UNIVERSITY OF TRENTO - Italy

Department of Information Engineering and Computer Science

Bachelor's Degree in
Computer Science

FINAL DISSERTATION

Multi-user virtual reality through mobile devices using Leap Motion

Supervisor

Niculae Sebe

Co-Supervisor

Fabio Poiesi

Student

Mirko Pani

Academic year 2016/2017

Contents

Abstract

Multi-user virtual reality environments allow multiple users to interact with each other by connecting to the same session via virtual reality (VR). These systems make it possible to apply the potential of VR to any type of application domain, from the computer games and entertainment industry to the manufacturing or design sector; pushing the limits of human experience. However, at present, systems of this kind require expensive VR hardware that is not accessible to many. In addition, these VR experiences often make use of remote controllers that limit the possibility for more immersive interactions and at the same time require some prior experience in order to be used. The purpose of this thesis is to study, implement and evaluate a VR setup that allows a multi-user VR experience at low cost, while being at the same time more easily accessible to everyone. The application is designed to be used on Android smartphones and is based on the Unity3D game engine. We use Google Cardboard as the head mounted display (HMD) for the VR and a Leap Motion for hand tracking, thus enabling gesture-based interactions. We have implemented a VR environment in which users can interact with each other or with mutable objects through the use of hands. We have also designed and implemented a scalable system for the transmission of a large amount of data between users, used for exchanging high-throughput visualisation data (e.g. hand joints) amongst participants. The developed system has been tested in different scenarios with up to seven users connected using real devices. The results obtained have demonstrated the feasibility of this type of application on mobile devices, although with significant limitations compared to high-end VR experiences.

1 Introduction

Virtual reality (VR) has long been recognised for its potential to revolutionize entire sectors [10]. Through the creation of immersive visual experiences, virtual reality provides experiences and interactions that are difficult or impossible to realize in reality. Many application domains already make use of immersive single user VR experiences; such as education, urban planning, simulations, computer games and medical applications. However, the diffusion of VR technology and the recent commercialisation of high quality VR hardware solutions suitable for the consumer market is increasingly leading to the creation of immersive multi-user environments [11]. In these systems, multiple users can interact with each other to achieve a common purpose, even at a geographical distance. Applications, such as education [9], manufacturing [6], engineering and construction [13], medics [16] and video gaming [39], have shown great interest in multi-user VR. Although it is becoming increasingly widespread, there are two significant limitations currently present in this kind of VR experience. The first limit concerns the high cost. Most of these systems require a high-end viewer such as HTC Vive [28] and Oculus Rift [37], in addition to a high-performance computer in order to work. Secondly, these experiences generally use physical controllers. These devices have the advantage of ensuring stability and ease of use for those who are already familiar with that type of device, but have strong limitations in terms of the naturalness of interactions [45, 17]. Moreover, a physical controller is difficult to use for those who have no previous experience. To achieve total immersion and to open up new opportunities in VR, technology developments are driving towards in-air hand tracking using devices such as Leap Motion [30]. Fig. 1.1 shows an example of collaborative virtual environment (CVE) where participants interact using hand gestures. Unfortunately, there are still several challenges that prevent an uncompromising experience using hand-free interactions in VR, such as transmission latency, high computational demand and hand tracking robustness [12, 21].

The work accomplished in this thesis aims to exploit the potential offered by VR multi-user expe-

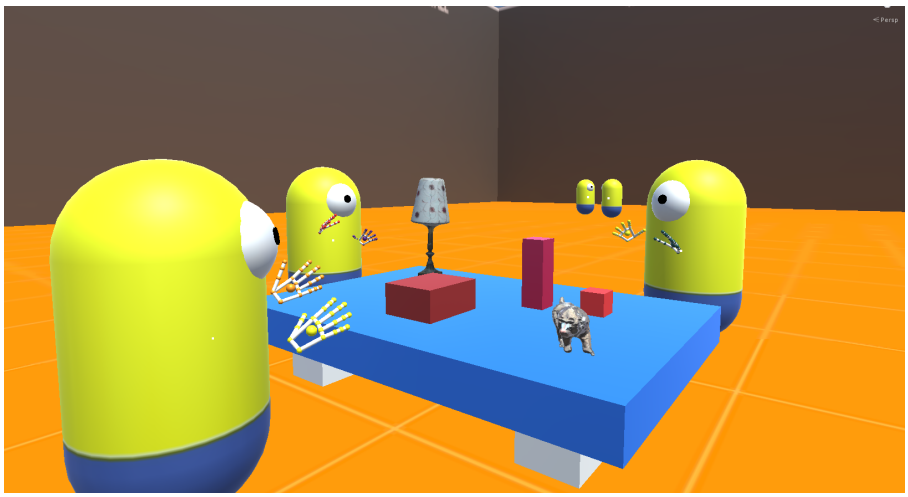


Figure 1.1: Collaborative virtual environment where two groups of participants are interacting via hand gestures.

riences, by trying to mitigate its stronger limits: cost and accessibility. Therefore, the purpose of this thesis is to study, implement and evaluate a prototype that allows a multi-user VR experience at low cost, while being at the same time more easily accessible to everyone. The application is designed to be used on Android smartphones: the wide diffusion and the possibility of using cheap VR headsets makes mobile devices an ideal platform for this purpose. In this regard we opted to use the Google Cardboard platform as the head mounted display (HMD), allowing a VR experience at a much more affordable cost compared to a high-end viewer. We use the Leap Motion device for hand tracking, in order to allow the user to interact with the world and other players through the use of hands. Figure 1.2 shows an user using his hands to interact with the prototype.

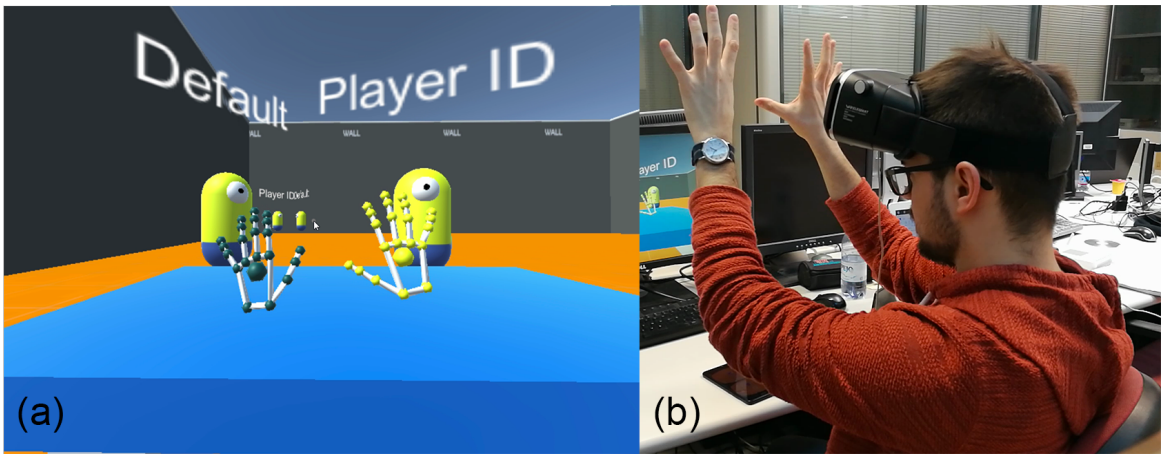


Figure 1.2: An user using the Leap Motion device mounted on a HMD to interact with the application.

The advantage of this setup is the low cost for the user and the ease of use of the application. In the realized prototype it is possible to interact with objects or other users within the same shared session. Many people, connected to the same local network or over the Internet, can play with each other by connecting to an already established game session or by hosting a new one. Users can interact with the simulated virtual world by picking, rotating or scaling mutable objects. Every change made to the game world is transmitted and synchronized between all users. In addition, users participating in the same session can also see virtual hands representative of other users' hands, in addition to their own hands.

Leap Motion does not yet support its direct connection to smartphones [21]. Therefore, we developed a system that can transmit information from a computer connected to the device to the mobile phone by using a server with Websocket protocol. The prototype has been implemented through the use of the Unity3D game engine [40]. The availability for a free license, the large number of features

and complete support for smartphones makes Unity3D an excellent platform to use to implement this type of application. Some technical challenges were faced in order to achieve the multi-user aspect: the use of devices such as smartphones, which have limited computational power, requires special attention to how data is processed and transmitted between multiple devices. One of the main challenges is the transmission of hand data reconstructed with Leap Motion. In fact, Leap Motion produces high-frame rate hand visualisation data that leads to high throughput when the hand visualisation data is exchanged over a network populated of clients [32, 21]. To mitigate the problem of high throughput, one can employ different data transmission strategies based on peer-to-peer or hybrid communications. Hybrid approaches use server and peer-to-peer communications interchangeably, and are typically employed in massively multiplayer online games (MMOGs) [5, 14]. One can also use an area-of-interest (AOI) based approach where a VR space is divided into zones and data are transmitted peer-to-peer amongst users that are located within the same zone [1].

In this thesis we implement a scalable mechanism to exchange data amongst clients in multi-user virtual environments:

- An ordinary client-server mechanism is used to update the states of mutable objects and to sync game-state information. We use a subset of Unity3D features called HLAPI [41] to implement this aspect.
- A distributed mechanism is used to handle high-throughput visualisation data exchanges. Differently from the AOI-based approach where the data transmission between clients is either active or inactive without accounting for levels of detail, we use clients' relative distances in the VR space to dynamically variate the resolution of transmitted visualisation data. We use a custom protocol using UDP sockets to exchange visualisation data.

The results and tests carried out have demonstrated the validity of the method and a decrease in measured network traffic and perceived latency.

The project was developed during the internship experience at the Bruno Kessler Foundation (FBK) in Trento, Italy [24]. We were responsible for implementing the whole application and designing the semi-distributed architecture under the supervision of Fabio Poiesi. Part of the work realised for this thesis is covered in a submitted paper entitled 'Distributed data exchange with Leap Motion' [19] for the "Salento AVR 2018 International Conference on Augmented Reality, Virtual Reality and Computer Graphics". The application has been tested in numerous scenarios on mobile devices. During the FBK 'Open Day' initiative, groups of two people were able to test the application by collaborating in the same virtual world. The demonstration involved several people and lasted more than an hour. The feedback received confirmed the ease of use of the application and the possibilities offered by possible future developments.

The thesis is structured as follows.

- **Chapter 2**: Provides an overview of the technologies and concepts used in this thesis.
- **Chapter 3**: Explains the features and the implementation of the prototype.
- **Chapter 4**: Describes and discusses evaluations and tests performed on the prototype. This chapter also presents a discussion about the limitations of the study.
- **Chapter 5**: Concludes the thesis and discusses possibile future research.

2 Devices and Technology

The purpose of this chapter is to introduce the reader to the basic notions necessary to better understand the work done in this thesis. Generic concepts such as VR and the devices that characterise it are presented. Moreover, we describe here the technologies used to implement the application: the Leap Motion tracking device and the Unity3D game engine. The approach used for transferring data related to the visual appearance of user's hands is also described.

2.1 Virtual reality

Virtual reality uses computer graphics to generate a 3D image or an environment with which it is generally possible to interact. A wide variety of devices are employed to simulate the physical presence of a user in a virtual world [3]. Images, sounds and possible physical representations of objects allow the user to look around, move and interact within a simulated world. VR technologies must therefore be as natural as possible in order to guarantee a good experience of use. Speed of reaction and accuracy to the user's interactions must be ensured so that no problems or discrepancies in the user's senses are caused. For example, a player's movement within the virtual world, if not matched by physical movement, can cause symptoms related to motion sickness [2]. Different types of sensors are used to create an experience that is as realistic and life-like as possible. Some are public and frequently used, others are still in the prototype or development stage. These devices can be divided into two categories: input and output. Input devices are used to record physical user interactions. Output devices, on the other hand, are a set of devices used to present information to one or more of the user's senses through the human perceptual system.

The most common and widespread VR devices are the head mounted displays, necessary to simulate an experience in VR.

2.1.1 Head mounted devices (HMD)

Virtual reality is generally experienced using devices mounted on the head, even if VR experiences created with rooms supplied with specific equipment exist [7]. They are the only fundamental device needed to access the world of VR, as they are necessary to present virtual images in the eyes of the user.

HMDs can be classified by their technical parameters:

- **Display**: The display can be produced with different technologies, such as LCD or OLED. In addition, each screen can have different resolutions and be equipped with a different pixel-per-inch (PPI) ratio. The higher the resolution and PPI ratio, the more immersive the experience will be.
- **Refresh rate**: Another feature related to the screen is the refresh rate, i.e. how long it takes for the screen to change its content. For a comforting VR experience, it is estimated that a minimum refresh rate of 60 Hz is required [8, 20], although there are currently displays that support a refresh rate of up to 120 Hz.
- **DOF Tracking**: Degrees of Freedom (DOF) refers to the movement of a rigid body inside a 3D space. Most HMDs only offer rotational tracking and therefore only 3 degrees of freedom: pitch, yaw and roll (3DOF). Higher performance HMDs allow positional tracking (6DOF), adding 3 axes of translation to the 3 axes of rotation.
- **Field of View**: The Field of View (FOV) is the extension of a world visible to the eye. Humans have a FOV of about 210 degrees for the horizontal arc and around 150 degrees for the vertical range. The current HMDs are unable to offer such FOV and generally stop at 120 degrees. A larger FOV allows for a better sense of immersion [15].

HMDs can also be divided into two main categories: wireless and wired. These two groups differ mainly due to difference in cost and technical characteristics.

Wireless HMDs are also called mobile-enabled HMDs. They include the set of head mounted devices that do not require a computer and therefore can be used without cables. Some of these can be simple containers for smartphones, while others are stand-alone devices with their own screen and processing unit. This type of viewer is the most cost-effective, at the expense of a limited experience compared to wired viewers. In fact, by using a mobile device as a processing unit, a mobile HMD has less processing power available if compared to a computer. However, the potential offered is nonetheless considerable: wireless, low-cost, and accessible VR experiences for everyone.

The HMD mobile market is currently characterised by three main platforms:

- **Google Cardboard**: Cardboard is a VR platform developed by Google in 2014. A user can buy a certified viewer or build one with simple materials following the technical specifications provided by Google [25]. The original headset uses simple cardboard and a pair of lenses to create a smartphone holder, which will then be used as a processing unit and screen. There are different versions available from different manufacturers, with a variable cost depending on the materials used.

  Compared to competitors, the Cardboard platform offers a more limited experience. The quality obtained from a VR experience on the Cardboard totally depends on the smartphone used, since there is no circuit in the viewer. Despite this, the Cardboard platform is currently the most popular VR ecosystem. The Cardboard platform can be used on any gyroscope-enabled smartphone equipped with Android or iOS operating systems.

- **Samsung Gear VR**: The samsung Gear VR is a VR headset for mobile devices designed by Samsung in collaboration with Oculus. The first model was released in 2015, although several revisions were released in the following years. Gear VR is only suitable for use on Samsung smartphones.

  Although a phone is still required to use Gear VR, there are significant differences from the Cardboard platform. While the phone is still used as the viewer's display and processor, the Gear VR also offers a custom rotational tracking unit called Inertial Measurement Unit (IMU). This unit enables more precise rotational tracking than sensors on smartphones. In addition, the Gear VR can also perform the function of controller, as it has a keypad. Gear VR is therefore equipped with its own electronic component, and in order to work it must be connected to the phone used via USB. The Gear VR software operates at a low level compared to the Cardboard SDK, enabling low persistence and less latency [38]. Gear VR therefore offers a more immersive experience than the Cardboard platform, at the cost of a much higher price.

- **Google DayDream**: Daydream is another VR platform developed by Google. It is the follow-up to the Cardboard platform and was announced in 2016. It was designed with the aim of offering a better VR experience than the one achieved with the predecessor. Unlike Cardboard, the Daydream platform has specific hardware and software requirements and only phones that meet them can be used with the viewer. These requirements include technical specifications deemed necessary by the platform to have good quality VR experience. Among other requirements, a minimum latency and resolution for the display is specified. Moreover, only smartphones with Android 7.0 or higher are supported, as a dedicated software mode is required for Daydream applications. In addition, the Daydream platform allows for standalone devices that do not require a phone as a processing unit [27]. The Daydream platform, unlike Cardboard and Gear VR, supports 6DOF tracking without the use of external devices. This feature allows VR experiences very similar to those allowed by wired viewers.

The second category of viewers refers to wired HMDs, generally used with computers or consoles. These types of viewers are considered to be high-end viewers as they allow a higher quality VR experience than a mobile viewer. These devices have their own integrated display which offers high resolution, a refresh rate of at least 90Hz and a FOV above 100 degrees. These wired HMDs generally

use USB and HDMI connections for transferring tracking and video signal information. The most popular viewers of this type are the HTC Vive and Oculus Rift.

Figure 2.1 illustrates some of the most commonly used HMD. Regardless of the type of HMD used, it is still necessary to have an input device to provide VR experiences in which the user can interact.

Figure 2.1: Virtual reality headsets. Starting at left from above: Google Cardboard, Samsung Gear VR, Google DayDream, HTC Vive, Oculus Rift, Playstation VR.

2.1.2 Input devices

In this section we intend to illustrate the different types and characteristics of devices capable of capturing user input. The most classic devices for this purpose are controllers. Controllers used for VR can be considered as an improvement to traditional controllers. In fact, they have the typical features of a normal controller, i.e. levers and buttons, in addition to 6DOF tracking, technology also offered in high-end HMDs. Each device has its own variants compared to the standard components of a controller. The Half Moon controllers [37], created by the Oculus group, have capacitive sensors capable of detecting finger movement. Another example could be the HTC Vive controller [28], which uses a touchpad rather than an analogue stick. In the mobile field, a controller [26] developed by Google as part of the DayDream project has recently been introduced. It has only 3 degrees of freedom and is therefore unable to recognize its position in space. However, its ergonomics and ease of use make it effective in many virtual applications. Even though the present controllers have forms and uses similar to traditional ones, some devices using concepts that lend themselves to a better VR experience are in development. For example, the Knuckles devices designed by Valve Corporation have a shape that allows the user to let go the controller without dropping it. While these devices are mostly suitable, they are obstructive and require some experience in using them. To overcome these limitations, devices capable of visually interpreting physical gestures are becoming increasingly popular. These tracking devices generally allow body or hands tracking in order to provide a more immersive experience. The numerous technical challenges make these devices still premature to be used on a large extent [12, 21]. Despite this, devices such as Leap Motion have reached sufficient maturity to be used as a control interface in VR experiences.

2.2 Leap Motion

The Leap Motion device [30], created by the company of the same name, is a hardware sensor that allows tracking of hands and fingers. It takes the form of a small USB device, designed to be positioned above a surface or alternatively mounted on an HMD (Fig 2.2).

Through the use of two monochrome IR cameras and three infrared LEDs, it is able to detect hands within a meter radius of its surface without any physical contact. It has a field of view of about 150° on the long side, and 120° for the short side. Its operating range is between 2.5 and 8 cm from the surface of the device. Leap Motion is able to provide an accurate tracking, given the small size

Figure 2.2: Leap Motion device.

of its field of view. The generated data, or leap-frames, have a normal frequency of 115 frames per second (fps) [31, 32]. According to various studies, the accuracy offered by Leap Motion is 0.7mm [12]. The images obtained from the cameras are transformed into data through the use of algorithms not made publicly available by the company. The data obtained by the device can be used by any software using special libraries provided by Leap Motion. Leap Motion therefore needs special software in order to work. The first versions of this SDK, called V1 and V2 [34], were released in 2014 and support Windows, Mac Os and Linux operating systems. These software versions do not support VR environments in a native way and are therefore not recommended for use by the manufacturer. The latest version of this software, called Orion [35], has been designed with the world of virtual reality in mind. It offers reduced latency and improved tracking compared to previous versions. Currently, the Orion version is only available for Windows environments and is released in Beta state. Given the high computational demand required for proper tracking performance, use in portable or low-cost devices such as RaspBerry Pi is not currently possible. The release of a mobile version of the device compatible with smartphones has already been announced [33], even if details about the release date are not yet available. The Leap SDK also offers libraries for C++ and C# programming languages, in addition to the support for Unity3D and Unreal Engine game engines. Integration of Leap Motion with the Unity3D environment is covered in greater detail in section 2.3.3. A javascript library called 'leap.js' is also available for the creation of Web applications.

Leap Motion, in addition to being used as a peripheral device for applications and games, has already been used in some research fields, such as medical and industrial research. Reference [4] studied the possibility of using Leap Motion as a controller for a robotic arm. In addition, numerous studies have evaluated the possibility of its use to recognise sign languages in real time [18].

Although not originally designed for VR, the Leap Motion has found a natural application in this field. Its small physical dimensions combined with its light weight allow it to be placed on any HMD without weighing it down significantly. The high tracking accuracy allows for a new kind of user interaction. Through the use of one's own hands, in fact, any person can interact with a virtual world: from the recognition of specific gestures to taking objects by pinching with fingers. Therefore, a more immersive experience can be provided compared to traditional controllers. In order to recreate the tracked hands accurately and offer this type of immersive interaction, the Leap Motion needs to encode specific data about fingers and hands, such as their position, rotation, etc. As previously discussed, this data is saved in special objects called leap-frames.

2.2.1 Leap-frames data

Data contained in leap-frames differ according to the version of the software used. There are also small variations on the structure of the classes depending on the programming language. The data composition used by the latest version of Orion available for the C# language [36] will now be briefly reviewed.

Each single frame object contains all the properties relative to the hands traced for a precise moment of time, as well as useful information such as the current refresh rate and the timestamp associated with that frame. Each Hand object in turn contains all the specific information related to

a single hand: position in space, rotation, number and type of fingers associated with that hand. In addition, it also contains information about its associated arm. Each traced finger consists of four bones, represented by the Bone object. Numerous position and rotation vectors for each bone allow an accurate representation of the fingers. In the case of the thumb, the metacarpal bone is set to zero length to reflect the actual anatomical model. All measurements made by the Leap Motion are provided in millimetres and radiants in the case of distances and angles, respectively. The Leap Motion device uses a right-handed Cartesian coordinate system, where the origin is centered at the top of the device.

The leap-frames obtained from the device are made available to any application that requires them through the use of sockets. The Leap software provides a TCP socket for sharing data for local applications, and a server using the Websocket protocol for remote applications.

2.2.2 Websocket server

For Web or remote applications, Leap Motion's installed software provides a Websocket server [36] for obtaining tracking data. The server is hosted in the computer connected to the Leap Motion device. Any client application that supports a Websocket connection can therefore access the Leap Motion tracking data in the form of JSON-encoded strings. In addition to sending tracking data, the server can also notify connected clients of events such as connecting or disconnecting the Leap Motion device with the computer. The Websocket server has a subprotocol used for communication with the client, in order to be able to specify the type and composition of sent messages. If a connected client needs to use the VR tracking mode, it can, for example, notify the server by sending the JSON-encoded message '*"optimizeHMD": true*'. The version of the protocol to be used may also be specified, where each version of the protocol has its own different features set.

2.3 Unity3D

Unity3D [40] is a game engine developed specifically for the creation of 2D and 3D games and simulations. It is available for many platforms, including computers, consoles and mobile devices. Features such as graphics rendering, custom materials creation, physical simulation, multiplayer support and scripting capabilities make Unity a powerful tool to create a wide variety of applications. Its ease of use, extensive support and numerous updates have made Unity a popular graphics engine [44]. By using the Asset store custom assets created and made available by the Unity community can be bought or sold.

Currently Unity is available in four licenses: Personal, Plus, Pro and Enterprise. Unity Personal [43] is available free of charge and is especially suitable for use by students, researchers and hobbyists. It contains all of the engine functionalities and deployment capabilities for every supported platform. Its use is limited to those who do not exceed $100K in annual gross revenues. The Plus, Pro and Enterprise versions contain additional features suitable for large development studios, at the expense of a monthly fee. Our prototype is based on Unity personal.

Unity defines some basic concepts for the creation of scenes and graphic objects. The most relevant concepts will now be explained:

- **Gameobject**: basic fundamental class. Each entity in the application is in fact a GameObject, with a custom behaviour defined by specific scripts called Component.
- **Component**: custom script that defines a certain behaviour or characteristic. A component must be associated with a Gameobject in order to work. There are a number of components already made available by Unity. Creation of custom components is possible using the C# programming language. Each component inherits from a parent class called MonoBehaviour that provides some useful functions to execute code at specific points in the life cycle of a component.
- **Scene**: a scene contains the environment and all the objects in the game present in a certain level or screen. A scene may contain a menu, interface, level or the whole game.

- **Asset**: any file that can be used in the game or project. Models, texture, audio files and images are all examples of assets. A Unity3D project therefore consists of a set of assets.

Our project, like any other Unity3D application, makes use of these concepts. In addition to using some essential Components such as those used for graphic rendering, the implementation aspect of the thesis focuses on some specific Assets needed for Leap Motion to work in a VR environment. In addition, the prototype produced makes particular use of the multiplayer aspect of Unity. We will now explain these concepts in more detail.

2.3.1 Multiplayer support

Unity offers support for local network or Internet multiplayer applications [42]. Two different systems are available for building multiplayer capabilities for Unity games: the High-Level APIs (HLAPIs) and the Low-Level APIs (LLAPIs). LLAPIs offer a real-time transport layer and are suitable for the creation of network infrastructures or advanced applications. On the other hand, HLAPIs are recommended for the creation of video games without particular demands. Our working prototype has been developed using HLAPIs.

HLAPIs have been built using LLAPIs as a lower transport layer and are responsible for managing numerous tasks common to this type of games. Below is a list of the main features that HLAPIs support:

- Network Manager for managing the networked state of the game
- General-purpose data serializer
- Ability to send and receive network messages, including events, commands and remote procedures
- Object spawning system
- State-synchronization across the network

HLAPIs implement a client-server architecture with an authoritative server. Since HLAPIs use an authoritative server system, the server has the role and the authority to update the state of the objects in the environment. It is however possible to assign the authority for an object to a specific client. In this way, a client can request changes to an object by sending a private command to the Server. This approach is used for player representative objects in the game world, so that a client can communicate to the server the latest information based on local user input. When an object has to be created in the game world, its creation is delegated to the Spawning System. This system creates the object in all instances of the game, and synchronizes its properties to all connected players. Each object has a unique identifier known as 'NetId', that is the same on the server and clients. This attribute is used to identify the object on the network. The use of this object management system is indispensable to realize applications using these APIs. Section 3.4 describes how these concepts have been used and adapted to implement the prototype.

Although it is possible to have a dedicated server, the creation of a game is generally entrusted to one of the participants, who therefore assumes the role of client and host. In our case, both the host and connected clients use smartphones to participate in the same game session. Connected users can play with each other by connecting to the same LAN network.

2.3.2 Virtual reality integration with Google VR

Unity supports integration with VR. A single API interface is provided to interact with a variety of VR devices. If enabled, Unity will automatically render on an head-mounted display. In addition, head tracking and FOV are automatically applied to the game's camera, so that the movements in the game may match those of the user. This aspect is necessary to get a good VR experience without nausea.

Many different VR SDKs are supported by Unity, including Google VR. The Google VR SDK provides support for Cardboard and DayDream technologies. It makes a large number of assets available to support different kinds of HMDs and input controllers. Development with Google VR is allowed for devices running Android Lollipop or higher; or alternatively iOS 8 or higher.

2.3.3 Leap Motion assets

The Leap Motion Orion SDK contains specific assets and modules to support the Unity game engine. This collection of assets allows the use of the Leap Motion device in any Unity application, as well as providing a set of tools to simplify the design of user interfaces, hands and interactions. Specifically, a set of assets called "Core Assets" is necessary to obtain Leap data in Unity environments. In addition, some optional modules are provided, each with a specific function:

- **Interaction Engine**: The Interaction Engine allows natural custom interactions with any game object. By using the interaction engine, specific behaviours can be specified for actions such as touching, grasping or hovering over a Gameobject with the virtual hands.
- **Graphic Renderer**: The Graphic Renderer Module is used to optimize draw calls and to design user-friendly curved interfaces.
- **Hands Module**: The Hands Module provides a handful collection of hands models ready to use, in addition to an autorigging pipeline used to create custom models.

In order to visualize and create hands contained in leap-frames, some components provided by the Core Assets must be used. The LeapServiceProvider class is used to communicate with the Leap software installed on the computer. It makes Frame objects available to all components that require them. LeapHandController and HandPool components use the frame objects obtained by the provider class to generate and update hand models based on frame data in real time. Moreover, it is possible to specify the aspect of the virtual hands through different implementations of the HandModel class.

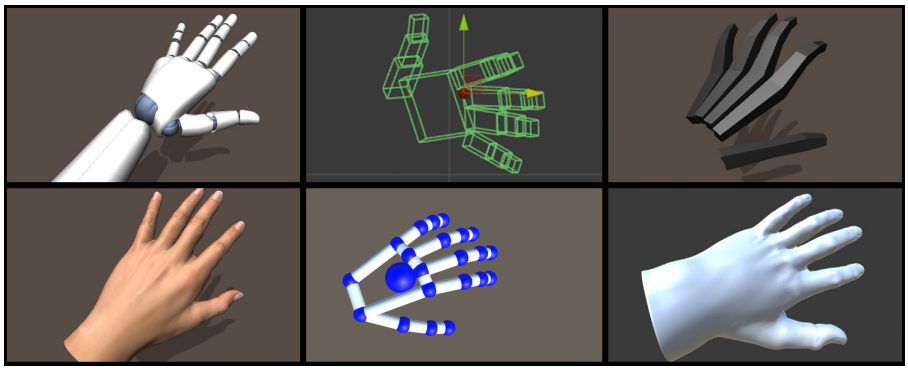


Figure 2.3: Various aspects of virtual hands, based on the Leap Motion assets.

2.4 Proposed distributed approach

This section will explain the method by which we managed the transfer of a large amount of data between multiple users, using a distributed approach. In particular, this approach is used for the purpose of transmitting visual information about the hands of each player to allow for gesture based interactions. Due to the large amount of information generated by Leap Motion, it is not possible to transfer these data between multiple users in a "vanilla" way. If fluid interactions are to be achieved with as little latency as possible, optimizations and specific approaches are therefore required. Given a set of participants that interact in the same virtual environment, we will describe the proposed communication mechanism where pairs of users exchange visualisation data under reciprocal requests of levels of detail that depend on their spatial distance in the VR space.

2.4.1 Communication mechanisms

The communication amongst users is decoupled to handle data differently based on its type. We use the typical authoritative mechanism to handle client enrollments and a peer-to-peer data exchange strategy to handle high-throughput visualisation data efficiently. In our case visualisation data are produced by Leap Motion [21].

A user that initiates a VR session can be both the host and a client. The host is in charge of (i) updating the states of the mutable objects that require synchronisation amongst clients and

(ii) managing client enrollments/disenrollments. When a client enrolls to a VR session, the host broadcasts its IP and port addresses (enclosed in a broadcast message) to the other connected clients. When a client disenrolls, the host informs the connected clients that a client has left the virtual environment. These broadcast messages are used by each client to store and maintain an updated table with the network addresses of the connected clients. We name this table as *Sync Table*. Each client has the same copy of the Sync Table and can use this global knowledge to establish peer-to-peer communications with other clients to exchange visualisation data. In this way, high-throughput data does not go through the host thus reducing its the computational load. This is an important element because when devices like smartphones are used, they have limited computational capability and battery duration.

Fig. 2.4 shows an example where we can observe that the requests and the information exchanged over the network can be halved in the case of peer-to-peer communications.

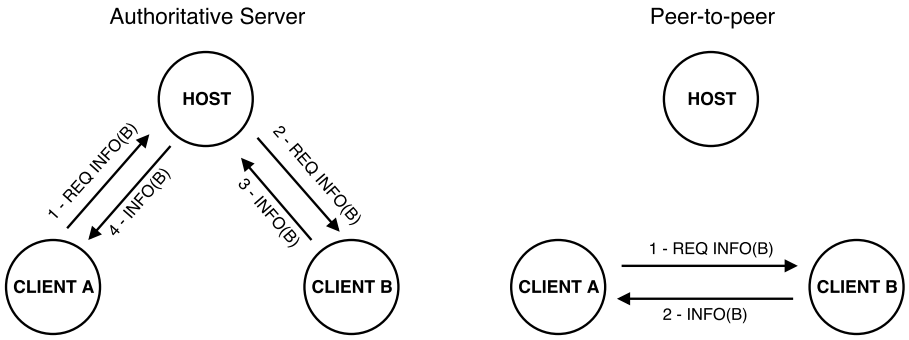


Figure 2.4: Difference between authoritative and peer-to-peer mechanisms. Requests (REQ) and information (INFO) exchanged over the network are halved in the case of peer-to-peer communications.

2.4.2 Distance-based level of detail

Data that represents hand gestures are typically sampled at a high frame rate to visualise natural movements. However, when hand gestures are seen from far, details might be unnoticeable. We exploit the knowledge that each client has about the position of the other clients to dynamically variate the level of detail at which the visualisation data are exchanged. Levels of detail are typically used to define multiple representations of a model with decreasing resolution in order to reduce the rendering cost for distant or less important objects [23].

In order to achieve this we designed a request-based mechanism. Clients that are involved in an interaction, reciprocally request their desired level of detail to other clients. The queried clients that accept the request will transmit the visualisation data at the requested level of details. We use the spatial distance between clients in the VR space as a criterion to select the appropriate level of detail: the closer the two clients, the higher the resolution of the visualisation data they exchange. Note that, a system based on requests can also provide the possibility to extend this distance-based criterion to additional criteria for example based on network or rendering capacity. In this work we analyse the case of distance only.

Fig. 2.5 shows five clients that are connected to the same VR space, one is the host/client and the others are clients. The top part of the figure shows an example with four levels of detail:
L0 defines the maximum level of detail where no approximations of the hand joints are applied. This could be the case where two or more clients are interacting "face to face" and high-detailed joint movements is key to visualise natural gestures;
L1 defines an intermediate level of detail where a subset of joints are transmitted, while ensuring that a well approximated hand motion can still be perceived. This could the case where the client requesting the visualisation data is not interacting directly with the queried client, but their distance is such that their hand movements are still visible;
L2 defines another intermediate level of detail where a minimal subset of joints is transmitted to show basic hand movements. This could the case where the client requesting the visualisation data is far from the queried client and its hands are barely visible;
L3 defines no data transmission as clients are out of line of sight.

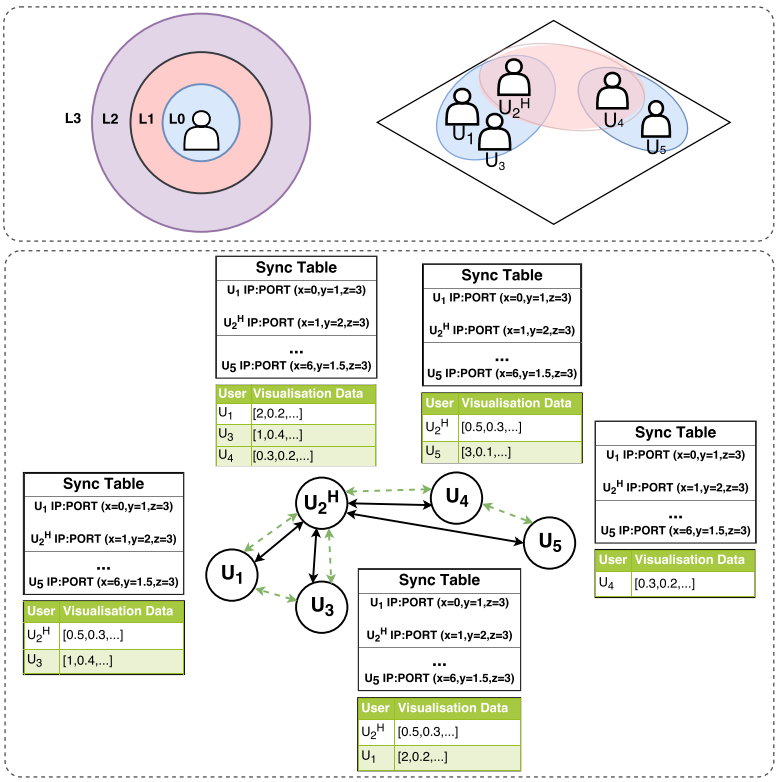


Figure 2.5: Five clients are connected to the same VR space, one is the host/client and the others are clients. On the top four levels of detail are defined: L0 defines the maximum level of detail, L1 and L2 define intermediates level of detail, and L3 no data transmission. The lower part of figure illustrates the connections (i.e. black arrows) the server uses to broadcast the information about enrollments/disenrollments that are used to update the Sync Table on each client. Each client uses the information included in this Sync Table to establish peer-to-peer connections with other clients. Green arrows show the peer-to-peer connections between clients to exchange high-throughput visualisation data.

The lower part of figure illustrates the connections (i.e. black arrows) the server uses to broadcast the information about enrollments/disenrollments that are used to update the Sync Table on each client. Each client uses the information included in this Sync Table to establish peer-to-peer connections with other clients. Green arrows show the peer-to-peer connections between clients to exchange high-throughput visualisation data.

2.5 Summary

In this chapter we proceeded to deepen the theoretical knowledge necessary to understand the work addressed in this thesis. We have described the features of the Leap Motion device used for tracking and have also outlined the operation of the Unity3D game engine. In addition, we discussed and explained an approach designed by us with the specific purpose of transmitting visual data relating to users' hands. The next chapter will explain how these technologies and approaches have been used to create the prototype.

3 Prototype development

This chapter describes the implementation of a working prototype. The created prototype serves as a proof of concept and therefore aims to demonstrate the creation of a VR multi-user environment using low-cost platforms accessible to all. For this purpose, the prototype produced is capable of working on any recently released Android mobile device, although iOS versions could easily be developed. Google Cardboard technology is used as a HMD for VR and the Leap Motion is used as the input device. In order to make a VR experience accessible to all, as well as not being too expensive, it must be easy and intuitive. The use of the Leap Motion device, which allows gesture-based interactions, provides a much more natural type of interaction than a traditional controller.

The prototype implements the following basic features required in environments of this kind:

- **Player movement**: players can move and look around inside the virtual world.
- **Hand based objects interactions**: virtual objects inside the environment can be grabbed, moved, rotated or scaled.
- **Reticle based interactions**: interactions based on where the user is looking at.
- **Hand visualisation**: hands are transmitted among users in high details.
- **Multi-user support**: players can host or join already created sessions and play with each other.

In the next section, a general overview of the application architecture will be presented. We will then discuss the features listed above how they have been implemented.

3.1 Architecture overview

The application has been created using Unity3D version 2017.2 and is designed for use on Android smartphones. Each client is connected to the Leap Motion's Web service that is hosted on a computer and leap-frames are transmitted encoded in the JSON format. The client deserialises the JSON-formatted leap-frames and converts them to C# classes. The obtained leap-frames are then used to render the local hands and make physical calculations on objects. Using this system, real hands are tracked and used as an interface to interact with the application in real time. Changes made by a client to an object are synchronized to all. We implemented the client-host requests to update the states of mutable objects using the Unity3D's native authoritative mechanism provided through the HLAPIs. Peer-to-peer visualisation data exchanges are managed by the proposed distributed request-based mechanism. We use the UDP protocol for timely delivery of data. Each client periodically listens for incoming UDP packets on the network socket. The system architecture of our prototype can be seen in Fig 3.1. The figure shows three conceptual blocks: green for the local Leap Motion connection, blue for the local processing of the hand and its interactions and red for multiplayer.

3.2 Bringing Leap Motion to mobile devices

Leap Motion does not yet support its direct connection to smartphones as previously discussed on Sec 2.2. Therefore, the Leap Motion is connected to a computer that in turns transmits the hand tracking data to the smartphone via Websocket server. The created prototype uses the latest available protocol, the "v6.json" version. Note that our communication approach is independent from this computer-smartphone connection because hand joints' data is handled at network level as it were processed on the smartphone. When the mobile version of Leap Motion is released, it will therefore be possible to use the prototype with the new version of the device without major modifications as the leap-frames obtained and converted from the server behave in the same way as those obtained

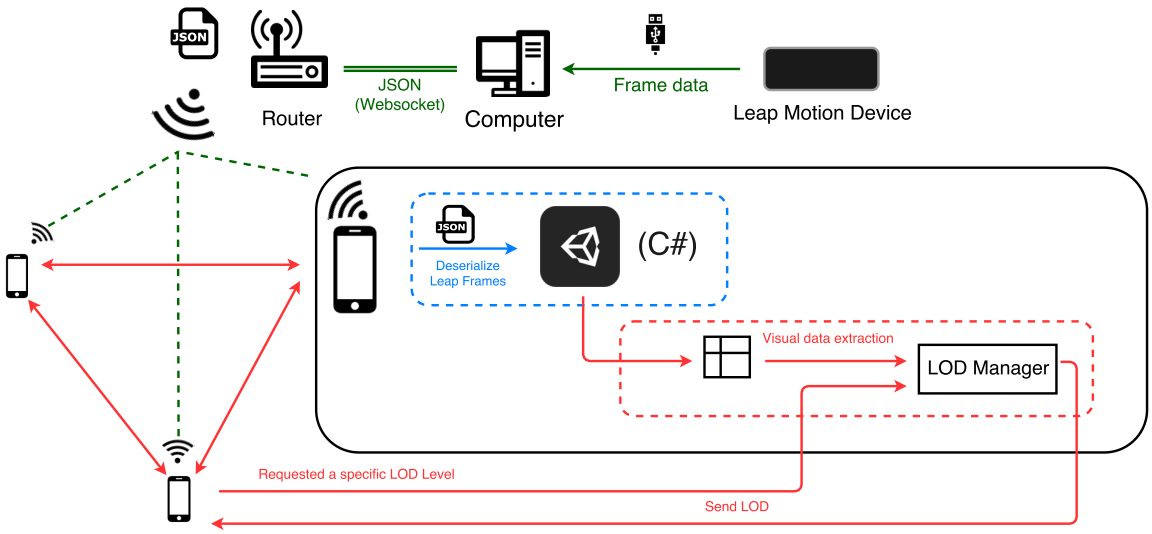


Figure 3.1: Illustration of the setup architecture. The figure shows three conceptual blocks: green for the local Leap Motion connection, blue for the local processing of the hand visualisation and red for the multi-user aspect.

natively. When these leap-frames are received and converted by the application, hand information is used locally on clients for hand interactions with mutable objects (that are then managed by the authoritative server).

3.2.1 Frame conversion

Figure 3.2 illustrates how the Components used by Leap Motion assets work in the Unity environment compared to our custom Components. The custom class 'WebSocketConnection' on the client side takes care of getting messages from the server as a string and stores them in a buffer. These messages are then taken over by another custom class, 'FrameProvider', which converts the strings into native C# objects, ready to be used with the original assets created by Leap Motion. For conversion from string to class, a custom parser specific to the protocol version used has been created. In this way, a series of optimizations and operations are carried out to convert the data received from JSON. The Leap motion data encoded in JSON does not fully correspond to the data used by C# classes, and some calculations are therefore necessary to obtain a copy of the leap-frame with the same semantic value as if it were obtained natively. For example, to handle rotations, the corresponding C# class uses quaternions, while in the JSON rotations are coded as vectors.

Unity APIs are not thread safe and are therefore only usable in the main thread, including those used for graphic rendering. In order not to increase the number of tasks in the main thread and not to cause a significant drop in frame rate, we carry out all conversion tasks in separate threads. For resource sharing between different threads, the lock construct was used to protect critical sections. Given the high frequency with which data is generated on the computer, there may be problems with the large amount of bandwidth used or the large amount of data to be converted. Restrictions may therefore arise in the wireless network used or in the phone's computational capabilities. These limitations would lead to an exponential increase in latency and thus make the VR experience not adequate. To solve these problems we have adopted two similar approaches based on limiting the Leap Motion hands update frame rate.

3.2.2 Frame rate limit approaches

To decrease the large number of calculations to be performed, a naive solution can be adopted, i.e. to ignore the processing of some leap-frames. Instead of converting each string received from the server, we can choose to convert only those at a certain frequency. This approach has the advantage of allowing each individual device to choose the fluidity of the recreated hands but does not lead to any improvement in the amount of bandwidth used. The opposite alternative is therefore to limit the

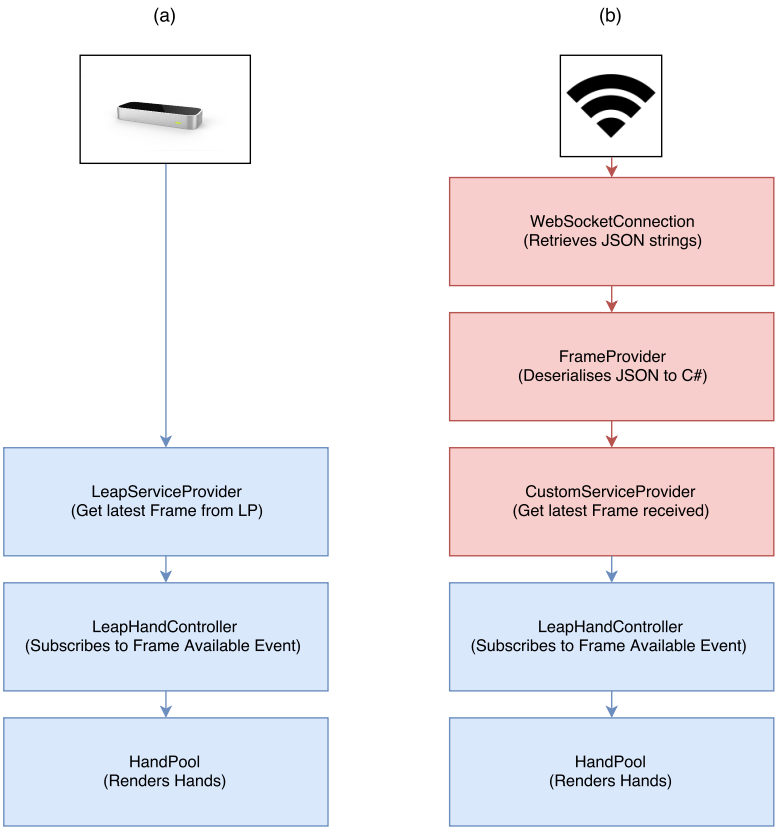


Figure 3.2: Obtaining and using Leap Motion data in Unity3D: comparison of standard flow and our remote approach. Red rectangles represent custom Components, whereas blue ones represent untouched Leap Motion Components.
(a) Shows the native method, usable by every Unity3D application directly connected to the Leap Motion.
(b) Illustrates the approach we used to get usable leap-frames in smartphones.

frequency of server-side updates, in order to decrease the data usage. The Websocket server made available by Leap Motion software does not allow for a customization of the update frequency, so to implement such an approach we developed a custom Websocket server. This server, located on the computer connected to the Leap, is used as an intermediate layer between the native server and the client. This custom server therefore receives all the leap-frames from the native server but sends only a subset of them to the client. The prototype created uses this methodology. In this way we can achieve a great improvement in the amount of data sent. Considering a minimum frame rate limit of 30 fps to have a good hand fluidity, and a 10KB leap-frame size in JSON, we get considerable reductions if we compare this approach to sending every leap-frame using the native server with a medium frequency of 110 fps.

3.3 VR interactions

A collaborative environment, to define itself as such, must at least have some kind of interaction between players and between objects present in the virtual world. In our prototype, interactions can be divided into two categories: those based on the gaze and those that make use of hands.

3.3.1 Gaze interactions

Gaze-based interactions are very common in VR environments and allow actions to be carried out based on the direction in which a user is looking. Therefore, we placed a reticle in the middle of the screen that has a double purpose: to make the user more aware of his actual direction in the virtual world and to notify him about possible actions that can be taken with the object the user is looking

at. To activate a gaze-based interaction, the user can close his right hand while observing a particular object to express the intention to activate an action. Alternatively, the user can use a physical key in a controller connected to the prototype as confirmation input. This type of interaction allows the prototype to be used even without the Leap Motion device, albeit with limited functions.

An observed object can belong into three possible cases, illustrated in Figure 3.3:

- **No action available**: no interaction with this object is possible. The reticle colour changes to match gray (a).
- **Available action that can be activated by pressing a button**: The observed object contains an action that can be activated by pressing a button on a controller or by closing a hand while you are watching that object. An example of an object with this type of action could be a music player, where we want to change audio tracks quickly. The reticle colour changes to red (b).
- **Available action that can be activated by holding down a button**: The observed object contains an action that can be activated by holding down a key on a controller or closing hands for a certain period of time. This type of interaction is used for important actions that require additional user confirmation, such as moving around the game world. The reticle uses a radial selection bar and a selection slider (c) (d).

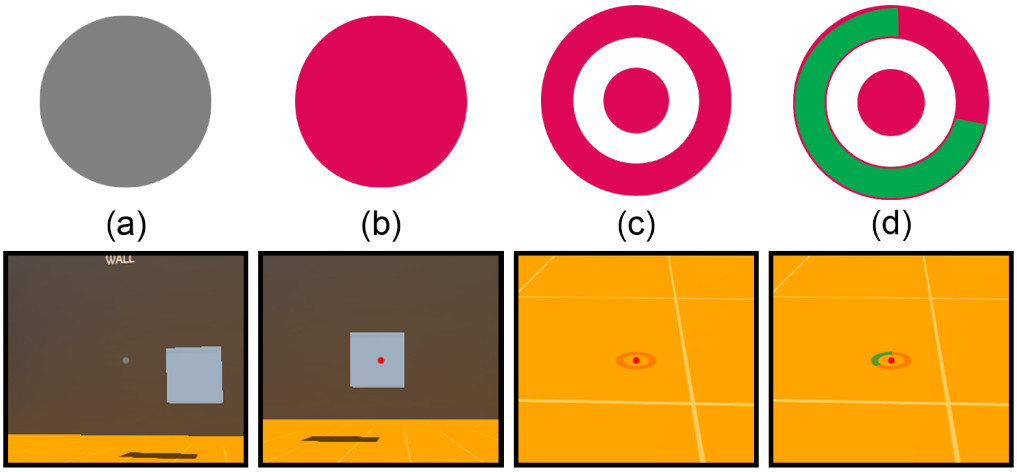


Figure 3.3: Different aspects of the reticle for gaze-based interactions.

For the implementation of this type of interaction, we created a generic Component applicable to any Gameobject that we want to make interactive, following the observer design pattern principle. The purpose of this component is therefore to notify any observer that a particular event has been called in order to initiate the corresponding action. The raycast technique is used to locate the observed object: a ray is fired from the player's position in the direction of his gaze. If the ray hits an object, we check if it is interactive and if true the corresponding event is fired.

3.3.2 Hands interactions

In addition to creating interactions based on gaze, our prototype is characterised by the possibility of using hands to interact with objects in the virtual world. Objects can be moved, rotated or scaled. Each object, through the use of native Unity Components, has physical properties and therefore reacts to gravity, motion or collision with other objects. The last available leap-frame is analyzed to understand the hands' gestures: if it is a "pinch" gesture, the area close to the hand is checked, and if an object is present, the object is picked up and the grab mode is activated. The object therefore changes colour to indicate that it has been taken and will move together with the hand as long as it respects that pinch gesture. When the hand is released, the object returns to its original state and regains its physical properties.

Figure 3.4 shows a possible sequence of actions for this type of interaction. A user wants to take an object, so he approaches it with his hand (a). By pinching, the user takes the object (b) and places it on top of another object. Once positioned, the object is released (c).

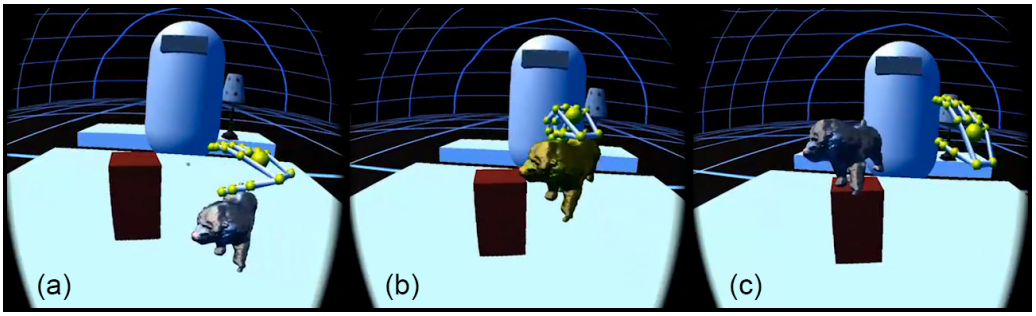


Figure 3.4: A player picks up an object and places it on top of a cube (First person view).

3.4 Multiplayer

The interactions addressed in the previous chapter are part of the local aspect of the prototype. In fact, any user could start the application to perform one or more of the interactions just discussed, even in a single player gaming session. However, in the case of multiple connected users, a modification or result caused by any of these interactions must be transmitted to all connected users. The multiplayer aspect of the prototype is therefore responsible for synchronising the latest game-state between users, such as the position of users and objects at a given time. The transmission of visualisation data is also included in the multiplayer aspect.

3.4.1 Player and object synchronization

We synchronise the latest game-state using a client server architecture, following the HLAPIs implementation. Each client, at regular intervals, notifies the server of any changes to be made to objects under its authority. The server, being authoritative, is responsible for making changes and updating all clients, including the requester. There are therefore two possible circumstances for a client: either it causes changes in the game status, or it receives changes made by others. Clients transmit information about objects for which they have authority, including the player itself or a handheld object. The transmission takes place only in case of changes compared to a previous time interval. In order to save bandwidth, only the actual changes since the last time interval are transmitted, and not the full status of the objects. A mechanism of the kind described above is adequate to create multiuser experiences, but it is not sufficient to ensure fluid experiences [29]. The first problem concerns the fluidity of actions carried out at local level. For each action that the user performs, in fact, before being able to view the changes caused, he must wait for the server to receive his request and for it to respond with the modified data. There would therefore be a delay between the player's actions and the results seen and this would lead to frustration and poor immersion. We used the client-side prediction technique [29] to solve this problem. Instead of sending the inputs and waiting for the new game state to start rendering it, we can send the input and start rendering the outcome of that inputs as if they had succeded, while we wait for the server to send the "true" game state. With this approach, there are no longer any delays between the actions performed by the player and the results seen, while the server maintains its authoritative status. Another problem caused by the naive implementation would be related to the fluidity of entities updated by others. Since the server sends state updates every certain interval of time, objects' movement would be seen as choppy and jumping at discrete positions. We therefore apply interpolation between the last two received state-updates of the object, in order to create a more fluid movement. Algorithm 1 presents generic pseudocode to illustrate the procedure described above. Each individual object whose status must be synchronized within the network implements a Component with the same functionality.

```
Data:
SyncVector3 latestPosition;
SyncVector3 latestRotation;
SyncVector3 latestScale;
t_1 = Time.Now;
Δ_s = 0.11s;
HasAuthority() = true if local player controls object, false otherwise;
while true do
    if HasAuthority() == true then
        if (t_1 - Time.Now) > Δ_s then
            if (position is Changed) then
                SendPosition(transform.position);
                latestPosition = transform.position;
            end
            if (rotation is Changed) then
                SendRotation(transform.rotation);
                latestRotation = transform.rotation;
            end
            if (scale is Changed) then
                SendScale(transform.localScale);
                latestScale = transform.scale;
            end
            t_1 = Time.Now;
        end
    else
        InterpolatePosition();
        InterpolateRotation();
        InterpolateScale();
    end
end
```

Algorithm 1: Generic Object Synchronisation

Table 3.1: Messages defined in the communication protocol.

Header	Syntax	Description
AIP	[AIP]:<ip>,<port>,<netID>	Message the host sends to inform clients to add a new entry in the Sync Table
RIP	[RIP]:<netID>	Message the host sends to inform clients to remove the entry of the disconnected client from the Sync Table
RQ<n>	[RQ<n>]:<netID>	Message a client sends to another client to request visualisation data at a specific level of detail
AK<n>	[AK<n>]:<HandsParam>,<netID>	Message a queried client sends to a requesting client that contains the visualisation data at a specific level of detail in the payload; <HandsParam> integer value that defines the type of hand contained in the payload (0: left, 1: right, 2: both)

3.4.2 Visualisation data exchange

We use the UDP protocol for timely delivery of data. Each client periodically listens for incoming UDP packets on the network socket.

To enable effective communications, we define a protocol that specifies the structure of the messages exchanged between the host and clients to update the Sync Table, and between clients to send requests and transmit visualisation data at different levels of detail. Table 3.1 lists the messages defined by our protocol.

The protocol functions in the following way. A client joins a VR session through HLAPIs. The host assigns a unique identifier (*NetID*) to this client and broadcasts a message informing the connected clients that an enrollment has occurred. Each message is composed of a header and a payload. The header uses the first three bytes of the message to define the type of data carried by the payload. When a client connects, the host sends an 'AIP' message to inform clients that a new entry in the Sync Table should be added. When a client disconnects, the host sends an 'RIP' message to inform clients a client should be removed from the Sync Table.

Clients are responsible for requesting the visualisation data to other clients. If client A wants visualisation data of client B, A will send a request message 'RQ' to B requesting the visualisation data at the desired level of details (i.e. based on their distance). When B receives the request, B encapsulates the visualisation data in a message 'AK' and sends it to A. This procedure is repeated periodically at a pre-defined frequency to show hand movements. Algorithm 2 details the overall message handing procedure.

When the leap-frames are received by the smartphone using the Websocket connection (sec 3.2), we use all the hand interaction elements on each client to handle hand interactions with mutable objects through communications with the authoritative server. Note that all the hand interaction elements within a leap-frame are necessary to Unity3D to trigger the correct physics when interactions with mutable objects occur. Differently, we use only a subset of the hand interaction elements within a leap-frame (i.e. joints positions) as visualisation data to transmit (at the appropriate level of detail) to the other clients.

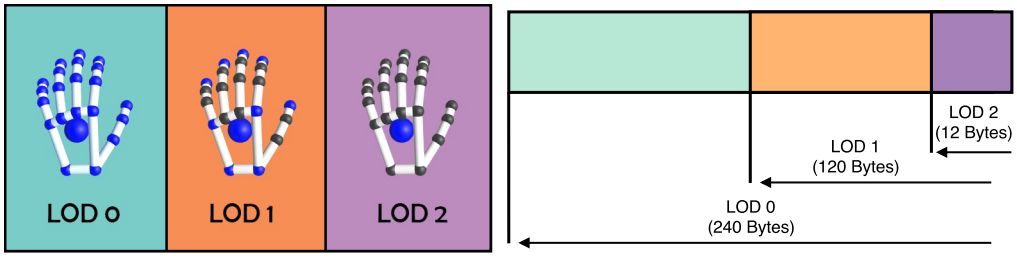
As described in Sec. 2.4.2, we use four levels of detail: L0, L1, L2 and L3. L0 defines the highest level of detail, L1 and L2 are intermediate levels, and L3 defines no data transmission.

Fig. 3.5 illustrates how the hands are defined at different levels of detail and how visualisation data are organised to efficiently access them for different levels of detail. One hand is represented at L0 with 240 bytes, at L1 with 120 bytes and at L2 with 12 bytes.

Fig. 3.6 shows a scenario where five clients are within the same VR space and visualisation data are exchanged using the proposed distributed mechanism with levels of detail. The left-hand figure is a screenshot taken from a Cardboard device, while the right-hand figure is a screenshot taken from a

Data:
t_1 = Time.Now; t_2 = Time.Now;
Δ_s = 0.2s; Δ_r = 0.3s;
m = message; c = client;
LOD = level of detail;
Handle(m) = message handling as defined in Tab. 3.1;
while *true* **do**
  if *(t_1 - Time.Now)* $> \Delta_s$ **then**
    handData = getHandsData();
    t_1 = Time.Now;
  end
  if *(t_2 - Time.Now)* $> \Delta_r$ **then**
    for *c in ConnectedClients* **do**
      LOD = calculateDistance(localClient, c);
      requestHandData(c, LOD);
    end
    t_2 = Time.Now;
  end
  if *m == received* **then**
    Handle(m);
  end
end

Algorithm 2: Message handling procedure.



(a) (b)

Figure 3.5: Example of three levels of detail with associated bytes count for the Leap Motion hand. Blue joints are encoded and transmitted, gray joints are ignored.

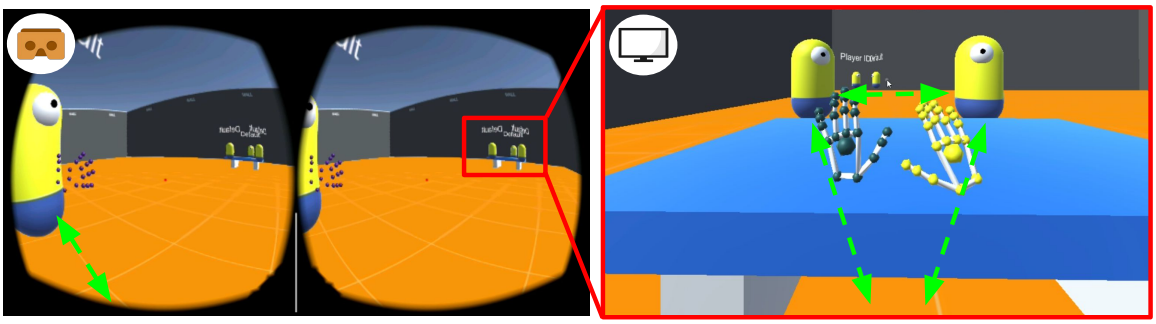


Figure 3.6: Scenario where five participants are within the same VR space and visualisation data are exchanged through the proposed distributed mechanism using levels of detail. The green arrows illustrate whom the visualisation data is shared with.

desktop computer. The distance between clients is such that they create two groups of data exchange: one group of two clients (left-hand figure) and one group of three clients (right-hand figure). In each group there is one client waving its hands. With this configuration L0 (max resolution) is applied between clients of each group and L3 (no data) is applied between clients of different groups. The green arrows illustrate whom the visualisation data is shared with. From the Cardboard's view we can see the group of three clients in the background and we can observe that their hands' movements would not be visible even if their visualisation data were transmitted to the clients of the other group.

3.5 Summary

In this chapter we have presented the prototype produced, intended as a proof of concept. In addition to listing the expected features, we have explained how they work and how they have been implemented. The features presented, even if basic, already allow a decent level of interaction and represent a starting point for many different types of applications. In the next chapter we will analyse in more detail some technical aspects of the created product, such as frame rate or data consumption. As a result, we will evaluate the actual usability and robustness of the prototype.

4 Results

In this chapter we will present and discuss results obtained from different evaluations made on our prototype. The aim of the project is to create a VR application on mobile devices. Therefore, particular attention has been paid to the frame rate obtained and to the latency measured by the interactions between several users, as a VR experience must be fluid in order not to cause problems for the player. In addition to quantitative results, some qualitative results will be discussed as a result of the demonstration performed at the FBK 'Open Day' event.

4.1 Frame rate analysis

In this section we wanted to verify the frame rate obtained during the application execution. This is very important in a VR application because, if the frame rate is low, the user may experience motion sickness. The frame rate was measured on a scenario with a connected user in a stationary position while using two hands. The scenario was recorded on a Huawei Honor 8. Two measurements of the same scenario were made. The first one was carried out on a "cold" phone, i. e. without having been used for several minutes, in order to have a CPU temperature equivalent to a normal use. The second test was recorded after intensive use, where the CPU temperature was close to 40° Celsius. We made these two measurements to evaluate the effect of thermal throttling. Termal throttling is a widespread phenomenon in all smartphones, where the system lowers the frequency of use of CPU and/or GPU to better disperse heat [22]. Figure 4.1 shows the frame rate recorded in these two sessions.

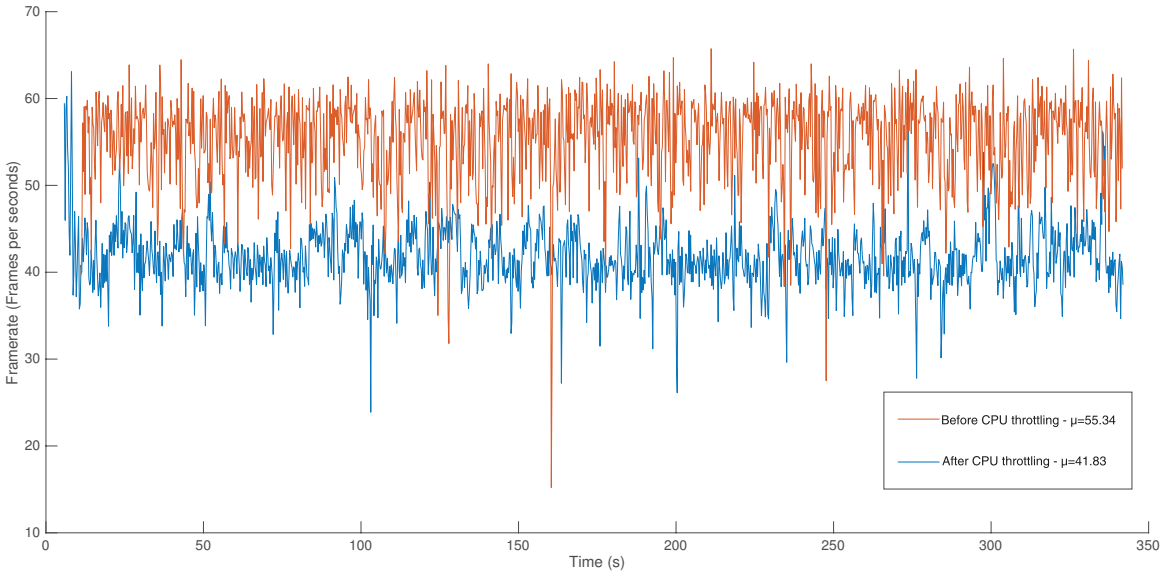


Figure 4.1: Frame rate recorded using the same scenario in two different instances: before and after aggressive usage of the smartphone.

As the figure illustrates, performance after moderate use is severely degraded. For maximum fluidity, it would be desirable to have a frame rate equivalent to the screen refresh rate. For most smartphones, the refresh rate is 60Hz. Neither of the two measurements managed to maintain a stable frame rate at 60fps, even if the recording made on a cold telephone managed to get very close, with an average of μ =55.34.

4.2 Distributed method analysis

We evaluated the proposed method by performing two experiments. Firstly, we measured the number of bytes transmitted during a simulated collaborative VR session. Secondly, we used multiple smartphones as Cardboard devices and measured their transmission latency within a local wireless network. In all the experiments, we compared the performance of our distributed approach against a centralised (i.e. authoritative) approach, and by enabling and disabling the level of detail strategy, namely LOD and NO LOD, respectively. All experiments involved up to seven clients connected to the same VR space. One of the clients was the host. In order to stress the system, we streamed sequences of leap-frames corresponding to two tracked hands, i.e. 480 bytes per leap-frame.

4.2.1 Network traffic

To measure the network traffic, we designed a simulation where the seven clients were positioned in the VR space randomly. The VR space was 20x15 Unity3D units. The level of detail regions had radii of 4, 8, 12 and >12 Unity3D units to define L0, L1, L2 and L3, respectively. Because the VR space was limited and because the duration of the experiment was 150 seconds, we could simulate several combinations of relative distances between clients and hence trigger transmissions with different levels of detail.

Fig. 4.2 shows the trends of the total number of bytes transmitted as a function of the duration of the experiment. When we account for the level of detail, we can effectively reduce the network traffic using the proposed distributed approach as opposed to the centralised approach. Interestingly, when the level of detail is not used, i.e. the visualisation data are exchanged amongst clients regardless their distance, the centralised approach generates less network traffic than the distributed one. This happens because, although the centralised approach is used, the level of detail strategy led to an effective reduction of bytes when clients are distant from each other. For example, if client A is located in L3 with respect to B, B will request visualisation data using the NO LOD distributed strategy, whereas with a LOD centralised strategy no data will be requested.

The oscillations that are visible in NO LOD strategies are due situations where clients are distant from each other in the VR space and data transmissions are reduced or even absent.

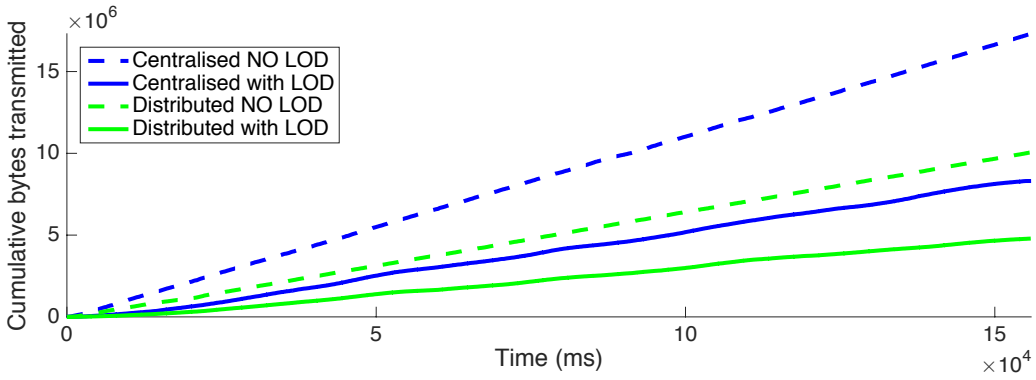


Figure 4.2: Cumulative network traffic measured in bytes in the case of seven clients. Centralised and distributed approaches that consider level of detail (with LOD) and that do not consider it (NO LOD) are illustrated.

4.2.2 Transmission latency

In this experiment we evaluated the latency of the data transmission. We quantified the latency as the delay measured by a client to send the request to another client and to receive the data. We tested scenarios with three, five and seven clients connected via wireless. We used Google Nexus 5x, Huawei Honor 8 and Nvidia Shield for the scenario with three clients, these three devices plus two Huawei P10+ for the scenario with five clients and these five devices plus two computers for the scenario with seven devices. Nvidia Shield was used as the host in all scenarios.

Fig. 4.3 shows the average latency as a function of the number of clients corresponding to each scenario. The exact values of average and standard deviation are reported in the legend of the graph.

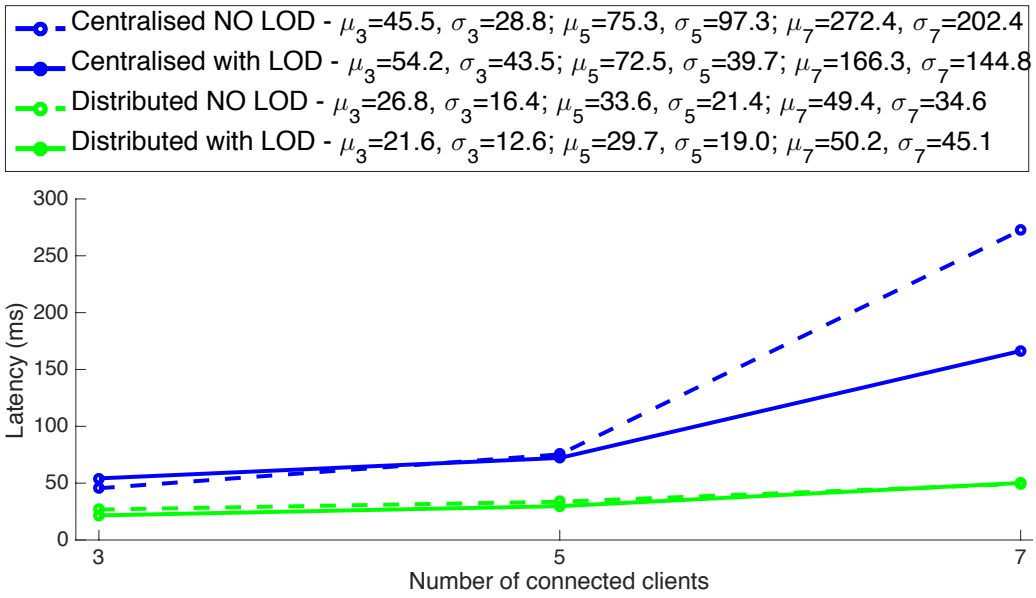


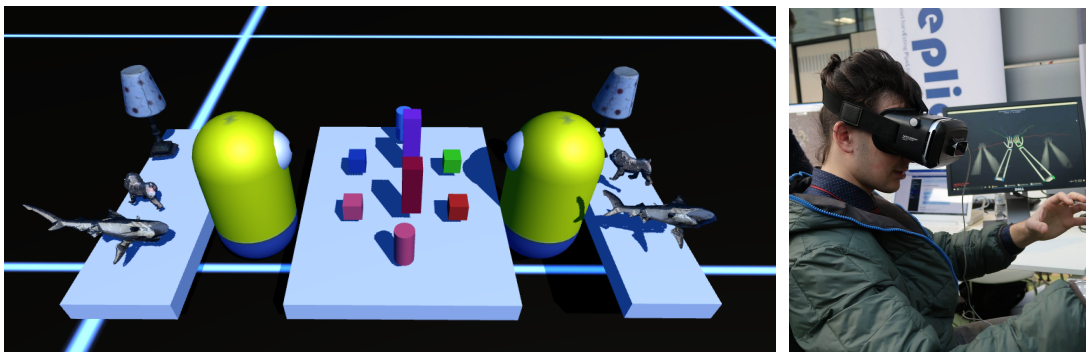
Figure 4.3: Average latency as a function of the number of clients corresponding to centralised and distributed approaches that consider level of detail (with LOD) and that do not consider it (NO LOD). The network latency is the average of latency measurements collected over a period of three minutes.

The average latency measured when visualisation data are exchanged with the distributed strategy is smaller than the latency measured with the centralised strategy. The use of level of detail reduces in general the latency. We can observe that the variance is in general fairly large and this is due to background threads of the devices that delay the processing of the received packets.

4.3 Qualitative analysis

In this section we will discuss the experience gained during the application test at the FBK's 'Open Day' event. In this initiative we were able to have the prototype tested by workers and researchers of the foundation, of different ages and without previous experience on the prototype. Each user, after a brief explanation of the application and the possibilities offered, was able to try the prototype for a duration of 5 minutes. In overall, the demonstration lasted more than an hour. The demonstration created for this event included two seats where two people could sit. Each user had its own viewer in conjunction with a smartphone and a Leap motion connected to a computer. In the VR environment, a table was created where the two users were placed at the ends. The demonstration offered the opportunity for two people to interact in the same game world, exchanging objects and being able to position them at will over the virtual table. In this demonstration it was not possible to move around the game world, so it was necessary to ask the other user to take a certain object in case it was not reachable. Figure 4.4 shows the VR environment used for the demonstration and a user trying the application.

Users were impressed by the naturalness and faithful reconstruction of their hands. After a short initial moment used to look around in the VR environment, users immediately started using their hands to interact with objects. Although most were able to interact with them without great difficulty, some users had difficulties. Some of these users have often tried to take objects as they would be taken in the real world, i. e. not using the pinch gestures. This made us understand the need for more natural interactions, or the possibility of introducing an initial tutorial to notify players of the gestures to use. Another reason for discussion was the area reached by the hand in the virtual world. Some users have in fact had difficulty in understanding the proximity of an object to their position in the virtual world, and have therefore continuously tried to take an object too far away from them, without success. Higher graphic fidelity or a different feedback system within the VR experience could address this problem. For example, if an user is trying to take an object too far away, this object could have a red border, to indicate that it is not reachable. In addition to interacting with objects, users



(a) (b)

Figure 4.4: FBK's 'Open Day' demonstration. (a) VR environment created for the demonstration. (b) User trying the application for the first time.

were impressed by the possibility of seeing the hands of the other player. Interactions based on hand movement such as greetings or other forms of sign communication were frequent. As for the more technical aspects, the connection between the two instances of the prototype did not give any problems and lasted more than one hour. The only disconnection problem occurred with the connection from the Leap Motion to the smartphone, which was solved by an automatic restart of the software. The overall duration of this test can therefore be compared to a typical gaming session, although there is no feedback on the actual sensations caused by such a duration to a single user.

In conclusion, the results obtained from this demonstration are generally positive. The robustness and immediacy of the prototype have been validated, while numerous feedback has been gathered on how to improve interactions and make them more natural.

4.4 Discussion

From the results obtained we can now answer our research questions. Is a low cost multi-user VR application feasible on mobile devices? The results obtained show that it is indeed possible, although with some limitations. They demonstrate the feasibility, for about ten players connected to the same local network, of playing with each other using their smartphones. The experience, even if possible, is certainly not ideal:

Performance: A VR application of this kind requires great fluidity to avoid discomfort to the player. It is very difficult for smartphones to maintain a stable level of performance, given the problem of termal throttling and battery power. Compared to high-end counterparts, the ideal duration of gaming sessions is considerably shorter. The release of mobile devices that are increasingly powerful and adapted to AR/VR experiences could reduce this disadvantage in the future.

Leap Motion connection: Although the Leap Motion device provides an unprecedented level of immersion, it currently has complications for use in a mobile environment. Given the need for a wired connection with a computer, this setup is currently suitable for research purposes but not ideal in a consumer environment. The release of a version natively compatible with mobile devices may solve this limitation.

Despite this, the prototype produced represents a good compromise compared to high-end VR experiences, which would require a high-performance computer and a high cost viewer. It makes a VR reality available to everyone, even if limited.

4.5 Summary

In this section we presented and discussed some experiments carried out on the prototype in different contexts. These experiments served to better understand the quality of the prototype and its limitations. The results obtained demonstrate the feasibility of achieving such an application in mobile

devices, albeit with moderate limitations compared to a high end experience.

5 Conclusion and future work

This chapter briefly summaries the thesis and its conclusions. Possible future work is also presented.

5.1 Conclusion

In this thesis, we wanted to study the possibility of creating a VR multi-user application at low cost and accessible to everyone. For this purpose, smartphones and Google Cardboard technology were used. The Leap Motion hand tracking device was used to make the VR experience immersive and accessible even to those who have no previous experience with controllers, while keeping the total cost of the application low. The use of this device allows for gesture-based interactions. In this proof of concept we wanted to implement some essential features for applications of this type, in order to create a possible starting point for applications from different domains. Therefore, we have implemented movement within the VR environment and the possibility of interaction with the world or with other players through the use of their own hands. Specifically, we have implemented the possibility of manipulating virtual objects and the ability to see the exact shape of each user's hands, to create interactions based on sign language. We analysed and discussed some of the technologies used to implement the prototype, such as the Unity3D game engine and Leap Motion software. We then listed and presented the features implemented and discussed their implementation. We also presented the system designed to transmit a large amount of data in an optimised way, to allow a detailed reconstruction of each player's hands in real time. When compared to a traditional data transmission method, the system created has reduced bandwidth usage, reduced latency and is more scalable, allowing more users to partecipate in the same VR environment. We have performed tests in real scenarios and simulations to verify the actual validity and effectiveness of the prototype. The results obtained have demonstrated the possibility of using the prototype by at least seven players. However, the prototype produced has strong limitations when compared to high-end VR applications on computers, due to the fact of using smartphones. The high battery consumption, the impossibility of using the Leap motion device natively and the low computational power render the prototype a possibility, but not an ideal application.

5.2 Future work

There could be numerous developments and future works for our thesis. Some of them are listed as follows:

1. **Development in various domains**: The prototype produced serves as a basis for many possible similar applications in different domains. Think of an application whose purpose is to improve the brainstorming or product design phase. We could create a collaborative environment where objects can be built. Another development could be in the area of videoconferencing. With the possibility of interacting with three-dimensional objects, numerous possibilities would open up. Think also of the construction sector, where it would be possible to explore and interact with a building still under construction. In the furniture sector, a user could, from home, view the various pieces of furniture and interact with them. In general, many industries could benefit from an application accessible to all that would allow such a level of interaction.

2. **Spatial audio**: To improve the interaction between users, we could implement a voice chat that allows you to communicate also by voice. In addition, as many VR experiences already offer, we could use 3D spatialization technology to deliver spatial audio in order to increase the immersion of the users in the VR environment.

3. **Leap Motion mobile support**: When mobile device support is released for the Leap Motion

device, we could integrate and use this technology without the need for a computer, greatly reducing latency and complexity of use.

4. **Testing and optimization for internet sessions**: We could understand in depth how the prototype works on the Internet by simulating games sessions on a non-local network. In addition, further optimizations could be studied to make the gaming experience better.

Bibliography

[1] Ahmed, D. T. and Shirmohammadi, S. "A Dynamic Area of Interest Management and Collaboration Model for P2P MMOGs". In: *Proc. of IEEE Symposium on Distributed Simulation and Real-Time Applications*. Vancouver, CAN, Oct. 2008.

[2] Akiduki, H. et al. "Visual-vestibular conflict induced by virtual reality in humans". In: *Neuroscience Letters* 340.3 (2003), pp. 197–200.

[3] Anthes, C. et al. "State of the Art of Virtual Reality Technologies". In: *2016 IEEE Aerospace Conference* (Mar. 2016).

[4] Bassily, D. et al. "Intuitive and Adaptive Robotic Arm Manipulation using the Leap Motion Controller". In: 2014, pp. 1–7.

[5] Carlini, E., Ricci, L., and Coppola, M. "Reducing server load in MMOG via P2P Gossip". In: *Proc. of Workshop on Network and Systems Support for Games*. Venice, IT, Nov. 2012.

[6] Choi, S., Jung, K., and Noh, S. D. "Virtual reality applications in manufacturing industries: Past research, present findings, and future directions". In: *Concurrent Engineering* 23.1 (Mar. 2015), pp. 40–63.

[7] Cruz-Neira, C. et al. "The CAVE: Audio Visual Experience Automatic Virtual Environment". In: *Commun. ACM* 35.6 (1992), pp. 64–72.

[8] Davis, J., Hsieh, Y., and Lee, H. "Humans perceive flicker artifacts at 500Hz". In: *Scientific Reports* 5 (Feb. 2015), pp. 2706–2713.

[9] Freina, L. and Ott, M. "A Literature Review on Immersive Virtual Reality in Education: State Of The Art and Perspectives". In: *Proc. of eLearning and Software for Education*. Bucharest, RO, Apr. 2015.

[10] Gobbetti, E. and Scateni, R. "Virtual reality: Past, present and future". In: *Studies in health technology and informatics* 58 (Feb. 1998), pp. 3–20.

[11] Greenwald, S. W., Corning, W., and Maes, P. "Multi-User Framework for Collaboration and Co-Creation in Virtual Reality". In: *Proc. of Conference on Computer Supported Collaborative Learning*. Philadelphia, US, June 2017.

[12] Guna, J. et al. "An analysis of the precision and reliability of the Leap Motion sensor and its suitability for static and dynamic tracking". In: *Sensors* 14.2 (Feb. 2014), pp. 3702–3720.

[13] Hilfert, T. and Konig, M. "Virtual reality applications in manufacturing industries: Past research, present findings, and future directions". In: *Visualization in Engineering* 4.3 (Dec. 2016), pp. 1–18.

[16] Khademi, M. et al. "Free-hand interaction with leap motion controller for stroke rehabilitation". In: *Proc. of Human Factors in Computing Systems*. Toronto, ON, May 2014.

[14] Kim, K., Yeom, I., and Lee, J. "HYMS: A Hybrid MMOG Server Architecture". In: *IEICE Trans. on Information and Systems* 87-D (Jan. 2004), pp. 2706–2713.

[15] Lin, J. J. W. et al. "Effects of field of view on presence, enjoyment, memory, and simulator sickness in a virtual environment". In: *Proceedings IEEE Virtual Reality 2002* (2002), pp. 164–171.

[17] McMahan, R. P. et al. “Evaluating natural interaction techniques in video games”. In: *2010 IEEE Symposium on 3D User Interfaces*. Waltham, US, Mar. 2010.

[18] Mohandes, M., Aliyu, S., and Deriche, M. “Arabic sign language recognition using the leap motion controller”. In: 2014, pp. 960–965.

[19] Pani, M. and Poiesi, F. Distributed data exchange with Leap Motion. submitted. 2018.

[20] Pausch, R., Crea, T., and Conway, M. “A Literature Survey for Virtual Environments: Military Flight Simulator Visual Systems and Simulator Sickness”. In: *Presence: Teleoper. Virtual Environ.* 1.3 (July 1992).

[21] Pretto, N. and Poiesi, F. “Towards gesture-based multi-user interactions in collaborative virtual environments”. In: *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci., XLII-2/W8*. Hamburg, GE, Nov. 2017, pp. 203–208.

[22] Sahin, O. and Coskun, A. K. “On the Impacts of Greedy Thermal Management in Mobile Devices”. In: *IEEE Embedded Systems Letters* 7.2 (2015), pp. 55–58.

[23] Schmalstieg, D. and Gervautz, M. “Demand-driven geometry transmission for distributed virtual environment”. In: *Computer Graphics Forum* 15.3 (Aug. 1996), pp. 421–431.

[24] Bruno Kessler Foundation. https://www.fbk.eu/it/. accessed: Feb 2018.

[25] Google Cardboard Manufacturers Kit. https://vr.google.com/cardboard/manufacturers/. accessed: Feb 2018.

[26] Google Daydream. https://vr.google.com/daydream/. accessed: Feb 2018.

[27] Google Daydream Standalone HMD. https://vr.google.com/daydream/standalonevr/. accessed: Feb 2018.

[28] HTC Vive HMD. https://www.vive.com/us/. accessed: Feb 2018.

[29] Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization. http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf. accessed: Feb 2018.

[30] Leap Motion. leapmotion.com. accessed: Feb 2018.

[31] Leap Motion framerate. https://forums.leapmotion.com/t/sample-period-frames-frequency/3281. accessed: Feb 2018.

[32] Leap Motion latency. blog.leapmotion.com/understanding-latency-part-1. accessed: Feb 2018.

[33] Leap Motion mobile support. http://blog.leapmotion.com/mobile-platform/. accessed: Feb 2018.

[34] Leap Motion software V2. https://developer.leapmotion.com/sdk/v2/. accessed: Feb 2018.

[35] Leap Motion software version Orion (Beta). https://developer.leapmotion.com/get-started. accessed: Feb 2018.

[36] Leap Motion Software version Orion SDK. https://developer.leapmotion.com/documentation/csharp/index.html. accessed: Feb 2018.

[37] Oculus Rift HMD. https://www.oculus.com/. accessed: Feb 2018.

[38] Samsung Gear VR specifications. https://www.oculus.com/blog/introducing-the-samsung-gear-vr-innovator-edition/. accessed: Feb 2018.

[39] Second Life. secondlife.com. accessed: Feb 2018.

[40] Unity3D. unity3d.com. accessed: Feb 2018.

[41] Unity3D High Level API. docs.unity3d.com/Manual/UNetUsingHLAPI.html. accessed: Feb 2018.

[42] Unity3D Multiplayer. https://docs.unity3d.com/Manual/UNet.html. accessed: Feb 2018.

[43] Unity3D Personal license. https://store.unity.com/products/unity-personal. accessed: Feb 2018.

[44] Unity3D Popularity Survey. https://unity3d.com/public-relations. accessed: Feb 2018.

[45] VibeHub. youtu.be/azUXUr6rWSc. accessed: Feb 2018.