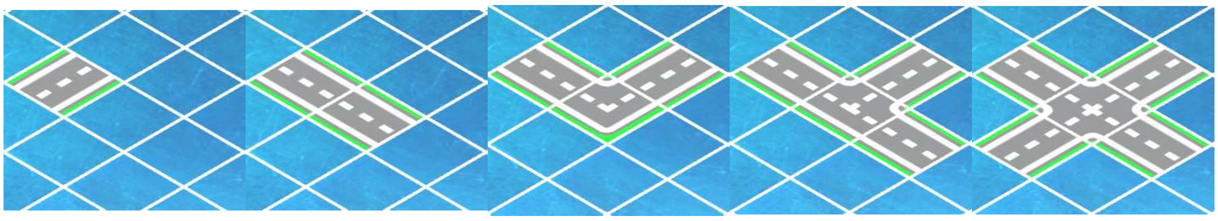
The document created by the modelling part of the solution can, and while testing, is the objective to, be similar to the miniature city of the room with the equipment and cars communicating with each other.
The proposal is divided into two parts , and in which part it is going to be discussed the architecture and the Interface.

3.1 Architecture

In this section we will talk about what was done in the process of creating the solution and how it works. Going from overall functionality to specific details in features, both for modelling and visualisation part of the application.

3.1.1 Modelling

For modelling the city, the use of mouse and keyboard in a computer was chosen to facilitate the activity in a classroom environment, where several people can model at the same time, which also gives the possibility to model the environments outside of the university environment, and thus out of school schedules, making it more versatile for students to model it anywhere anytime. As the modelling process can be somewhat lengthy, this approach was also chosen because the glasses can cause nausea if worn over a long period of time, which depends on person to person, leaving then only the visualisation of the city in virtual reality.
The modelling of the city is done in a executable program built in Unity 3D so it can run in any Windows computer with a recent version of the operating system.
The positions where the objects can be placed are delimited by a white grid on a blue background since it makes it easier for the user, for processing and generating the output file for the modelling itself. The grid can be moved around freely while holding the left click on the mouse and moving the mouse's cursor around, the grid moves with a one on one relation to the mouse so it seems like the the user is grabbing it with it's hand making it more familiar to move around.
Users can clean the whole grid, removing all the objects in it. Open previously exported XML files by the modelling program, after clicking the ”Open” button a new window is brought up and asks the user to select a XML file from his or her computer to import it to the current grid. Finally, files can be written with the ”Write” button, this pops a window so the user can select the destination and name of the XML file that is going to be created with the objects in the current grid.
Because it would be hard to model, since some objects might occlude others or the scene may be too confusing at times, with a camera with the same direction towards the grid, it was implemented the option to rotate the camera. There are two buttons that allow this. Each button makes the camera turn either ninety degrees or negative ninety degrees, depending on the button, with the center of the screen as the middle stationary point.
Zooming in and out is also a feature in this program. Sliding the mouse's wheel upwards will make the camera move closer to the grid, and doing the opposite will move it further.



(a) Single road piece on the grid (b) Two road pieces put next to each other (c) Intersection with three road pieces (d) T crossroad, with four road pieces (e) + crossroad, with five road pieces

Figure 3.2: Example of road pieces forming different paths

The modelling will be done by first selecting the desired object, on the list on the right part of the screen, that the user wants to put on the grid. This is done by clicking on the object with the mouse's left click over the object on the list on the right, this will make it have a small highlight so the user knows which object is selected at the certain time.
After the object is highlighted the user can now click on the desired grid and left click on it. There is another highlight feedback on the grid itself so the user is sure where he or she will place the selected object. When the left click on the grid is done, the object that was previously selected will now be placed in the desired position. The user may place as many as he or she desires in other grid cells until he selects the same object on the list on the right to deselect it.
The list of objects include Roads, houses, trees, statues, street lamps and traffic lights.
Each object can be right clicked once the object is on the grid and new options menu will appear with the objects properties. The main property is different from object to object and we will go over them briefly one by one but all of them have a "Del" button to delete the object if necessary.

Roads

When the road object is selected it is not placed like the other objects on the grid. The user selects a position on the grid by left clicking on it and a new road with two ways and grass on the sides will be instantiated. Now everytime a new road is placed, and if this new one is adjacent to another road already placed on the board, an algorithm will run for both so they fit like a puzzle piece.
For example, if a piece is laid on the grid and a new one would be placed next to it, the first one would turn around so it's adjacent to the new one and it looks like a normal straight road (Figure 3.2b).
Making turns (Figure 3.2c), T like intersection (Figure 3.2d) and + like intersections (Figure 3.2e) are also possible with a similar process. When a new instance is placed on the grid all adjacent roads, and itself, will compute how it is the best way to behave themselves in order to make a road that connects to the its neighbours.
With this algorithm students can replicate the scenarios in the laboratory with the carpets and the cars (Figure 3.3).

Figure 3.3: The laboratories equipment: some cars and roads, that can be inspiration to build the city with SmartMob VR

Houses

The House object represents building that will be shown in the virtual reality world. In that world their colour will be randomly generated but while modelling, the user, with the edit menu (right clicking on the object), can edit the number of floors of the house.

Trees

The Tree object represents three different types of trees that will be shown in the virtual reality world. In that world its type will be randomly generated but while modelling, the user, with the edit menu, can edit the number of trees on that grid cell.

Statue

The Statue object represents a simple aesthetic statue that will be shown in the virtual reality world. While modelling, the user, with the edit menu, can edit the statue's rotation on that grid cell.

Street lamps and Traffic lights

Street lamps and Traffic lights represent street lamps and regular traffic lights but these objects are not like the previous objects. This objects do not occupy a whole grid's cell. Users while having one of these selected will not have a highlight like the previously shown on the grid before placing them. The objects will be placed on the edges of the grid cells. The objects will both be placed like it can be seen in and the user can edit their rotation on the grid with how many degrees as they want.
Every object was either bought from the unity asset store or downloaded for free in Blender or just public libraries in public websites.
At the end of the modelling process, the final result will be a blue print created and manipulated by the users that can then be exported in to an XML file. These files contain the modelled city information, such as the object ID's, properties, for example, rotation of the roads, the buildings' number of floors,

the amount of afforestation that will be used to place objects, and position. These files can then be re-opened on the same program or opened in the Virtual Reality SmartMobVR simulation program.

3.1.2 Visualisation

The simulation and visualisation of the city is done with Unity 3D as well, where the city model, the implementation of the cars and the selection techniques are combined in one environment.
The city model will be done by importing a XML file generated in the previous step, when modelling the city. The program checks object by object in the file, instantiating every object according to its ID and position, then edits its properties to correspond to the given ones. Depending on the ID given, the object will be rotated, added more floors or even number of instances per position.
The cars' movement was controlled either by a script, coded in Unity or by the SmartMob simulation.
To have moving cars in our virtual city, besides their script, we need models. To do so, we started with objects representing cars, taken from the SimpleTown asset from the Unity Asset Store, in unity that were coded in C#. These scripts were used in experiments because they had to have the exact same behaviour every time the program was played. All the cars would have the same path for each scenario. The script consisted on moving the car forward and when encountering invisible objects in the map, make them turn either left or right, depending on its properties. These invisible objects were manually put throughout the different scenarios so the cars would have set routes.
The simulation will be specified in depth in the next subsection.
When developing the visualisation part for this project another problem rose: how will students select the cars to visualise its information, for example? We wanted to be something familiar, that wasn't frustrating and had a good success rate when choosing what was desired. A simple ray or cone was not enough since objects can be far, obstructed, are dynamic in this dynamic world and have unpredicted movements. We can see an example in figure 3.4 where the user would have some difficulty trying to select a car with a simple ray shooting or cone technique. Near cars move too quickly and cars moving in the opposite direction are obstructed, to solve this, the user may fly upwards to see the scene in a more distant viewport but selecting moving objects at a distant is also difficult because the user's hand and the equipment may jitter. For this, three selection techniques were chosen and tests run to see which is the best for this environment.
The selection techniques are chosen before the simulation starts to run, in the unity editing tool. Each technique produces a CSV file that contains information about the selections done.
Within the visualisation the user will see the city created but this time in three dimensions with the help of the Oculus rift glasses.
Besides the HMD, users will also have two controllers with them, one for each hand. Each controller has a unique purpose.
The left is used for moving the camera around the virtual scene. The user is able to see his or her hands inside the virtual world. The direction of where the player in the game will be heading is obtained while pointing with his or her index finger with the controller. To move around, in this virtual world, and

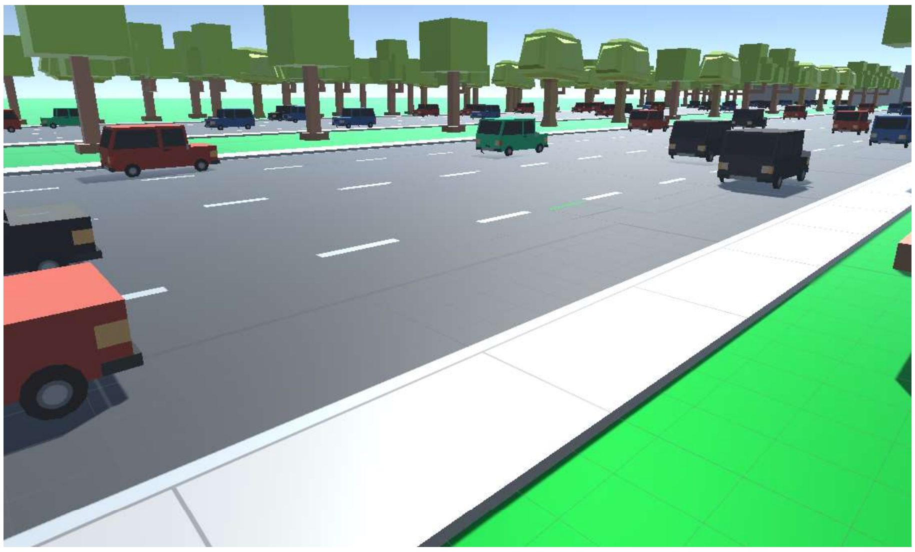


Figure 3.4: Example of a scenario where it's difficult to select a car

according to the index finger's direction, the user can simply move the joystick in any direction.
The right controller is used to activate the current selection technique. Like previously stated, the technique is selected before hand when the program is about to be played. Once it starts the user can point with their right hand and click on the back button of the controller and start the process of selecting an object.

3.1.3 Simulation

With the integration of SmartMobVR, the cars, from the laboratory in which the students are performing tests, can communicate with the cars in SmartMobVR. When in the same ad hoc network, the physical cars send CAM type of messages through UDP to the VR application. These messages sent by the cars have two parameters that are used to locate the car: its latitude and longitude. If the cars are used outside and are equipped with GPS they can send the coordinates used by the device, otherwise, an estimation is used instead. The velocity is determined beforehand, according to its battery life and the script controlling its wheels. With the velocity, being distance over time, the script uses the current and initial time to calculate where the car can be positioned at the current time. The CAM message is sent with these parameters calculated.
The visualisation part of SmartMobVR is prepared to receive UDP communication. Every span of time, for example, a third of a second, a new update on the car's position is sent to SmartMobVR so the respective car (the information on which car to update is taken from the CAM message's parameter ”id”) would move towards that position. The desired car starts at a given position, this information is gathered from the first message received, and will save up to 10 pairs (latitude and longitude) in a position array queue. From the moment this queue has 4 or more pairs the car starts to calculate where to move next. It was chosen 4 pairs of coordinates because, even though the car only needs 2 to start moving, sometimes the simulation would take longer to send a third or fourth message so the car would make a wrong assumption or stop in the middle of its movement. Giving an extra one or two positions reduced

the number of bad assumptions. This way, there are 2 more positions on which the car can rely on while the simulation sends more positions. There is some delay according to the car in the laboratory but not much difference is noticeable with fast messages from the simulation.

The CAM messages also include a variable that represents the time on which the corresponding message was sent. With the time and pair of coordinates for each message, the velocity and direction is calculated and applied to the car. After the car reaches the second position, corresponding to the desired position at the given time, the first position pair is discarded from the queue and the same algorithm applies with the next position.

3.2 User Interface

This section will cover the visualisation aspect of the application that will be seen by the users.

3.2.1 Modelling

The application that allows users to model the city has a very intuitive interface: starting from the right side, the user can find the various types of objects that can be placed in the city.

The user starts with the screen in Figure 3.5. There is a big blue background with a white grid on top of it where objects can be placed. The grid's cells are where the objects can be placed, making it easier to distinguish each objects possible position.

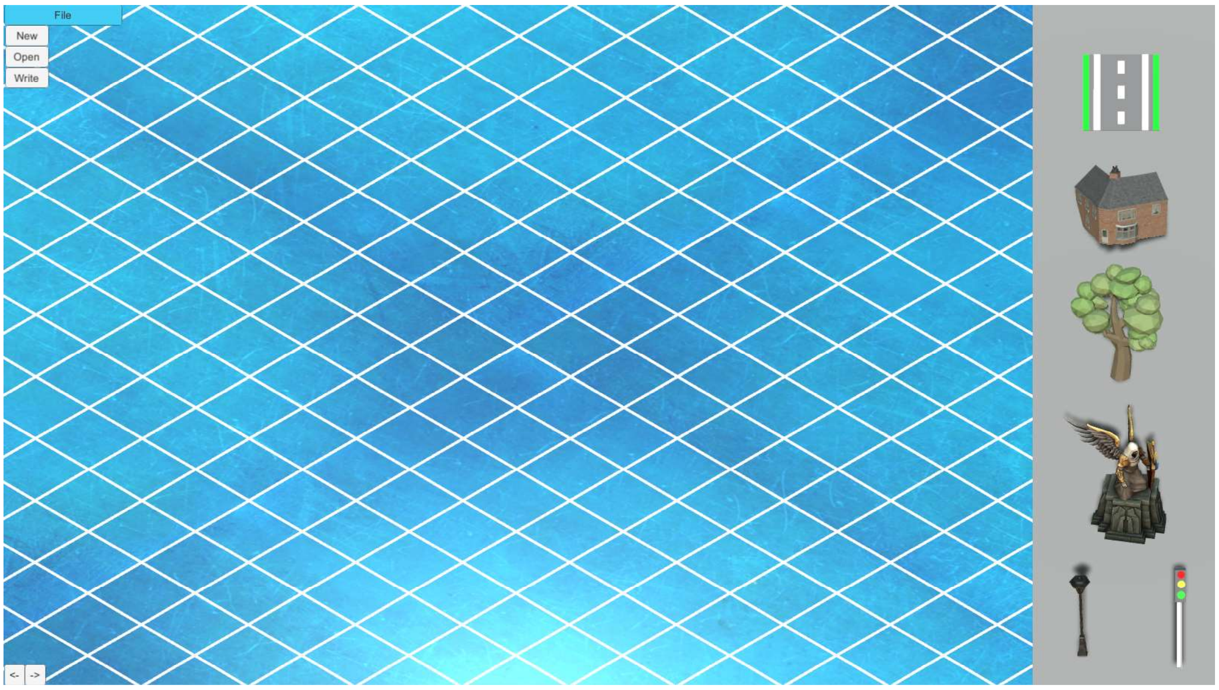


Figure 3.5: Modelling starting screen

On the top left of the monitor there is a lighter blue button with the name ”File”. If this button is pressed, three new buttons will appear under it.

The ”New” option cleans the whole grid, removing all the objects in it.

”Open” makes a new window pop up and asks the user to select a XML file exported previously by the modelling program to import it to the new grid.
Finally, ”Write” pops a window so the user can select the destination and name of the XML file that is going to be created with the objects in the current grid.
On the left lower corner of the screen we can see two buttons with characters representing an arrow pointing to the left and another pointing to the right . Each button makes the camera turn either ninety degrees or negative ninety degrees, depending on the button, with the center of the screen as the middle stationary point.
On the right part of the screen there is the list of objects that can be placed on the grid itself.
When modelling the user has to first select the desired object that wants to be placed in, after the selection a small highlight will appear on the selected object on the list on the right, so the user knows which object is selected at the certain time.
After the object is selected, and by consequence, highlighted, the user can now try and place it on the desired grid’s cell. Before placing the object in one of the cell’s grids there is another highlight feedback on the grid itself so the user is sure where he or she will place the selected object.
The list of objects include Roads, houses, trees, statues, street lamps and traffic lights.
When accessing the object’s properties a new options menu will appear in the bottom middle of the screen (Figure 3.6). The selected object has a blue tint to it so it is faster for the user to recognise which object is going to be manipulated. All of them have the red button ”Del” on the top left corner to delete the object if the user desires.

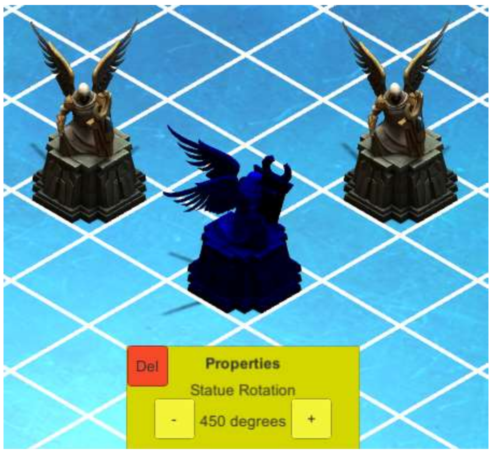
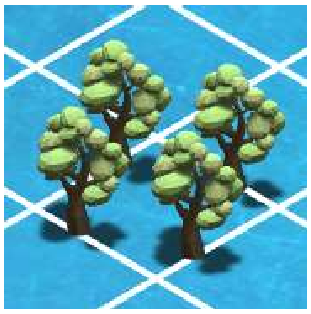


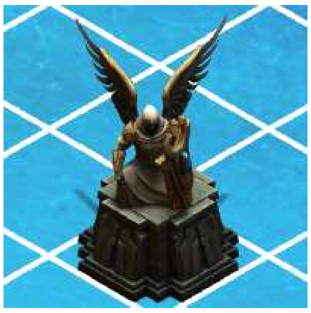
Figure 3.6: Properties from the middle statue which is the one being selected

Objects on the grid

The model of the houses 3.7a were picked with an L shape because it is easier to understand the angle of which the user is seeing if he or she decides to rotate the grid. All the sides to the house are different so each angle is distinct.
A Tree object with the ”number of trees” property changed to four on the grid figure 3.7b.
Statue on the grid 3.7c is another aesthetic object chosen from an online library. We wanted the statue

(a) House


(b) Trees


(c) Statue

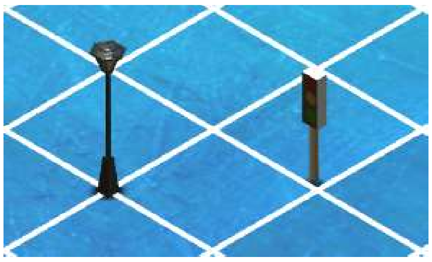

(d) Lamp and Traffic light

Figure 3.7: Example of objects on the Modelling grid

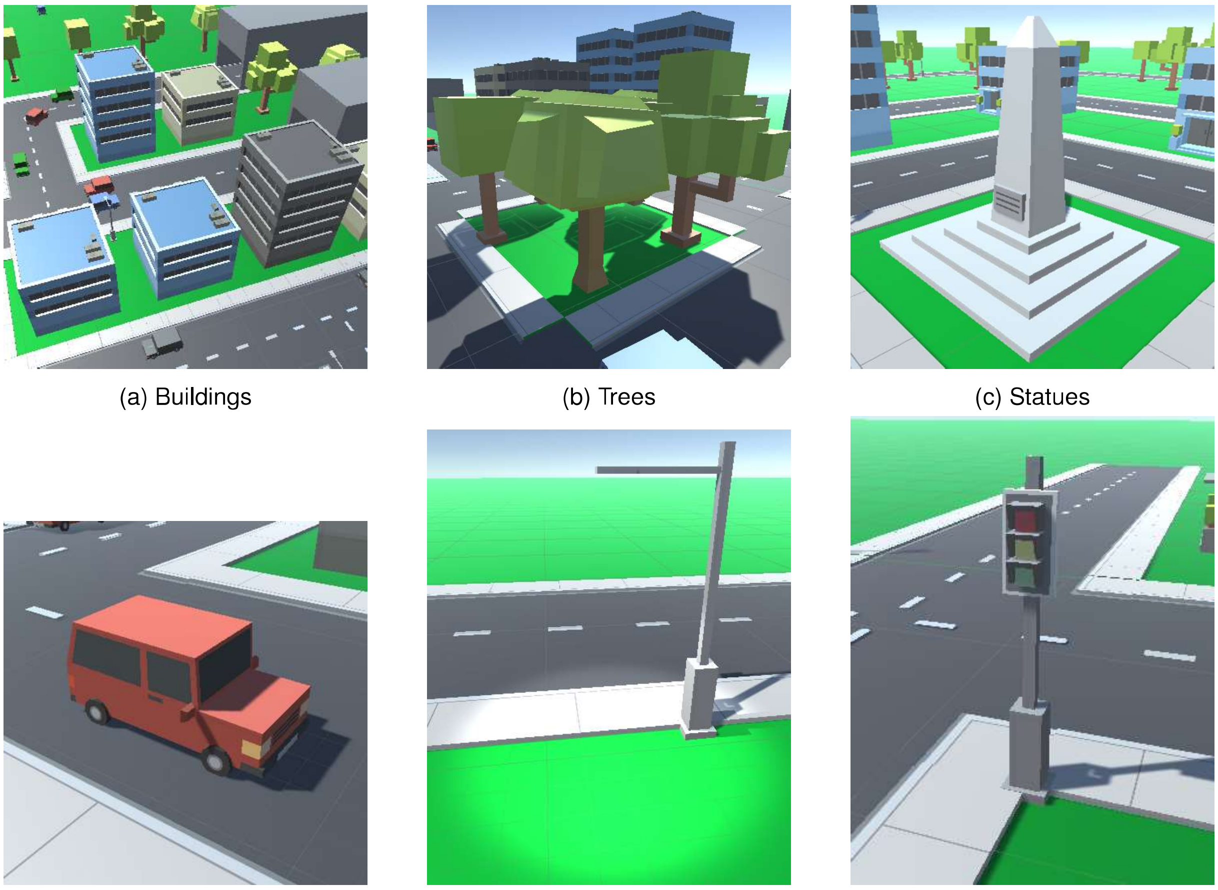
to be asymmetrical, similar to the house model, so the object is different from each angle that is approached.

Examples of Street lamps and Traffic light objects on the grid, figure 3.7d.

Every object was either bought from the unity asset store or downloaded for free in Blender or in public online libraries.

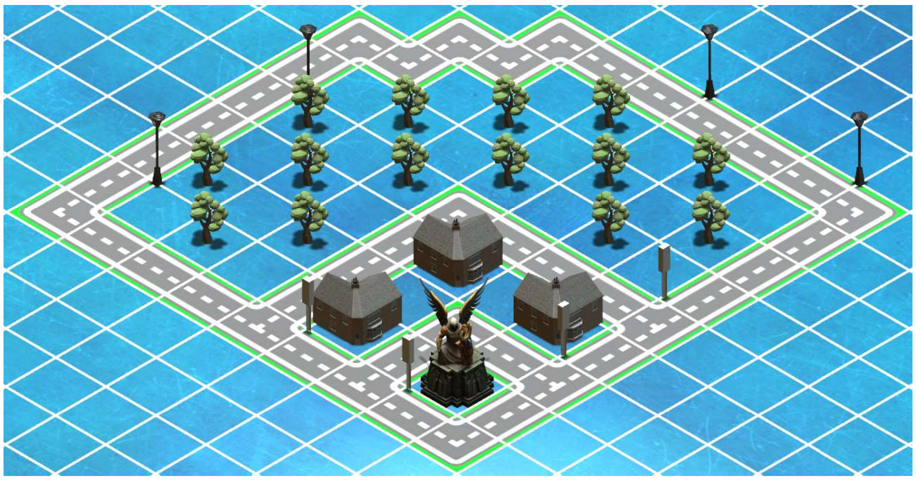
3.2.2 Visualisation

The user starts with a blank environment and a single button on the up left side of the screen.This button opens an operating system's windows to let the user choose which XML he or she wants to open. After choosing the modelled city file the corresponding city is automatically built.
The art for the city is a low poly type of style, which means objects do not have much three dimension detail but the simplicity makes the city a less confusing and overwhelming one, and by consequence, a welcoming feeling.
The roads are similar to the modelling program ones. They can appear in four formats: Single, turn, T crossroad, and '+' crossroad. With the XML file the VR program knows how to rotate the road element so it sticks together with the adjacent ones.
There are three type of buildings (Figure 3.8a) where each one has a unique colour: blue, beige or black. Then each building has three different levels of floors ranging from 1 to 3.
The trees have a simple brown log with three different types of leaves (Figure 3.8b).
Statues are also an aesthetic aspect of the environment (Figure 3.8c).
The dynamic objects in this world will be cars (Figure 3.8d), simple four wheeled objects that move around the environment. Their movement is controlled mainly by the communication with the students code received over UDP. Although in the tests that will be referenced in the next chapter, these cars are controlled manually. By manually, it means each car has its own script for the specific route or test run.
For more artistic objects, Street lamps (Figure 3.8e), can be placed in the city and can illuminate the nearby roads, since each lamp has a source, cone shaped, light pointing towards the ground.
Lastly, traffic lights (Figure 3.8f), also get information through the same system as the cars over UDP from students making it switch its state from red light to green light to yellow light and repeating.
After the simulation reads the previously exported XML file created with the modelled city information, users enter this world with the Virtual Reality glasses.
The virtual city can now be seen, which previously, when it was being built, was seen in two dimensions on a computer screen, in three dimensions (Figure 3.9).

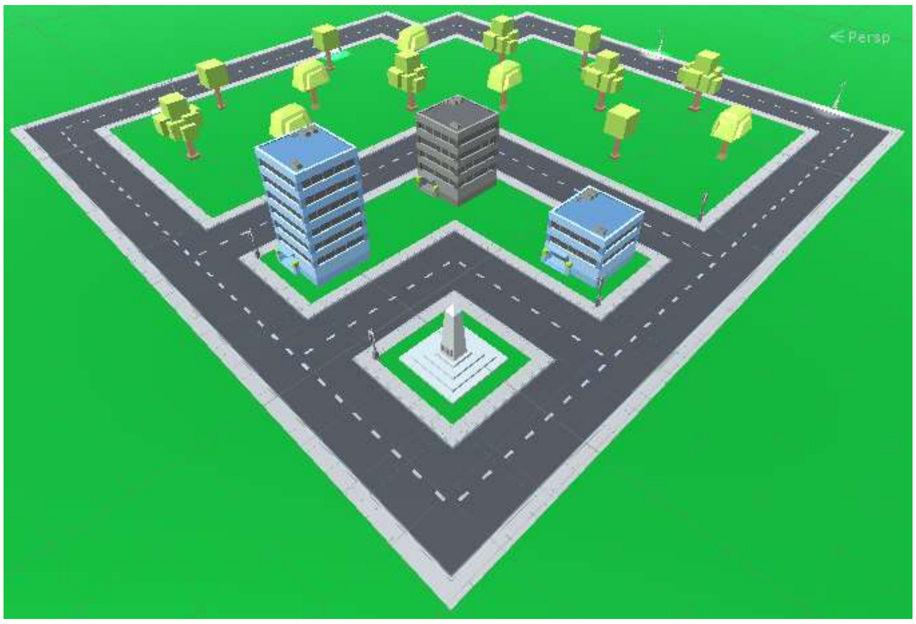

(a) Buildings (b) Trees (c) Statues

(d) Car (e) Street Lamp (f) Traffic Light

Figure 3.8: Objects in Virtual Reality



(a) Modelling



(b) Virtual Reality

Figure 3.9: Example of a city made while modelling and then being seen in SmartMobVR

Chapter 4

SmartMob's Validation

4.1 Modelling

While developing the Modelling aspect of the application, integration tests were constantly being made. For every new feature, tests were made to see if it was doing what is was expected to do. Throughout the semester some iterations were done with users, getting feedback.
Making sure all the objects were well put on the grid; the road's type (straight, turn or intersection) and rotation that was automatically set from the moment it was placed; which object was being selected and edited; the grid's movement and rotation; Cleaning, opening and writing the xml file; these were all features that were constantly being tested to see if worked as expected.
Some usability testing was also made throughout the semester while developing the program, with it, the following changes were made:

- In the beginning, the program had a green background, trying to mimic grass, but switching to a blue background was the best option since it is more soothing to the human eye and it gives the impression that the modelling is being done on a city's blueprint.
- When selecting the object to then put it on the grid, a small icon would follow the mouse but this was substituted by the highlight in objects since it clusters the screen less and gets a faster feedback.
- The feature to zoom in and out, rotate left and right was also something learned with usability testing. It was usually the first thing they would do when opening the program, and felt stuck when not having the option to.
- Finally, the "new" button was requested due to the fact that sometimes it was easier to clean everything up with a button than to select each object and delete one by one.

At the end of all the iterations users are able to build cities such as the one we can see in figure 4.1.
After exporting the city, the XML file is easy to read even for the human eye, in case there are some errors with the file itself. We can see (figure 4.2) objects with different identifications, positions, rotation and properties. Each property number works differently depending on the given object.

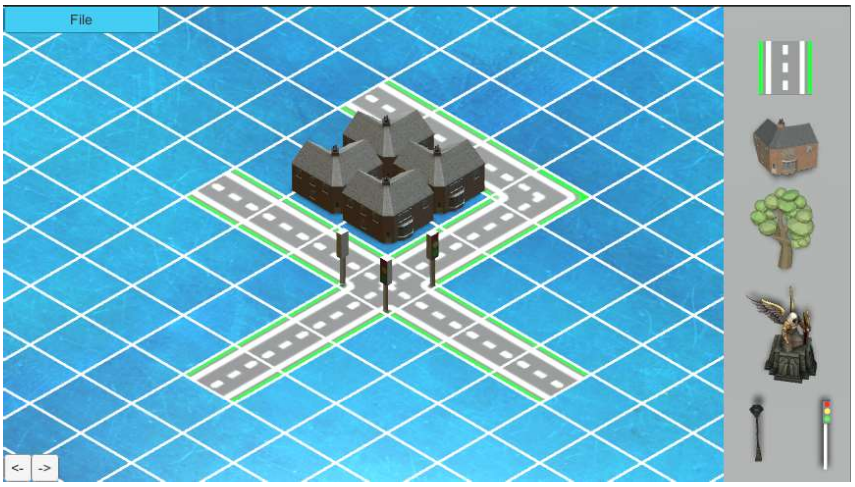
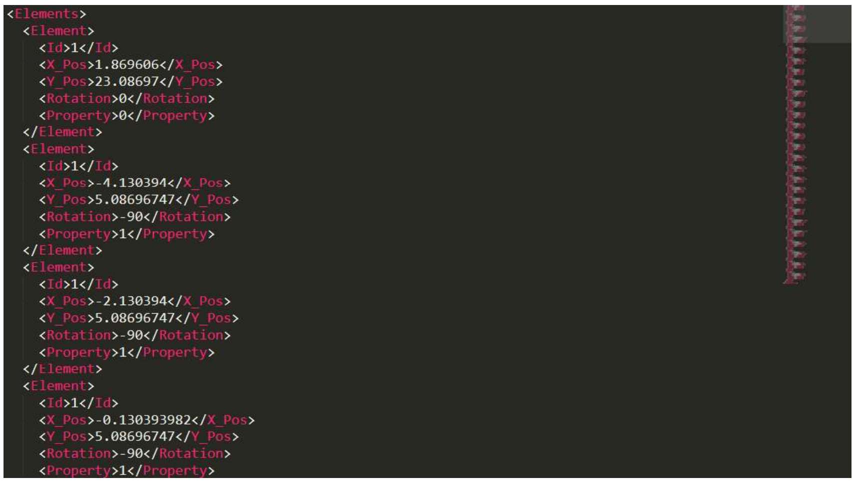


Figure 4.1: Example of a city using Modelling



```
<Elements>
  <Element>
    <Id>1</Id>
    <X_Pos>1.869606</X_Pos>
    <Y_Pos>23.08697</Y_Pos>
    <Rotation>0</Rotation>
    <Property>0</Property>
  </Element>
  <Element>
    <Id>1</Id>
    <X_Pos>-4.130394</X_Pos>
    <Y_Pos>5.08696747</Y_Pos>
    <Rotation>-90</Rotation>
    <Property>1</Property>
  </Element>
  <Element>
    <Id>1</Id>
    <X_Pos>-2.130394</X_Pos>
    <Y_Pos>5.08696747</Y_Pos>
    <Rotation>-90</Rotation>
    <Property>1</Property>
  </Element>
  <Element>
    <Id>1</Id>
    <X_Pos>-0.130393982</X_Pos>
    <Y_Pos>5.08696747</Y_Pos>
    <Rotation>-90</Rotation>
    <Property>1</Property>
```

Figure 4.2: Example of a XML file

4.2 Visualisation

When validating the Visualisation, integration tests were made to check if the application could read, from the previously exported XML file, and place every object correctly in the virtual world. The first models that were used, tried to be realistic but this required computational performance that wasn't needed and it broke immersion since the models were not real enough. Because of this, the pack, Simple Town - Cartoon Assets[1], was bought on the Unity Asset store and its models were used for the cars, roads, traffic lights, houses and statues. After importing the XML files exported in the previous program, the users can see the city in virtual reality (figure 4.3) move around freely and select cars that move in the environment.

[1]S. Studios. Simple Town - Cartoon Assets from the Unity Asset Store made by S. Studios in 2015. https://assetstore.unity.com/packages/3d/environments/urban/simple-town-cartoon-assets-43500

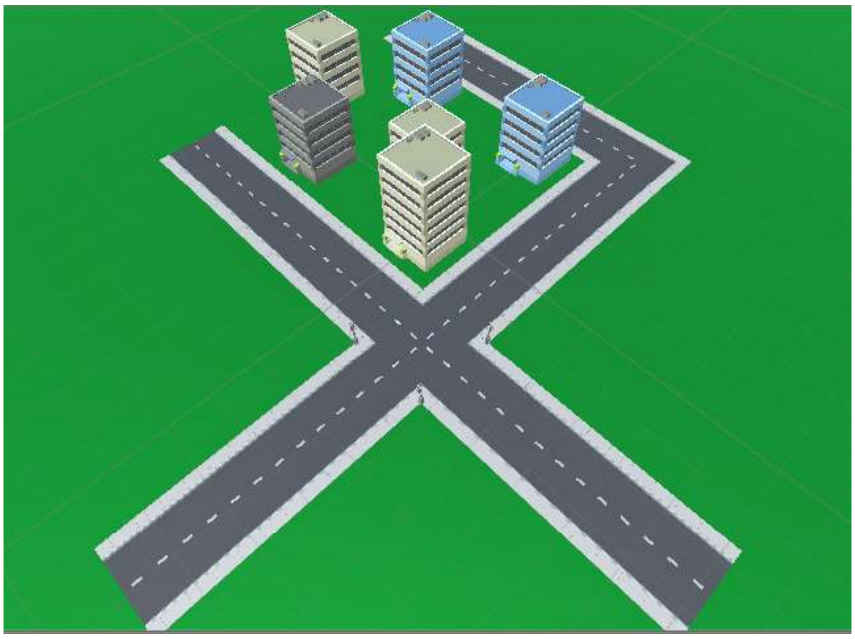
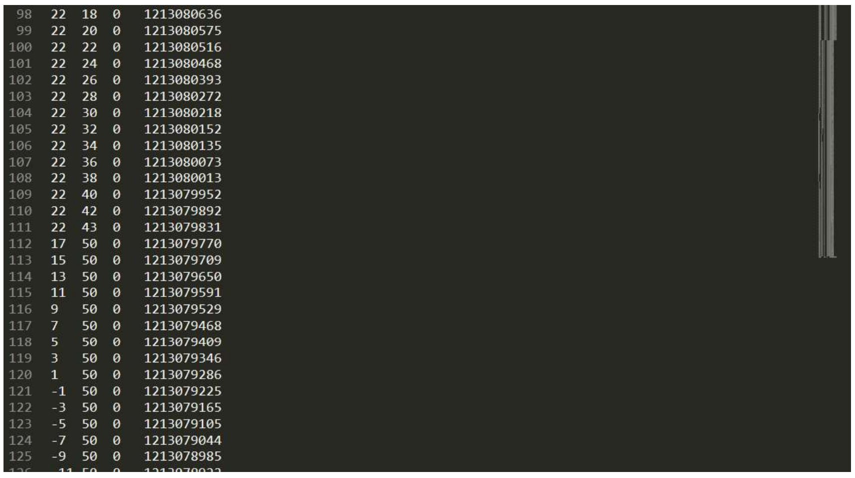


Figure 4.3: Example of a city in virtual reality

4.3 Simulation

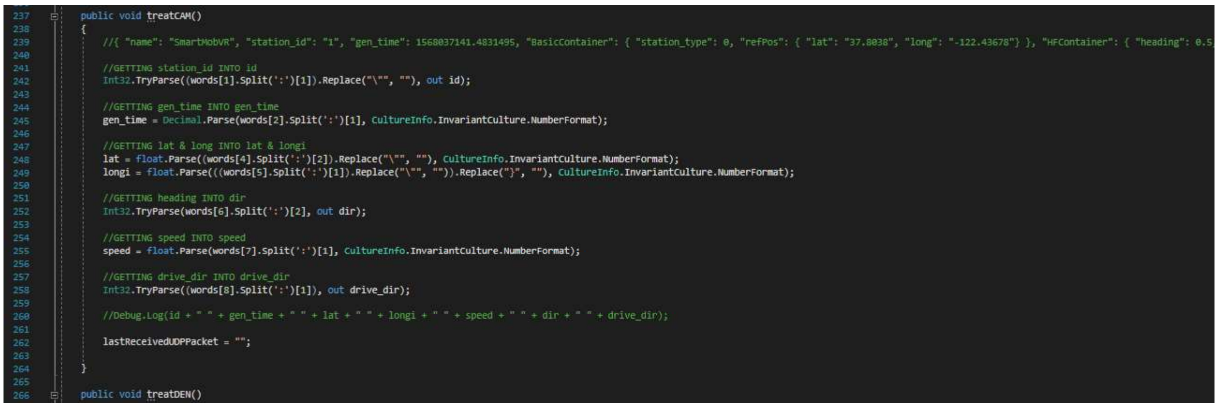
To verify the integrity of the simulation, a set of scripts in python would send messages, through UDP, to the Visualisation's program. These scripts have three examples of coordinates to help and test the simulation. This coordinates could be read either simultaneously, through three servers running at the same time, or at different times, having only one car move around the city at once. Integration tests consisted in making sure the car would run the set course established by the text file with the coordinates (figure 4.4. These tests were particularly useful when looking at how the car would make turns. On the first iterations the car would turn 90º or 180º instantly to the desired direction but in order to make it look more natural and realistic, it was added a way for it to turn, giving two distinct coordinates.



```
22  18  0   1213080636
22  20  0   1213080575
22  22  0   1213080516
22  24  0   1213080468
22  26  0   1213080393
22  28  0   1213080272
22  30  0   1213080218
22  32  0   1213080152
22  34  0   1213080135
22  36  0   1213080073
22  38  0   1213080013
22  40  0   1213079952
22  42  0   1213079892
22  43  0   1213079831
17  50  0   1213079770
15  50  0   1213079709
13  50  0   1213079650
11  50  0   1213079591
9   50  0   1213079529
7   50  0   1213079468
5   50  0   1213079409
3   50  0   1213079346
1   50  0   1213079286
-1  50  0   1213079225
-3  50  0   1213079165
-5  50  0   1213079105
-7  50  0   1213079044
-9  50  0   1213078985
```

Figure 4.4: Example of a pair of coordinates the car sends the server and is received by the visualisation

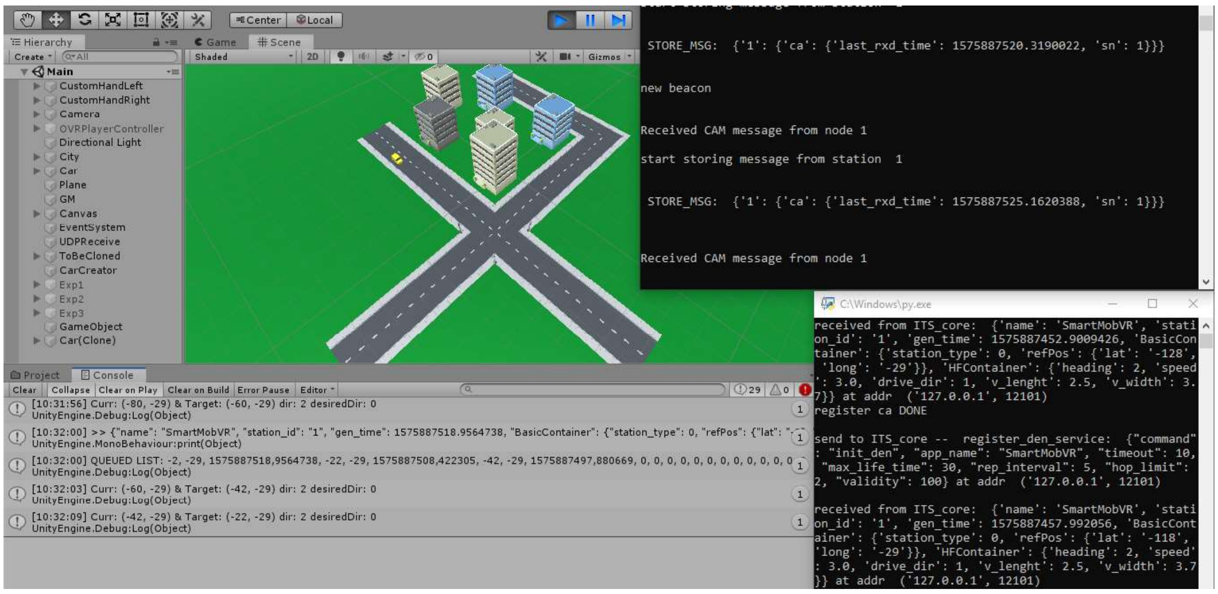
The majority of tests run were done with running servers and clients locally. The client had a python script similar to the ones implemented in the laboratory cars. This client would send messages with information about the current set of coordinates according to the time that was associated with it. The server would then receive the packages and show it in the terminal. To integrate this communication to the visualisation aspect of the SmartMobVR, another set of messages was sent to the program also using UDP. The messages were sent equally to the ones sent to the server. In the visualisation the message is received and processed (figure 4.5) getting each field from the message and then utilising it for the car's movement.



```
public void treatCAM()
{
    //{ "name": "SmartMobVR", "station_id": "1", "gen_time": 1568037141.4831495, "BasicContainer": { "station_type": 0, "refPos": { "lat": "37.8038", "long": "-122.43678"} }, "HFContainer": { "heading": 0.5

    //GETTING station_id INTO id
    Int32.TryParse((words[1].Split(':')[1]).Replace("\"", ""), out id);

    //GETTING gen_time INTO gen_time
    gen_time = Decimal.Parse(words[2].Split(':')[1], CultureInfo.InvariantCulture.NumberFormat);

    //GETTING lat & long INTO lat & longi
    lat = float.Parse((words[4].Split(':')[2]).Replace("\"", ""), CultureInfo.InvariantCulture.NumberFormat);
    longi = float.Parse(((words[5].Split(':')[1]).Replace("\"", "")).Replace("}", ""), CultureInfo.InvariantCulture.NumberFormat);

    //GETTING heading INTO dir
    Int32.TryParse(words[6].Split(':')[2], out dir);

    //GETTING speed INTO speed
    speed = float.Parse(words[7].Split(':')[1], CultureInfo.InvariantCulture.NumberFormat);

    //GETTING drive_dir INTO drive_dir
    Int32.TryParse((words[8].Split(':')[1]), out drive_dir);

    //Debug.Log(id + " " + gen_time + " " + lat + " " + longi + " " + speed + " " + dir + " " + drive_dir);

    lastReceivedUDPPacket = "";

}

public void treatDEN()
```

Figure 4.5: Script excerpt that treats CAM type messages

When receiving the information we can see in figure 4.6 the coordinates being received and processed. In Unity's console we can see the coordinates queue that saves pairs of coordinates and their time (to calculate the velocity and desired position). Next the visualisation uses the current position (Curr) and the next position from the queue (Target). The car has a direction (dir) and desired direction (desiredDir), which is the new direction the car needs to be faced to when moving to the new position, if the car is doing a turn.



```
 STORE_MSG:  {'1': {'ca': {'last_rxd_time': 1575887520.3190022, 'sn': 1}}}

new beacon

Received CAM message from node 1

start storing message from station  1

 STORE_MSG:  {'1': {'ca': {'last_rxd_time': 1575887525.1620388, 'sn': 1}}}


Received CAM message from node 1
```

```
[10:31:56] Curr: (-80, -29) & Target: (-60, -29) dir: 2 desiredDir: 0
UnityEngine.Debug:Log(Object)
[10:32:00] >> {"name": "SmartMobVR", "station_id": "1", "gen_time": 1575887518.9564738, "BasicContainer": {"station_type": 0, "refPos": {"lat": "-
UnityEngine.MonoBehaviour:print(Object)
[10:32:00] QUEUED LIST: -2, -29, 1575887518,9564738, -22, -29, 1575887508,422305, -42, -29, 1575887497,880669, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
UnityEngine.Debug:Log(Object)
[10:32:03] Curr: (-60, -29) & Target: (-42, -29) dir: 2 desiredDir: 0
UnityEngine.Debug:Log(Object)
[10:32:09] Curr: (-42, -29) & Target: (-22, -29) dir: 2 desiredDir: 0
UnityEngine.Debug:Log(Object)
```

```
received from ITS_core:  {'name': 'SmartMobVR', 'stati
on_id': '1', 'gen_time': 1575887452.9009426, 'BasicCon
tainer': {'station_type': 0, 'refPos': {'lat': '-128',
 'long': '-29'}}, 'HFContainer': {'heading': 2, 'speed
': 3.0, 'drive_dir': 1, 'v_lenght': 2.5, 'v_width': 3.
7}} at addr  ('127.0.0.1', 12101)
register ca DONE

send to ITS_core --  register_den_service:  {"command"
: "init_den", "app_name": "SmartMobVR", "timeout": 10,
 "max_life_time": 30, "rep_interval": 5, "hop_limit":
2, "validity": 100} at addr  ('127.0.0.1', 12101)

received from ITS_core:  {'name': 'SmartMobVR', 'stati
on_id': '1', 'gen_time': 1575887457.992056, 'BasicCont
ainer': {'station_type': 0, 'refPos': {'lat': '-118',
'long': '-29'}}, 'HFContainer': {'heading': 2, 'speed'
: 3.0, 'drive_dir': 1, 'v_lenght': 2.5, 'v_width': 3.7
}} at addr  ('127.0.0.1', 12101)
```

Figure 4.6: Example of the simulation receiving messages from a communication between the client and server

When handling with the cars from the laboratory, first, to test their communication with SmartMob's VR

program, both devices had to be connected to an ad-hoc network that is created everytime the car's raspberry pi is turned on. To test if the simulation worked as expected, a script coded in python is run within the car that controls the car's motors, making it able to move forward or backward and turning left or right. Integration testing could not be done with students from the Redes Veiculares course, mainly, because the schedules were incompatible. So objective tests were made to see if the end result achieved the expected.
In the current scenario, and unlike the expected, the cars were not equipped with GPS device, so they could not send messages with latitude and longitude. An estimate had to be made, assuming their speed is constant (measured previously with a track in the laboratory), each time interval (set as a fourth of a second) the car will be at position: $speed * timeelapsed$.

Chapter 5

Selection technique's Evaluation or Assessment

5.1 Experiment Overview

5.1.1 What is going to be evaluated

Tests were run to, mainly, compare the selected selection techniques: SmartRay, ShadowCone and IntenSelect, in order to see which one would have the best results when it comes to selecting a dynamic object in a dynamic virtual world, having three different objective metrics.

5.1.2 Test's setup/Environment

The tests were made in room 217, in an INESC-ID's laboratory, next to the Instituto Superior Técnico's facility.
The space was equipped with: a laptop, where users could answer the separate questionnaire about themselves and, after the tests, to give feedback about the tests made, regarding the scenarios, different techniques and movement in the virtual world; a desktop computer with more recent hardware, where the Virtual Reality tests were made and recorded; an Occulus Rift with two controllers and its sensors, that were used to see the virtual world, move in it and interact with.

5.1.3 Methodology

The tests were performed by randomly picked users individually, each one doing three different scenario, for each of the three different techniques. The order of the tests were as follows:

1. Presentation of the project, it's goals and an overview of the techniques at cause
2. Getting the user's consent
3. Users answering a simple questionnaire about their own information

4. Starting the experimentation
5. Users giving feedback on the completed task
6. Presenting the users a small game to play as reward for taking the experiment

To gather information about the tests, photos and videos were taken and recorded with the help of a smartphone's camera, registering the computer's main monitor (using Open Broadcaster Software, a third-party program) to get what the user could see throughout the whole experiment and saving raw numbers by Unity's scripts, where the tests were being run.

5.1.4 Tasks

The users were presented with the following task:

- A yellow car will appear three times in three different locations at different times in each scenario. Select only the yellow car between the other coloured cars (can be red, blue, green or black) for each of the three different scenarios with each of the different selection techniques.

5.1.5 Tests

The test starts with a brief explanation of what SmartMob is, so the user knows the environment in which all of this fits, the task the he or she is about to do, how different technique works and the rules on how the tests work. The rules consist on letting the user know: it is allowed to move around the virtual city at any given time in the experiment, the same yellow car may not be selected more than once (this is to ensure some consistency on the results and metrics that were evaluated) and when using ShadowCone, the yellow car is only considerate selected when only it is selected and no other car with it. After the first step is done, the user reads the consent, giving a small introduction, a purpose, procedure and confidentiality of the experiment. To get information about each person, building our demographics, users answered a questionnaire on a given laptop, using google forms. Before each scenario the user is reminded how the technique, that is about to be used, works. This is done so there is the best possible understanding of the technique and, consequentially, better use of it and most accurate results.
For each technique, all of the three scenarios are played separately and individually, meaning they all have a start and an end, being executed sequentially. Every scenario starts without any cars but slowly they appear one after the other coming out from rectangular facilities representing garages. The yellow car has a fixed timer for when it appears in the current scene, this way the consistency on the tests is preserved. After the yellow cars appears three times, then the application is restarted but this time with the next scenario loaded until all three of them are over.
So the experiment's sequence can be seen as:

1. Flashlight
   (a) Scenario 1

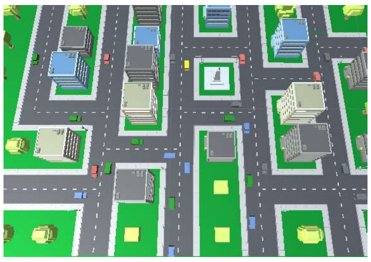
   (b) Scenario 2
   (c) Scenario 3
2. SmartRay
   (a) Scenario 1
   (b) Scenario 2
   (c) Scenario 3
3. ShadowCone
   (a) Scenario 1
   (b) Scenario 2
   (c) Scenario 3
4. IntenSelect
   (a) Scenario 1
   (b) Scenario 2
   (c) Scenario 3

The first scenario is a simple roundabout with trees in the middle and some buildings on the sides. Cars in this scenario move counter clock wise and usually take the second exit, except the targeted yellow car (Figure 5.1a), making it the simplest scenario, user can get used to the experiment before going to more complex maps.
The second is a small city with some buildings, a statue, trees, street lights and traffic lights (Figure 5.1b), this way users were encouraged to move around the city or find new viewpoints to better look at everything at once, making the environment more dynamic than a usual steady camera with a single viewpoint.
The last case is a mimic of a highway with four lanes. There's trees on each side and trees separating the two highways with different directions (Figure 5.1c). Cars move at different speeds, with higher speed the more to the left lane they are, this scenario is the most complex one because the point is to see how each technique works when the object that needs to be select is in a cluster and/or at a distant range.
For each scenario the yellow car appears three times, with different routes. This way we give a second and a third chance to the user to get used to the technique if needed.
The tests begin with SmartRay because the technique is easier to get familiarised with, this way users can get used to the environment without being too overwhelming with the technique they first encounter. Shadow Cone is the next technique to be tested, and the experiment ends with IntenSelect.

5.1.6 Metrics

For the previously stated experiments the objective metrics being saved were:

(a) Scenario 1 - Roundabout with cars moving around it


(b) Scenario 2 - City with buildings, trees, lamps and a statue

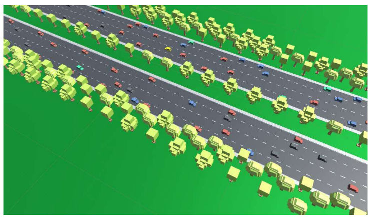

(c) Scenario 3 - Highway with two different directions separated by a row of trees

Figure 5.1: Scenarios

1. Technique's success rate
2. Time to select the car
3. Number of clicks
4. Hit rate
5. Heatmap for where the user was looking
6. Heatmap for where the user was pointing

The technique's success rate is calculated whether or not the user was able to select the targeted car before it could leave the scenario, for example, if the participant was able to select only 1 car out of 3 from a given scenario, the success rate for that scenario for the given technique is 33.3(3)%. This lets us know how hard or unfamiliar it was for the user to use and succeed with the given technique.
In each scenario the yellow car has three different courses. For each time the car appears, the participants can try and select the target as many times as they want to. If we take t as the number of tries to select the target on one of the three times and x as the time for each try, we can calculate its mean, mt as:

$$mt = (\sum_{n=1}^{t} x_n)/t$$

Because three yellow cars appear in each scenario we calculate the mean of all three means for the specific scenario, $mThree$. After getting the mean time for one scenario ($mThree$) and doing the same process for the other scenarios with different x's, we gather the three different means from each scenario and get the mean of the whole technique for the current participant, $mExp$:

$$mExp = (\sum_{j=1}^{3}(\sum_{i=1}^{3} mt_i)/3)/3$$

Time to select the car is calculated by doing a final mean with all the tests run using the different 15 $mExp$'s.

The number of clicks is a metric that shows how many times the user, in average, had to try to select the yellow car to finally select it.
It was also considered hit rate which is a different metric similar to success rate. Our previous one, just took into account if the yellow car was either selected or not at the end of its course. Now we can calculate it for each click. If it took a participant 4 tries to select the desired car before the end of its path, we'll now consider the success rate 25% instead of 100%.
For the subjective metrics:

1. How familiar is the technique
2. How easy were the different scenarios
3. How easy it was to move around the scene

With this questionnaire we can see what the users felt during the on going tests. Being familiar with the technique is important because it can show us, for example, if the user is actively thinking about how to act or if it's familiar, and consequently, and easier to use. The perception of how long it took also shows us the feelings of how hard the technique was or how hard they had to focus to complete the task.

5.2 Analysis

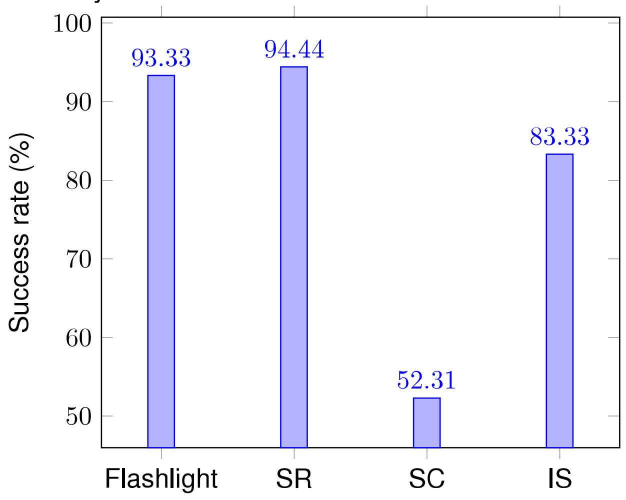
5.2.1 Demographics

A total of 14 people, half male and half female, with ages ranging from under 18 to 55. The majority of participants either had their masters finished or were currently studying to complete them. The remaining people were studying in high school or had completed their PhDs a few years ago. More than 70% played any type of games (mobile games, computer games or browser games) around an hour or two a day. Half of the remaining percentage played less than one hour a day or haven't played games in years, while the other half played from 7 to 8 hours a day. When asking about past Virtual Reality experiences it was included HmD, tablets and smartphones, making the percentage of people that didn't have any sort of VR experiences around 30%. When talking about Occulus Rift's experience, only 2 of the participants did have past experiences with it.

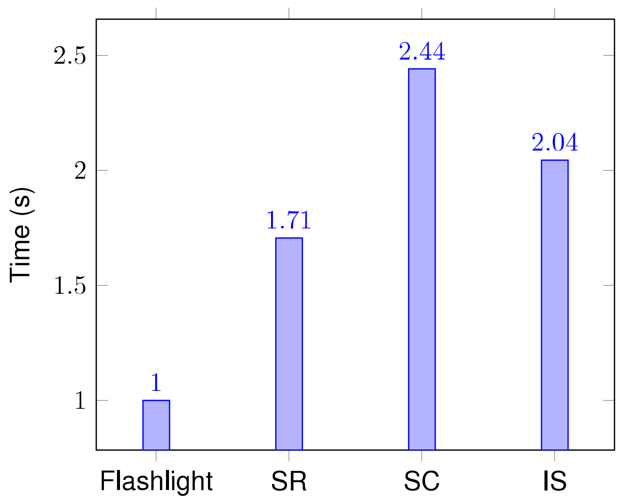
5.2.2 Results

The first objective metric is success rate. The yellow car appears three times per scenario, this means each technique will try and select 9 yellow cars at different times. Success rate is a percentage representing how many times the technique selected the targeted car out of the possible 9 times.
We can first see all success rates are high (above 80%) except ShadowCone. This was due the fact participants sometimes had a hard time selecting only the yellow car, meaning, the car wasn't usually the only one being selected before leaving the scene, considering then a miss and not a hit. Other times,

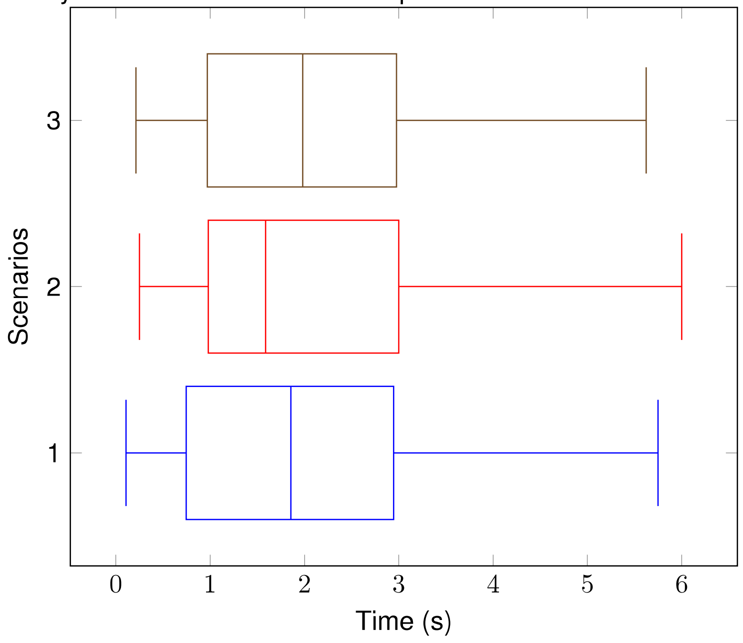
the participant's hand's jitter and/or some occlusion of building or other objects in the city would make users miss the yellow car even for a split second and had to retake the selection or miss completely.
Flashlight, SmartRay and IntenSelect had the best results considering success rate with no major differences. Participants would usually hit the yellow car, with some small exceptions, for example, another car obstructing the pretended one, losing vision for a while or even the remote would send not accurate information about its position because sensors are not accurate all the time.
IntenSelect could have a lower success rate because a few participants, when answering the post questionnaire, said the technique was too overwhelming sometimes. Having a cone and a ray may be too much sometimes. Not only this subjective aspect, but when analysing the calculations made in the program, when two cars were close enough to each other (because IntenSelect adds weight to every car inside the cone and the closer to the ray the heavier the weight is) and because of the remote's and participant's hand's jitter the non-yellow car could be selected instead.



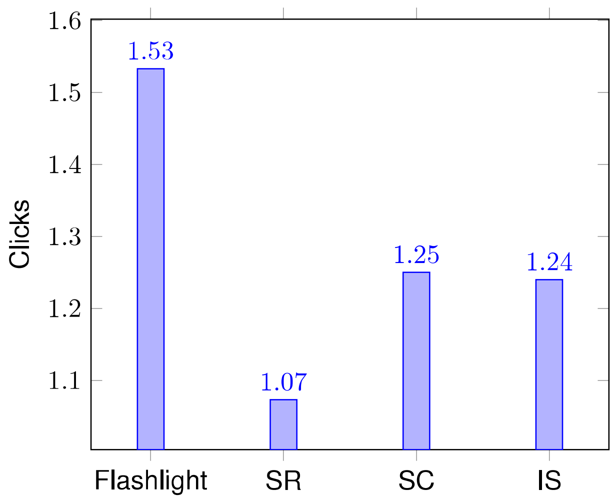
ShadowCone was the technique that had the higher amount of seconds to select the car since it's a technique that requires some time to refine the pretended objects, participants had to keep pointing at the car until all the other possible selected objects would leave the cone selection.
While SmartRay and IntenSelect had similar selection time, flashlight had a way lower selection time. This is due to the fact participants usually didn't hold the button to select when using flashlight unlike when using the other techniques. SmartRay and IntenSelect have a continuous refinement, meaning they can take some time to choose one car over the other. Flashlight on the other hand was used as an instant/no refinement selection technique. Participants usually pressed and released the selection button in under a second and try again if they missed.



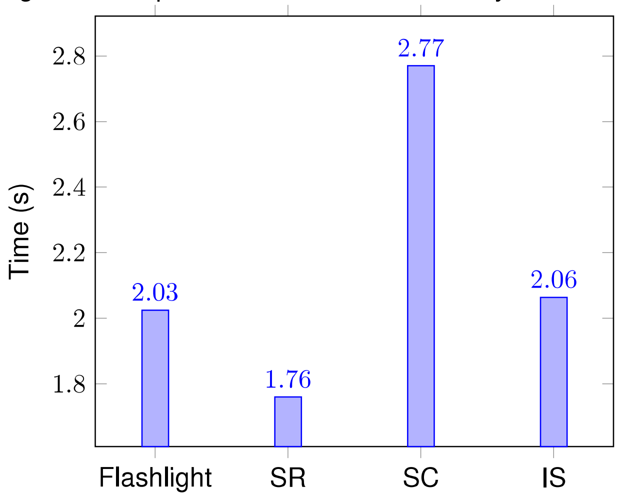
We can see all the scenarios had similar difficulties. The medians are somewhat close to each other. The first scenario has a higher median than the second because this scenario was the one where participants had their first experience with the selection techniques. The median goes back up a little in scenario 3 thanks to the complexity of the scenario. It required a higher time to refine objects from each other since they were very clustered and with similar speeds.



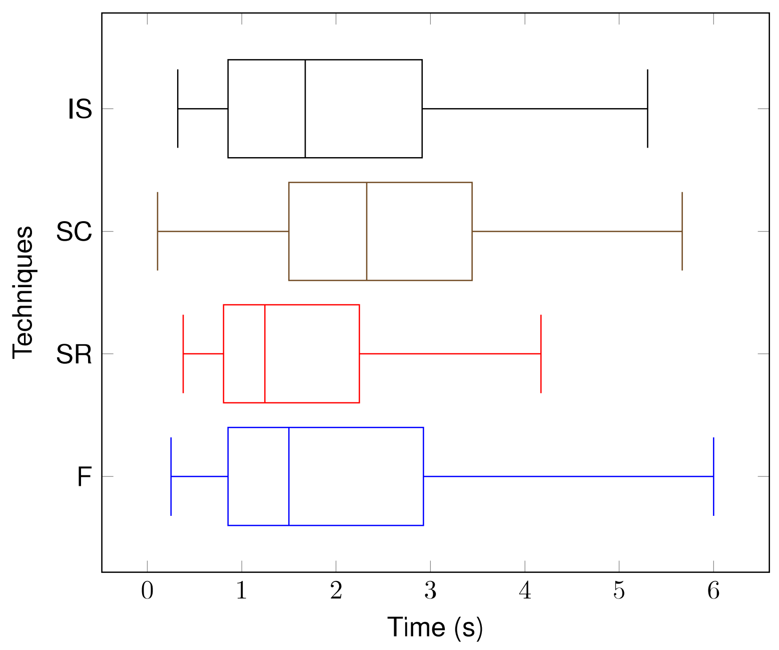
SmartRay had the least amount of clicks since participants could usually be able to select the car with one try. ShadowCone had a similar number of clicks because to select the car the time required was high meaning they had less opportunities to shoot the cone again.
This way we can see Flashlight had a higher amount of clicks than all the other techniques. Because users wouldn't refine their selection, there were many misses before actually hitting the desired object.



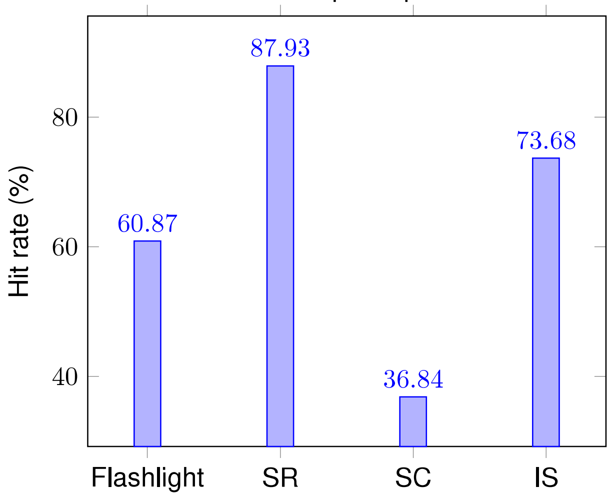
If now we say the time to select the object is the sum of every tries' time (instead of the tries' mean) we see the mean time usually rises, but even more significantly when looking at Flashlight. This data shows the average time a user takes to select the car from the moment he or she starts to try and select it. ShadowCone takes the longest to complete the task while SmartRay is the fastest.



Even though Flashlight had the second fastest median from all the techniques, it wasn't consistent. Flashlight was the technique with the highest time recorded, with 6 seconds, to select the wanted car. With this data, we can see SmartRay has the fastest scores, not only with its median, but also a smaller quartile compared to other techniques.

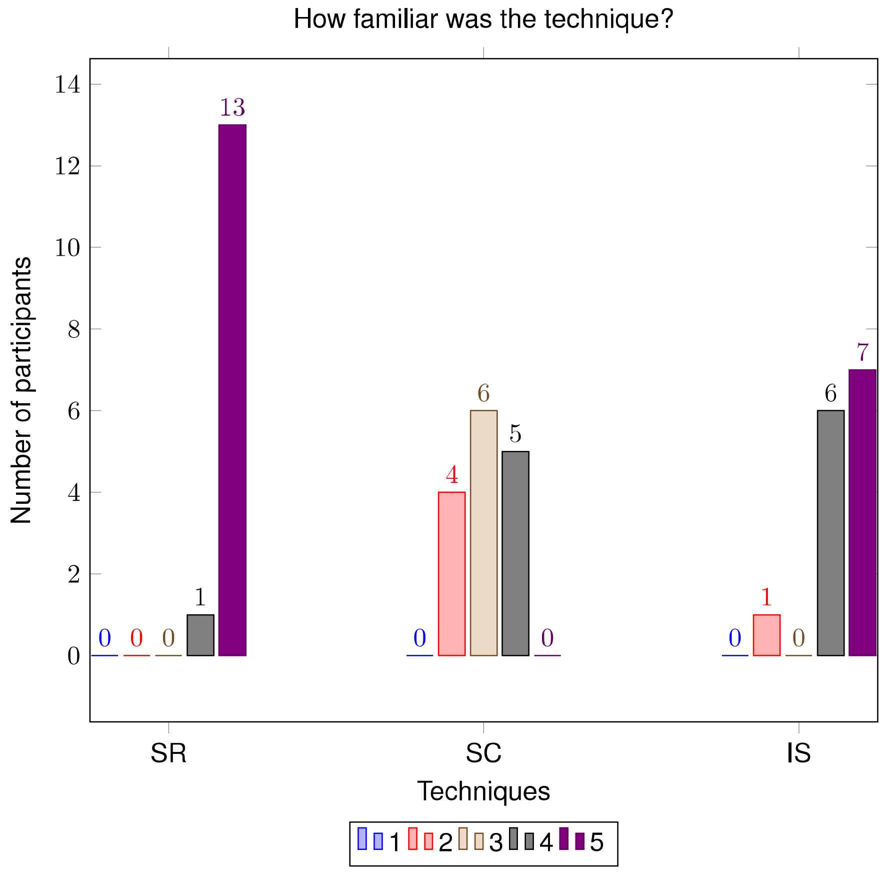


Finally, after getting data from the success rate and number of clicks, we see each technique's hit rate, which shows us the participant's difficulty or accuracy while using a technique. SmartRay has the best hit rate since less tries were needed to select the target. This is because participants usually followed the car for enough time to make its weight higher and higher compared to others. Users could hit other cars by mistake or lose sight for a while, but the yellow car's weight was high enough to still select it.
Now it can be seen that Flashlight falls down from the previous metric: success rate. This is simply because it is now taken into consideration the number of misses over the total amount of tries.
Lastly, ShadowCone has the lowest hit rate, not only because the number of tries was high but also because, before the yellow car could finish its course, participants couldn't select it in time.



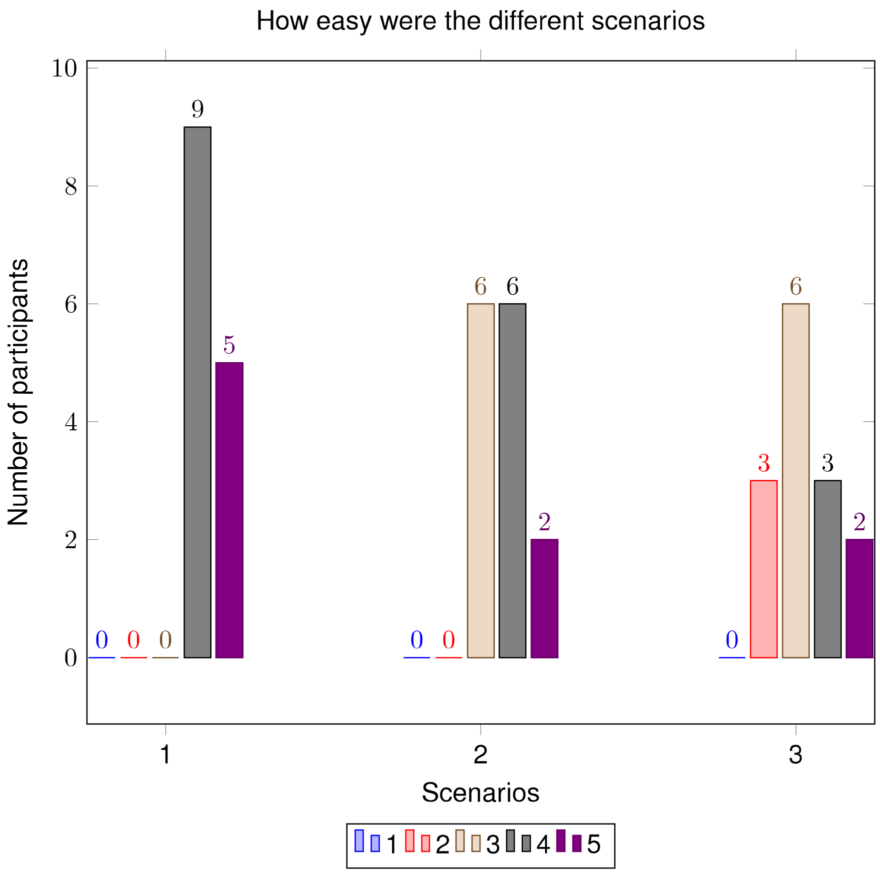
Participants found SmartRay and IntenSelect to be the easiest techniques to be familiarised with. Users found both of them to be easy to use, intuitive and with a precise selection method. The feedback on both techniques was easy to understand and learn.
For SmartRay, some participants said it looked more complicated to select a given object if there were more clustered environments because it looked easy to weight other objects that weren't suppose to. If it was hard to follow the targeted object, there would be a lower hit rate since users could select other

objects by mistake.
Unlike what was expected from the objective results we analysed or the ”familiarised” subjective metric, participants couldn't point out a disadvantage when talking about IntenSelect. Results show the time it took to select the car or tries to do it was higher than SmartRay but for users, in theory, the technique was better. It was easy, intuitive, easy to learn, had precision and a good feedback.
People in the experiment said ShadowCone was not easy to select objects in fast situations. They couldn't select only one car if they had to be fast with it. It was the least intuitive since they had to think about a lot of things at once, that it was very overwhelming because they needed to focus on which cars were currently selected and which they had to try and stop pointing at with the cone. Almost half asked if the cone could be adjusted, sometimes it was just easier to remove cars that were about to be selected if they narrowed the cone a little bit. Some users found annoying the fact that their progress of selecting the car could be lost in an instant if they flickered or lost track of the car for a bit and everything had to be done from the start. It wasn't a good technique for distant scenarios nor if objects had similar paths because it was hard to isolate the target from the rest.



Participants found the roundabout scenario to be the easiest. In this scenario users wouldn't usually move their heads around much as they could see the scenario clearly and fully while in the air looking down.
The city map had a lower rating because it was similar to the roundabout scenario but this time more cars were moving around and some buildings were obstructing certain roads. People would rather stay still and try to select than move to a position where the view of the yellow car was more favourable.
The highway scenario was the least easy one since the cluster was overwhelming and selecting distant

objects is harder.



Even though not many users moved around during the experiment, they all grade it a 4 or 5 out of 5 saying it was very easy to use and straight forward.

Chapter 6

Conclusions

In this chapter, it will be discussed what was achieved throughout the course of this work and what wasn't accomplished.

6.1 Achievements

The modelling and visualisation applications were developed successfully, according to their objectives and goals with some minor bugs. The modelling allows users to model cities, edit objects and export the modelled city.
The development and integration between Modelling and the Visualisation was also implemented successfully. The visualisation imports the cities correctly and allows users to move around the imported environment. The simulation communicates with the visualisation, with the current SmartMob's status. The visualisation represents cars according to the received information from the cars in the laboratory. In this, it is also possible to select objects using SmartRay.
SmartRay was the most successful technique in this experiment. Having the highest hit rate and lowest average amount of time when selecting the right object. It is a simple but effective technique with a small learning curve that outperforms the other techniques in the studied scenarios. IntenSelect had similar results to SmartRay, its weight algorithm and considering, not only the intersected object, but also objects inside the cone, was beneficial with some participants but a problem to others. That's why some users had better results with it. ShadowCone, was the technique with lowest hit rate and highest time but, mainly, due to being a technique that could select more than one object.
The experiment was also a success. We were able to test all four techniques with multiple different people with different ages, experience with games and education completed.

6.2 Future Work

Due to the lack of time and resources a validation of SmartMobVR with the course's students was not done in this project.

Information about cars is received by the simulation, if this project continues to be worked on, there could be a visualisation of that data in different ways for different objects, for example, cars and traffic lights.

Having new view ports or ways to control the cars, for example having a steering wheel and pedals that are connected to the computer and would control a given car in the city, could also be new features implemented to the project.

Cars in a city can be predictable and have limited areas where they move around. For future work and to better test the techniques in a dynamic environment, a flock of birds could be a scenario to test these on. Birds are unpredictable and have different speeds and turns in a three dimensional space, while cars are bound to the ground, which is two dimensional.

Bibliography

[1] J. M. B. dos Santos. Acoplamento de Proteínas em Ambiente Imersivo. Master's thesis, Instituto Superior Técnico, Lisboa, 2017.

[2] M. L. Pack. Visualization in transportation: Challenges and opportunities for everyone. *IEEE Computer Graphics and Applications*, 30(4):90–96, July 2010. ISSN 0272-1716. doi: 10.1109/MCG.2010.79.

[3] W. Chun, C. Ge, L. Yanyan, and M. Horne. Virtual-reality based integrated traffic simulation for urban planning. In *2008 International Conference on Computer Science and Software Engineering*, volume 2, pages 1137–1140, Dec 2008. doi: 10.1109/CSSE.2008.1074.

[4] C. Brenner and N. Haala. Fast production of virtual reality city models, 1998.

[5] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. Sumo - simulation of urban mobility: An overview. In *in SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 63–68, 2011.

[6] D. Medeiros, A. Sousa, A. Raposo, and J. Jorge. Magic carpet: Interaction fidelity for flying in vr. *IEEE transactions on visualization and computer graphics*, 2019.

[7] H. P. Wyss, R. Blach, and M. Bues. isith-intersection-based spatial interaction for two hands. In *3D User Interfaces (3DUI'06)*, pages 59–61. IEEE, 2006.

[8] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *ACM Symposium on User Interface Software and Technology*, pages 79–80. Citeseer, 1996.

[9] T. Grossman and R. Balakrishnan. The design and evaluation of selection techniques for 3d volumetric displays. In *Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 3–12. ACM, 2006.

[10] R. Stoakley, M. J. Conway, and R. Pausch. Virtual reality on a wim: interactive worlds in miniature. In *CHI*, volume 95, pages 265–272, 1995.

[11] M. Ortega. Hook: Heuristics for selecting 3d moving objects in dense target environments. In *2013 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 119–122. IEEE, 2013.

[12] T. Grossman and R. Balakrishnan. The bubble cursor: enhancing target acquisition by dynamic resizing of the cursor's activation area. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–290. ACM, 2005.

[13] R. Kopper, F. Bacim, and D. A. Bowman. Rapid and accurate 3d selection by progressive refinement. In *2011 IEEE Symposium on 3D User Interfaces (3DUI)*, pages 67–74. IEEE, 2011.

[14] A. Steed and C. Parker. 3d selection strategies for head tracked and non-head tracked operation of spatially immersive displays. In *8th International Immersive Projection Technology Workshop*, pages 13–14, 2004.

[15] G. De Haan, M. Koutek, and F. H. Post. Intenselect: Using dynamic object rating for assisting 3d object selection. In *IPT/EGVE*, pages 201–209. Citeseer, 2005.