Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Scienze di Internet

IL SENSO DI PRESENZA NELLE REALTÀ VIRTUALI VISSUTE TRAMITE HMD: L'INTERAZIONE E LE INTERFACCE

Relatore:
Chiar.mo Prof.
Marco Roccetti

Presentata da:
Lorenzo Casanova

Sessione III
Anno Accademico 2013/2014

Abstract

Grazie alla crescente evoluzione tecnologica è oggi possibile, tramite Head Mounted Display (HMD), vivere una realtà virtuale ricca nei dettagli, interattiva ed immersiva.
L'avanzamento in questo settore ha infatti portato a una vera e propria rivoluzione, aprendo la possibilità di utilizzare questa tecnologia in molteplici ambiti.
L'ostacolo riscontrato è che a un progresso di tale entità non si associa un adeguato aggiornamento e perfezionamento riguardo alle metodologie di interazione con oggetti 3D, dell'utilizzo di interfacce grafiche e del generale design ambientale.
La diretta conseguenza di questo mancato aggiornamento è quella di indebolire o addirittura annullare l'effetto presenza dell'HMD, requisito indispensabile che consente all'utente di immergersi sensorialmente nel contesto simulato.
L'obiettivo di questo studio consiste nel comprendere cosa è necessario tenere in considerazione e quali regole vanno cambiate per poter mantenere un'alta sensazione di presenza per l'utente all'interno di una realtà virtuale.
A questo scopo è stato creato un ambiente virtuale 3D in grado di supportare l'utilizzo di un HMD, l'Oculus Rift, e di diversi dispositivi di input in grado di consentire controllo tramite movimenti naturali, il Razer Hydra ed il Leap Motion, in modo da poter effettuare un'analisi diretta sul livello del fattore presenza percepito nell'effettuare diverse interazioni con l'ambiente virtuale e le interfacce grafiche attraverso questi dispositivi.
Questa analisi ha portato all'individuazione di molteplici aspetti in queste tipologie di interazioni e di design di intrefacce utente che, pur essendo di uso comune negli ambienti 3D contemporanei, se vissuti in un contesto di realtà virtuale non risultano più funzionali e indeboliscono il senso di presenza percepito dall'utente.
Per ognuno di questi aspetti è stata proposta ed implementata una soluzione alternativa (basata su concetti teorici quali Natural Mapping, Diegesis, Affordance, Flow) in grado di risultare funzionale anche in un contesto di realtà virtuale e di garantire una forte sensazione di presenza all'utente.
Il risultato finale di questo studio sono quindi nuovi metodi di design di ambienti virtuali per realtà aumentata.
Questi metodi hanno permesso la creazione di un ambiente virtuale 3D pensato per essere vissuto tramite HMD dove l'utente è in grado di utilizzare movimenti naturali per interagire con oggetti 3D ed operare interfacce grafiche.

Contents

List of Figures

1 Introduction

Virtual Reality technology showed great promise and was of great interest to a wide array of disciplines during the late 1980s and early 90s.
Though at the times it seemed as virtual reality technology would quickly diffuse into homes, it never materialized, mainly because the technology level was still too immature to allow the production of sufficiently good devices.

However, out of this initial interest in this technology came a strong interest in features of video games that create the sense for the player of really being in a virtual space, and its varying dimensions.
This interest has led to a great deal of work on the concept of **"presence"**, as it relates to video games and an emerging interest in control mapping.
Despite increased scholarship on the concept of presence and related factors such as control type or mapping there is very little current research concerning presence and mapping as it directly relates to virtual reality technologies.
However, as of today, technology has finally advanced to the point where creating a more advanced virtual reality headsets is possible.
A new generation of virtual reality technology, most notably the Oculus Rift, is close to being released to the public and ultimately the global interest in the general concept of virtual reality is being revitalized.

Therefore, the main focus of this study is to examine this new generation of virtual reality technology and to discover how different control types and input devices can be paired with this kind of technology, and ultimately used to experience and interact with in-game 3D objects and graphical user interfaces to see what combination is the best at keeping an high level of player's "presence" and, more generally, what precautions must be taken into account to maintain this level high enough.

This will be done using the free version of the Game Engine *Unity3D* to create a "test area" where, different hardware input devices support, different kind of graphical user interfaces and different possible gameplay actions will be implemented and tested thorugh the Oculus Rift.

This will allow to understand what type of controller and user interface is better suited in different scenarios with this technology, what of previous design methods still works with this new technology and what must be changed and, ultimately,how we can maintain an high level of "presence" sensation on the user experiencing the virtual reality.

2 The concept of “Virtual Reality”

Virtual Reality can be simply seen as a computer-simulated virtual environment that can simulate physical presence in places in the real world or imagined worlds.

Another point of view, though, shows that this technology can be seen as a tool by which humans can directly interact with computers in order to solve far more complex problems than by using strictly traditional interface methods.
It’s an high-end user-computer interface that involves real time simulation and interactions through multiple sensorial channels: at the moment these sensorial modalities are visual, auditory and, partially, tactile.

One of the most important elements about virtual realities is the ability of the software on emulating realistic environments. This is the area in which, in the last decade, video games progressed the most.
Developers now have the ability to create stunningly realistic worlds populated with artificial intelligences who behave in believable manners.
These simulated worlds can be similar to the real world in order to create a lifelike experience (e.g. in simulations for pilot or combat training) or they can differs significantly from reality, but always maintaining a believable experience.
But what are the parameters able to define a virtual reality?

Steuer[1], shows a good analysis of this point by defining description for his concept of “telepresence”, but this can be very well applied today as general factors for a convincing virtual environment.

Following his analysis these factors are:

- **Vividness:** this means the representational richness of a mediated environment as defined by its formal features, that is, the way in which an environment presents information to the senses.
  Vividness is stimulus driven, depending entirely upon technical characteristics of a medium.
  - **Breadth:** this is a function of the ability of a virtual reality device to present information across the senses (which can be orienting, auditory, haptic, taste-smell and visual).
    Inputs to several of these systems from a single source can be considered informationally equivalent.
    For example, a book is less vivid than a film due to a lower “breadth”, since a film stimulates vision and hearing simultaneously.

  - **Depth:** this concept can be described in terms of quality: an image with greater depth is generally perceived as being of higher quality than one of lesser depth; the same is true for auditory representation.
    Informationally, depth depends directly upon the amount of data encoded and the data bandwidth of the transmission channel.
    In real-world perception, depth is taken for granted, as our body's sensory mechanisms almost always operate at full bandwidth.

- **Interactivity:** this is defined as the extent to which users can participate in modifying the form and content of a virtual environment in real time.

  - **Speed:** this refers to the rate at which input can be assimilated into the virtual environment; nowadays this factor can be measured with few milliseconds, but in some cases (and with some input devices) this can still be a notable (and thus immersion-breaking) factor.
  - **Range:** this is determined by the number of attributes of the mediated environment that can be manipulated and by the amount of variation possible within each attribute.
    In other words, range refers to the amount of change that can be effected on the virtual environment.
    This depends on how the virtual environment is designed.
  - **Mapping:** this refers to the way in which human actions are connected to actions within a virtual environment.
    At one extreme, these mappings can be completely arbitrary and unrelated to the function performed, thus considered not natural-mapping[26].
    This is highly depending on two factors: the input devices used, and how the virtual environment was designed to use them.

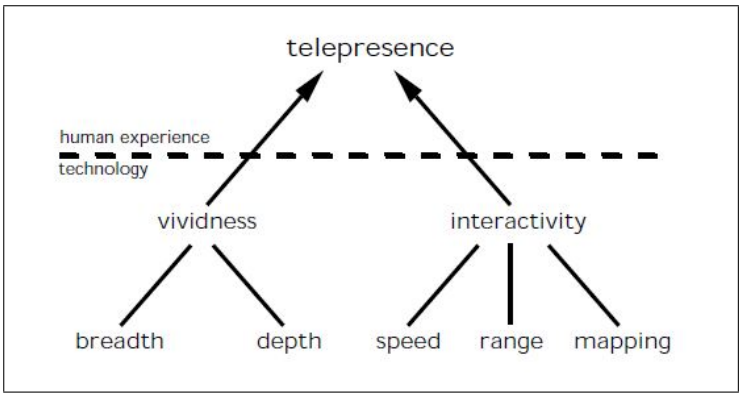

Figure 1:
Factors defining telepresence

Thanks to this conceptualization, it’s now clear that the concept of “Virtual Reality” represents a multi-sensory experience for the final user, and the technology used to achieve it simply defines how “convincing” this experience will be.

We now need to understand what “convincing” means, and how we can maintain an high virtual reality immersion-level for the final user.

3 Immersion and Presence

As we have seen, virtual reality technology allows users to have unique, different experiences, which can represent real worlds experiences as much as imaginary world experiences.
The primary characteristic distinguishing these recreated virtual environments experienced through virtual reality technology from other means of displaying information is the focus on **immersion**.

In a technical acceptation of the term, immersion is achieved by removing as many real world sensations as possible, and substituting these with the sensations corresponding to the virtual environment.
Immersion is by essence related to the multi-modal nature of the perceptual senses, and also to the interactive aspects of a virtual reality experience[5].

From this viewpoint, immersion is intuitively related to the resemblance of the virtual reality devices with human characteristics.
These include the size of the human visual field, the stereoscopic aspects of the simulation, the surround aspects of the sound, that is the extent to which the computer displays are extensive, surrounding, inclusive, vivid and matching. The term "immersion" thus stands for what the technology delivers from an objective point of view.

Lately, the concept of immersion have been of great interest to the games industry. It has also drawn the attention of researchers under the name of **"presence"**.
Because there is a large amount of presence research across various fields and focuses, the exact definition of the concept is still unclear, and we can find a lot of different definitions:

- Lombard and Ditton[2] describe a multifaceted concept, but it all ties in to the idea that presence is the perceptual illusion of non-mediation.
- Hartmann, Klimmt and Vorderer[8] call presence an umbrella term used to describe a variety of experiences. The commonality of these experiences is that the user is either less aware or completely unaware of the mediated nature of the experience.
- Westerman and Skalski[11] describe presence as a sense of "being there" which is caused by media technology, or a perceptual illusion of non-mediation.
- Tamborini and Bowman[10] describe presence as a multidimensional psychological state in which a user's experience in a virtual space is shaped by the technological features of that space that are not immediately apparent to the user.

The common thread through the literature seems to be that while presence is a complex and multidimensional concept, it can be generally described as a sense of "being there". A more in depth definition would be that a sense of presence is the illusion of non-mediation, which means the user of some technology believes they are actually in a virtual space, and they fail to account for the fact that they are actually interacting with a piece of technology.

So, what is the difference between immersion and presence?

Generally speaking, the more that a system delivers displays (in all sensory modalities) and tracking that preserves fidelity in relation to their equivalent real-world sensory modalities, the more that it can be considered "immersive".
From this technological standpoint, immersion is intended to instill a sense of belief that one has left the real world and is now "present" in the virtual environment.
This notion of being present in the virtual world has been considered central to virtual environments.

Thus, whereas immersion is a technology-related objective aspect of virtual environments, presence is a psychological, perceptual and cognitive consequence of immersion.

In conclusion, the presence factor is thought of as the psychological perception of "being in" or "existing in" the virtual environment in which the user is immersed.

3.1 Types of "presence"

Researchers have identified several distinct dimensions of presence.
There is some variation in the actual terminology but the dimensions are typically listed as[6]:

- **Spatial presence**
- **Social presence**
- **Self presence**

Spatial presence is understood as the sense of actually being physically located in a virtual environment.
This can be achieved with high-fidelity rendering devices and well-designed virtual environments.

An example can be seen in the image below of the free to play game “War Thunder”[1] played through the Oculus Rift: the user is placed inside the cockpit of his plane, and thanks to the head mounted display he’s able to look around in a natural way and perceive space and distance of surrounding world and objects (landscape, buildings, other planes...) much better than he would have done with a standard monitor.

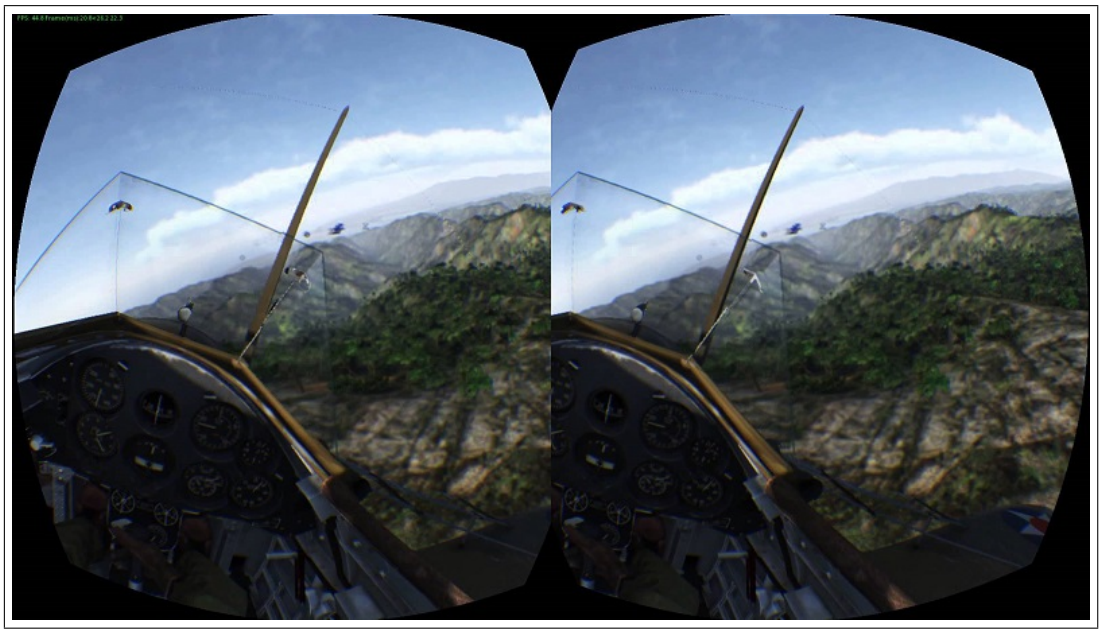

Figure 2:
The Game WarThunder, as seen through the Oculus Rift

Social presence is the idea that the user perceives the people in a mediated experience as actual human beings.
This is more prominent in massively multiplayer online games (MMOGs) where most virtual characters one encounters are controlled by other physical players.
This can be achieved giving players an high number of methods for interact with each other, to appear different from each other and to move (avatars animations) differently from each other.
That is because we perceive repetitions (such as same-looking characters, or same and synchronized walking animations on multiple characters) as something “artificial”, so in

[1]Free game downloadable at url: *http://warthunder.com*

order to convince a player of being surrounded by other human players, it's not sufficient to know that other avatars are controlled by real players, but they must also "look and feel" enough different from each other (like in real world).
An example of social presence is the software "Riftmax Theater 4D"[2]: in this software different people can join (or create) a server which will be a virtual place where different players can talk and interact with each other while experiencing and/or actively participating in a shared experience (like watching a movie in a cinema, having a karaoke night, participating at a talk show and so on).

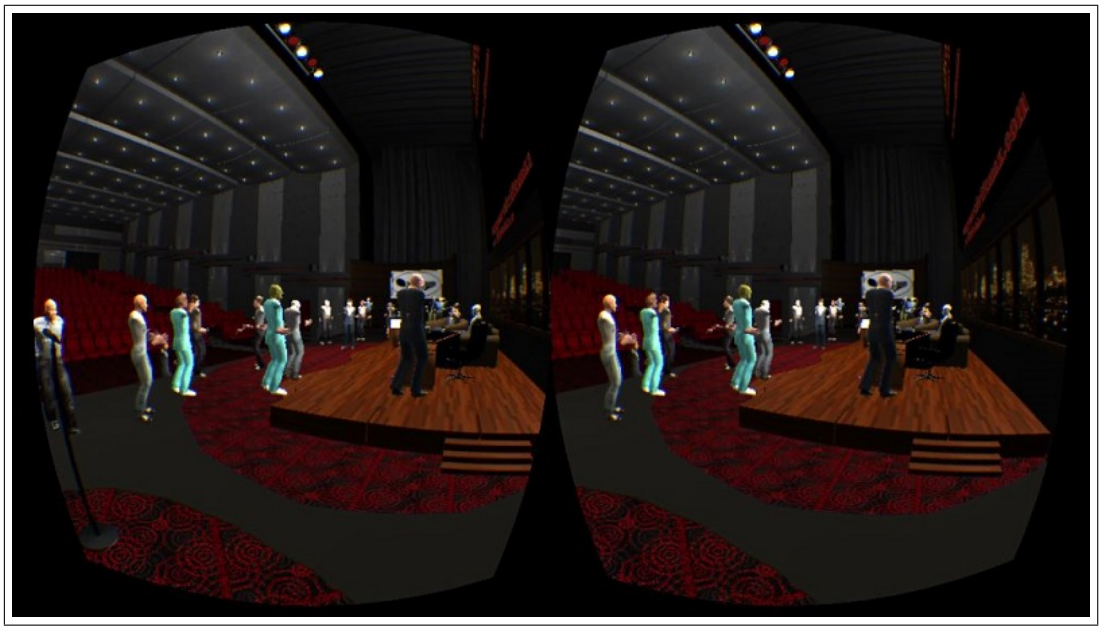

Figure 3:
The game Riftmax Theater 4D, as seen through the Oculus Rift, taking place in a movie theater. All avatars are actual human players

Self presence is the idea that the user is actually their avatar in the virtual world he's in.
This can be achieved in multiple ways, such as giving him a virtual body he can look at, giving him enough possible interaction with the virtual environment, allowing him to move and perform actions in a natural way, and so on.

[2]Available for download at url: *http://www.riftmax.com/*

The image below shows an oculus user with his avatar: thanks to the input device Hydra (see section 7.3 at page 45) he can control his avatar arms with natural movement.

Figure 4:
A virtual avatar controlled with an Hydra device.

3.2 Achieving “presence”

The important thing concerning presence is that no matter the media (videogame, film, real-like or imaginary virtual environment), if the person interacting with the media feels like they are actually in whatever place they are observing, then presence has been achieved.
But how to increase the level of presence of the user?

We’ve already got to the conclusion that presence is a psychological consequence of the immersion factor delivered to the user thanks to the technology level of the devices that are being used.
So, to improve the final presence, a good course of action is to first improve the immersion factor, and to improve this we can start by improving the devices.
According to Oculus VR, the society behind the Oculus Rift headset, the technology requirements to achieve presence in virtual reality are low-latency and precise tracking

of movements.
On the other hand, *Michael Abrash* gave a talk on *VR at Steam Dev Days* in 2014[3]: according to him and his team (the virtual reality research team at *Valve*), all of the following are needed, otherwise presence cannot be achieved:

- A wide field of view (80 degrees or better)
- Adequate resolution (1080p or better)
- Low pixel persistence (3 ms or less)
- A high enough refresh rate (at least 60 Hz, 95 Hz is enough but less may be adequate)
- Global display where all pixels are illuminated simultaneously (rolling display may work with eye tracking)
- Optics (at most two lenses per eye with trade-offs, ideal optics not practical using current technology)
- Optical calibration
- Rock-solid tracking - translation with millimeter accuracy or better, orientation with quarter degree accuracy or better, and volume of 1.5 meter or more on a side
- Low latency (20 ms motion to last photon, 25 ms may be good enough)

This, however, shouldn't be a problem since the general technological level of devices is always progressing and, specifically, *Oculus VR* is about to release his final model product, specifically designed to deliver the presence factor at his final users (see section 7.1 at page 30).

Increasing the technological level of devices (thus increasing the immersion factor) is not the only way to increase presence.
Being able to perform various actions inside a virtual environment and how these actions are being executed is a key component regarding the user's presence factor.

[3]Video of this speech available at url: *https://www.youtube.com/watch?v=G-2dQoeqVVo*

3.3 The use of "presence"

Virtual Reality is potentially the biggest transformation in our relationship with technology since the personal computer, and possibly much more: it promises to finally allow us to interact with the information in the way we're built to interact with reality.

Traditional media presents images and sounds that are a description of experiences.
Done properly, Virtual Reality presents experiences directly, in the way that our bodies have evolved to accept information: by experiencing it.

To cite Michael Abrash on one of his speech on *Steamworks Development*[4], the reality we experience is created inside our brain thanks to the perception of our surroundings, and the feel of *presence* is a simple perceptual phenomena: since a Virtual Reality headset like the Oculus Rift is able to "fool" our body sensors, our brain is tricked to believe that the Virtual Reality is the real reality on a subconscious level, even if we are fully consciously aware that it's not.
This works so well that studies[14] have shown that thanks to the presence factor, the virtual environments became so real to be able to transmit high degrees of stress an fear to user exposed in threatening virtual environments, even if the threats were predicted: that's because the users were actually experiencing them, which wouldn't be possible with any other media.
So presence factor is unique to Virtual Reality, and it will likely be the key for its success.

Thanks to this, Virtual Reality is going to be useful in more and more applications:

Training: the presence factor allows user to really feel the virtual environment, and this brings virtual training effectiveness to the highest levels possible.
Training in virtual environments enable users to develop complex skill in safe, isolated and totally controlled virtual environments.
This technology has been already adopted by the military (and this includes all three services – army, navy and air force), by general healthcare and it can be applied to any kind of training.

Movies: filming for virtual reality means a new different approach not only on the technical level, but under every aspect.
The viewer, while enjoying the movie, will actually "be" inside the movie itself, free to look where it wants and even move away. At the moment, Oculus have founded *Story Studio*, a team dedicate solely to produce movies for the Oculus

[4]Speech titled "*What VR Could, Should, and Almost Certainly Will Be within Two Years*" available at url: *https://www.youtube.com/watch?v=G-2dQoeqVVo*

Rift, and right now even *Pixar* and *Warner Bros* are interested in the emerging field of movies in virtual reality.

Education: this is another area which has adopted virtual reality for teaching and learning situations. The advantage of this is that it enables large groups of students to interact with each other as well as within a three dimensional environment.
It is able to present complex data in an accessible way to students which is both fun and easy to learn.

Healthcare: is one of the biggest adopters of virtual reality which encompasses surgery simulation, various phobia treatment, robotic surgery.
This is one of the area where virtual reality training gains more importance, since it allows healthcare professionals to learn new skills as well as refreshing existing ones in a safe environment, without the risk of causing any danger to the patients.

Heritage: virtual reality settings inside museums employ interaction as a means of communicating information to the general public in new and exciting ways.
There has been a move away from the traditional type of experience associated with museums, galleries and visitor centers.
The old model was that of passive engagement in which people viewed the exhibits but did not get involved to an experience in which interaction is the main feature.
Interactive displays form a large part of many exhibitions and particularly appeal to children. Children are often difficult to attract to a museum or gallery as they tend to see this as a boring experience.
But the use of interactive technologies such as virtual reality has changed that perception and opened up these spaces to a new audience.

Business: virtual reality is already being used in a number of ways by the business community, which include virtual tours of a business environment, training of new employees, a 360 "personal" view of a product.
Many businesses have embraced virtual reality as a cost effective way of developing a product or service.
For example it enables them to test a prototype without having to develop several versions of this which can be time consuming and expensive, plus it is a good way of detecting design problems at an early stage which can then be dealt with sooner rather than later.

Engineering: virtual reality engineering includes the use of 3D modeling tools and visualization techniques as part of the design process.
This technology enables engineers to view their project in 3D and gain a greater understanding of how it works, plus they can spot any flaws or potential risks before implementation.

This also allows the design team to observe their project within a safe environment and make changes as and where necessary, which saves both time and money.

Scientific Visualization: virtual reality is being increasingly used in the field of scientific visualization.
This field is based upon using virtual environments to express complex ideas and scientific concepts, for example molecular models or statistical results, in other word a tool for conveying complex information.

Construction: virtual reality can be extremely useful in the construction industry, which is often known as having a very high amount of inefficiency and low profit margins.
Using a virtual environment, an organization can not only render the resulting structure in 3D but also experience them as they would in the real world.

Gaming: this is obviously the first application that comes in mind: that’s because at present, virtual environments are used principally in the videogame industry. Achieving presense in videogames will ultimately revolution gameplay and possible interactions, all to the benefits of player’s enjoyment.

4 Concept

In order to analyze inputs and interfaces and before the test implementation, some literature studies are conducted as first step.
These theories will be used for the test implementation and for the final analysis in sections 9 and 10.

4.1 Engrossment and Engagement

Brown and Cairns describes[3] the road to an immersive experience, in a game context, as a three step process.
In order to create an that kind of experience, the game first has to *engage* and then *engross* the player.

Player engagement depend on the accessibility of the game itself: the complexity of the control scheme, the amount of investment and attention required from the player to fully master the game, as well as the game, in turn, constantly providing something worthy of attending to.
After the player has become engaged in a gaming experience there is the chance that the player develops an emotional connection to the game that goes well beyond the attention of the engagement phase.

In this stage, called engrossment, the player's emotions are directly affected by the game and the player will start to become less aware of his surroundings as well as becoming less self aware.
Full immersion is achieved when players have to commit (nearly) all their attention, their cognitive/perceptual systems, to playing the game.
Games that forces the player to pay great attention to both visual and auditory cues, thus using their own senses to interpret the game world much as in real life, are the most likely to approach a state of full immersion.

4.2 Diegesis

Genette states[23] that in the structuralist-linguistic understanding of narratives the term diegesis refers to the world in which the events of a story occur.

In film theory the term diegetic typically refers to the internal world created by the story that the film characters themselves experience, as well as any previous events or characters referred to in the story.
Film components like the musical score, narration (by a disembodied narrator) or subti-

tles are usually referred to as non-diegetic elements, since these can not be perceived by the characters within the story-world.
Consequentially, every component that can be perceived by the characters of the film is diegetic.
These concepts will be very useful in designing user interfaces coherent with the virtual environments.

4.3 Affordance Theory

Mateas and Stern talk[4] about the concept of affordances in a game world.
Material affordances can be defined as the things in the game world you can interact with and any effects that the player can have on the virtual world.
Formal affordances are defined as that which motivates the user and helps him choose from and prioritize between all the available actions.

To give an examples the authors praise the game *Quake* as finding a very good balance between material and formal affordances: given the plot and setting of the game the player can derive that everything that moves is trying to kill him, so he should try to kill everything, moving through as many levels as possible.
This set of formal affordances overlap well with the games material affordances: the player can pick up and fire different weapons, he can pick up objects that make him stronger, he can pick up keys and interact with doors, buttons, elevators and teleporters that will help him or her proceed through the levels.

4.4 Suspension of Disbelief

The literary term "suspension of disbelief" describes the willingness of an audience to forsake their knowledge of the real world for the sake of entertainment.

The concept was originally contrived[21] to explain how a modern, enlightened audience might continue to enjoy gothic, romantic and otherwise supernatural fiction.
Part of this concept is the idea of something called "the forth wall", an imagined boundary separating any fictional setting and its audience.

In games, having characters referring to save points or controller buttons would be a clear example of this kind of break since they recognize the existence of a player external to the game world.
Some thinks that these kinds of breaks can take players straight out of any immersive state, but others disagree pointing out that the player-game relationship is more complex than that of the actor-audience and that a forth wall in gaming is better viewed as a flexible membrane resistant to brakes.

This notion is similar to arguments made by theorists like Murray and Juul, pointing out[25] the dual nature of games in that they play out both in reality and in a fictional universe simultaneously and is dependent on "audience" (players) participation.
Dying in the game world is fictitious, but losing the game is real.

This concept was also criticized by Tolkien[27]: he argues that an author can bring the reader to experience an utterly strange world through the rigorous use of internal consistency and rationality alone.
The reader can enjoy these worlds without needing to suspend their knowledge of what is "real" and instead treats them as self contained worlds within their own minds.

More generally, we can state that in virtual environments the user must willingly accept "compromises" and layers of abstraction if he wants to maintain the sensation of presence, since no simulation is yet able to (or maybe wants to) perfectly mimic real life experiences.

4.5 Flow

Micheal Csíkszentmihályi[22] states that flow describes the mental state of operation in which a person is engaged in an activity by a feeling of energized focus, full involvement, and success or progress in the activity itself.

In order to induce flow, an activity needs to: have clear goals, require concentration and focus, provide immediate feedback, provide a good balance between challenge and skill level, provide a sense of full control and be intrinsically rewarding.

Characteristic of a flow inducing activity is that the person engaged experiences a loss of self-consciousness and sense of time as his or her awareness is narrowed down to the activity itself.

The relationship between the skill level and challenge level of a task as pertaining to flow is further described by figure 5: under general conditions, inside a virtual environment, we want to keep the user inside the highlighted area in the image.

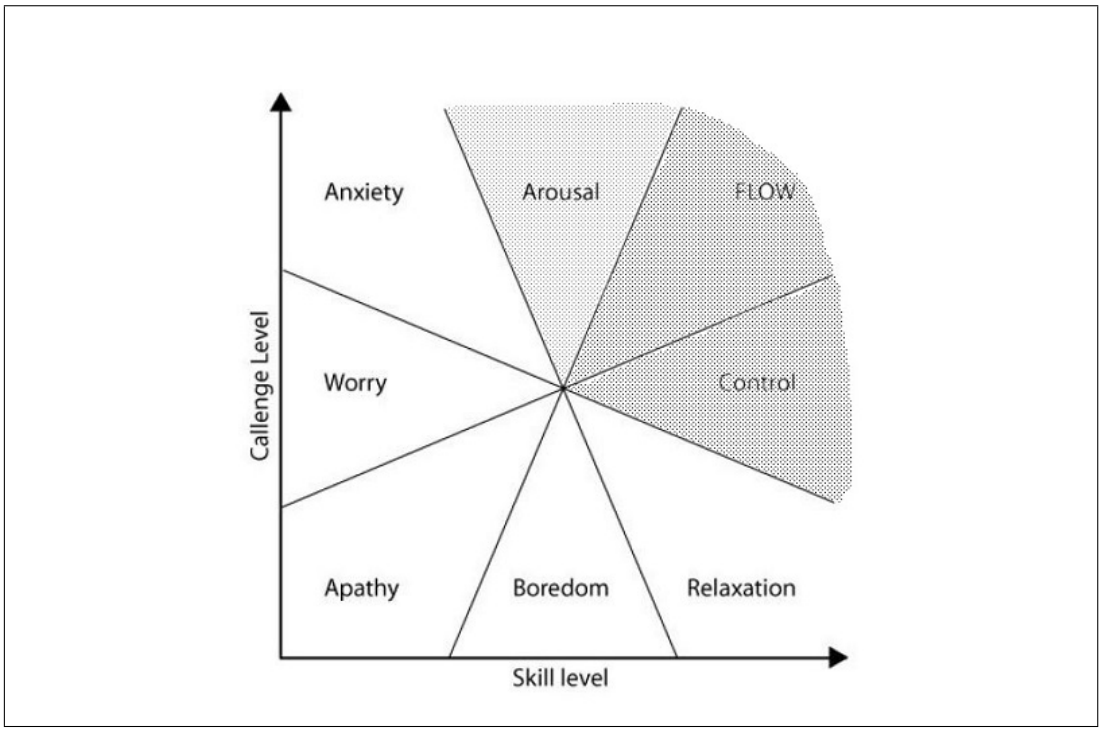

Figure 5:
Flow: relationship between skill and challenge

4.6 Design Methodology

John Chris Jones[24] describes a general design task as a three stage process consisting of:

- breaking the problem into pieces
- putting the pieces together in a new way
- testing the consequences of the new arrangement

He refers to these stages as *Divergence*, *Transformation* and *Convergence*.
He recommends diverging when the objective is tentative and the problem boundary is undefined: for this study this was very much the case, since presence delivered in virtual environments experienced through an headset and innovative input devices are relatively new and unexplored areas.

The transformation phase is supposed to identify critical aspects and constraints and set up realistic goals and subgoals of the design solution.
This is most often done by creating one or several solution patterns than can later converge into a final product.
It is important to remain flexible and free to change and redefine sub-goal throughout the transformation phase.

In the convergence phase the designers work to reduce the number of uncertainties and pick out the most promising design solutions.
The range of remaining alternatives should become more detailed, narrowing in on the solutions that best address the requirements identified earlier.
A modern design process is often a series of micro-iterations of these three distinct phases, with the result of one convergence phase feeding into a new divergence phase in order to gradually zero-in on the final design solutions.

This process was used to design and implement the test room, as we will see in sections 8 and 9.

5 Inputs in Virtual Reality

Research into the topic of game controls have found that control method has a large impact on player's enjoyment of games[12].
Skalski, Tamborini, Shelton, Buncher and Lindmark[9] also approach the topic of natural mapping in regards to video games.

5.1 Control Mapping

Norman[26] produced the seminal works concerning the ideas of natural mapping.
His work centers on how humans use control interfaces to achieve tasks.
Basically, it is the idea of using close analogues of what the user is attempting to accomplish when they interact with a control interface: at the two opposite we have a "full arbitrary mapping" and a "full natural mapping".
Norman states that the more the control mapping is close to the action it represents, the more the user will be able to execute that action without thinking about the process of controlling this action.

We can make a simple example: we have a virtual environment where we impersonate and move an avatar in first person view.
How do we move it?

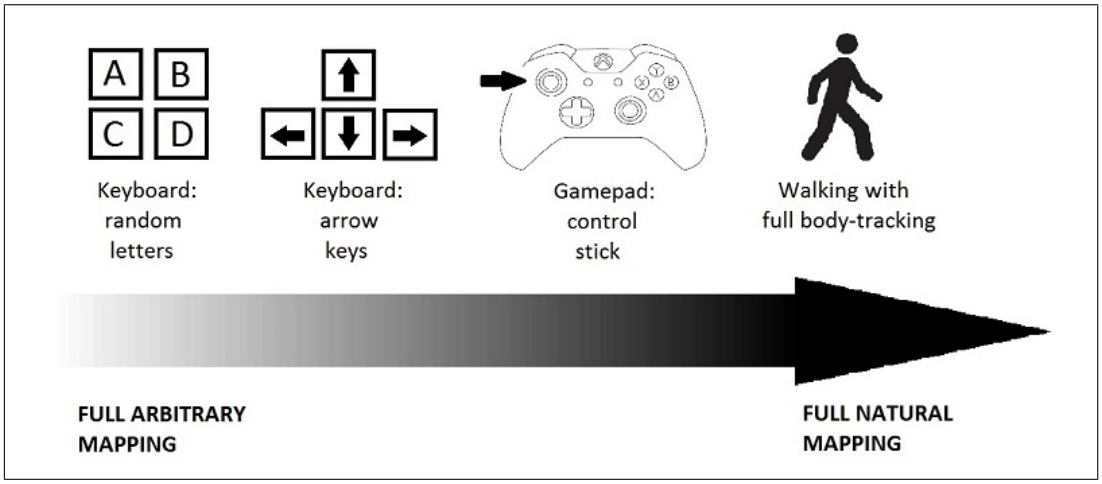

Figure 6:
Arbitrary and natural controls mapping in moving an avatar

Figure 6 shows a few option to move our character:

- **Keyboard with random letters:**
  this is obviously the worst control method for moving a character.
  The inputs reflects nothing about the action they controls, and the user who is about to walk must first "translate" his goal in the required actions sequence on the controls to achieve it.

- **Keyboard with arrows keys:**
  this mapping scheme has a natural component, the allocation of the keys.
  The up arrow can move the character forward, the back arrow backward and so on.
  This allows the user to remember the action performed by the key simply by touching it (thanks to his position relative to other keys), shortening a lot the "translation" process done with random keys.

- **Control stick of gamepad:**
  this control scheme not only is even more natural (to move in a direction simply push the lever in that direction), but it adds a new degree of movement, and so new possible actions in the virtual environment.
  Unlike the keyboard, where the character could only move in 8 directions (4 of them possible only with a combination of 2 different buttons), with this stick we have a full (simulated) 360 degree of directions available, and even a speed control (how much the lever is pressed in a direction).
  This increase control, possibilities, is more natural and so it offers more feel of presence.

- **Full body tracking:**
  this is obviously the most natural control mapping, since it requires no precedent input-training on the user side.
  Independently of how (and with what devices) it has been achieved, full body tracking allow full control (direction and speed), it adds new movement possibilities which before should have been mapped to new buttons (like crouching or jumping) and eliminate completely the "translation" needed by the user to perform an action: if he wants to do something in the virtual world(in this case movement-related), all he has to do is perform the same action in the real world, and this greatly increase the presence factor.

With this simple example it seems easy predictable that the more the control scheme is "natural" and the more the user can experience the virtual environment without feeling the layer of the control devices, increasing immersion and thus presence.

5.2 Typology of Natural Mapping

In Skalski, Tamborini, Shelton, Buncher and Lindmark[9] work, the authors lay out a typology of natural mapping.
This includes:

- **Directional natural mapping:**
  this is the most prevalent form of mapping seen today.
  The player presses the left direction on the game pad stick, and their controlled avatar moves left.

- **Kinesic natural mapping:**
  this is found in the *Microsoft Kinect* and has users performing the actions of their characters but without a tangible controller, since their body gets tracked.
  This could be like a "full natural mapping", but only for some actions (like movement) and there are a lot of problems for its use: the user is not stationary (if you walk in an open virtual environment, where do you go in real life?), the user body is not always full tracked (Kinect has a single point of view, so when the user turns around his arms can't get tracked), and so on.

- **Incomplete tangible natural mapping:**
  this works by giving players a controller that partially emulates the feel of what they're doing in a game.
  For example, the *Wii remote* in a golfing game.

- **Realistic tangible natural mapping:**
  this makes use of a tangible, realistic analogue to what the player is using in whatever virtual reality they are engaged in.
  An arcade game where players use a gun-shaped controller to aim and fire at onscreen enemies is an example of realistic tangible mapping.
  The use of this type of natural mapping has thus far been relegated to arcade games or, very rarely, a few home console games.
  However, due to the surge in interest in VR due to the new technology now in development, such as the Oculus Rift, and advances in motion tracking technology, it is possible to create controllers which fall into the category of realistic tangible natural mapping.

Their study explored the influence these different types of controls had on both player presence and player enjoyment.
The study consisted of two experiments, and both showed that using the more natural control type yielded higher levels of presence than the other control types.
Initial results showed no significant relationship between presence and enjoyment.

Path analysis results on the other hand showed that enjoyment was predicted by presence.

Shafer, Carbonara and Popova[13] also looked in to the topic of control types and their influence on presence.
The authors investigate control types in terms of interactivity.
They list the PlayStation Move and Microsoft Kinect as high interactivity, the Wii remote as medium interactivity and standard gamepad controls as low interactivity.
Their study consisted also of two experiments, and their findings were that higher controller interactivity lead to higher levels of spatial presence.
They also found that higher levels of spatial presence predict higher levels of enjoyment.

The literature concerning controller types as they relate to spatial presence and enjoyment is fairly clear.
Control types which are more naturally mapped or more interactive lead to players experiencing higher levels of spatial presence as well as enjoyment.

Therefore, summarizing about the input devices usage in virtual reality, we can predict that:

- A motion-control device will be judged as more natural than a traditional mouse and keyboard.
- A control method which is judged as more natural will cause higher levels of presence when paired with a virtual reality head mounted display like the Oculus Rift than with a standard display type, because this will immediately give the user higher levels of spatial presence.

5.3 Inputs through HMD

Keeping in mind the insights discovered in this section, we will later on implement a "test area" (see ections 8 and 9) where the user can perform different actions with different input devices, some with more "natural" controls than others, to see what are the best ways to perform actions maintaining an high degree of presence on the user, and if these action have flaws if experienced through the Oculus Rift.

6 Interfaces in Virtual Reality

6.1 The need of Interfaces

Speaking about videogames, the amount of abstraction gamers are willing (and used) to tolerate is noteworthy in many cases, as we've seen with the "suspension fo disbelief" concept in section 4.4.

Even for titles that claim to mimic or "simulate" reality with a great deal of accuracy, a large number of non-real artefacts remain.
Many of these have to do with interface and the heavy reliance on visual cues to replace the sense-data a person would absorb by being physically present in the gameworld.
The interface exists as a "film" between the player and the avatar controlled in the virtual reality.
But where does the "interface" takes place in an abstract view of the situation?

We can choose to see all of this as "sets".

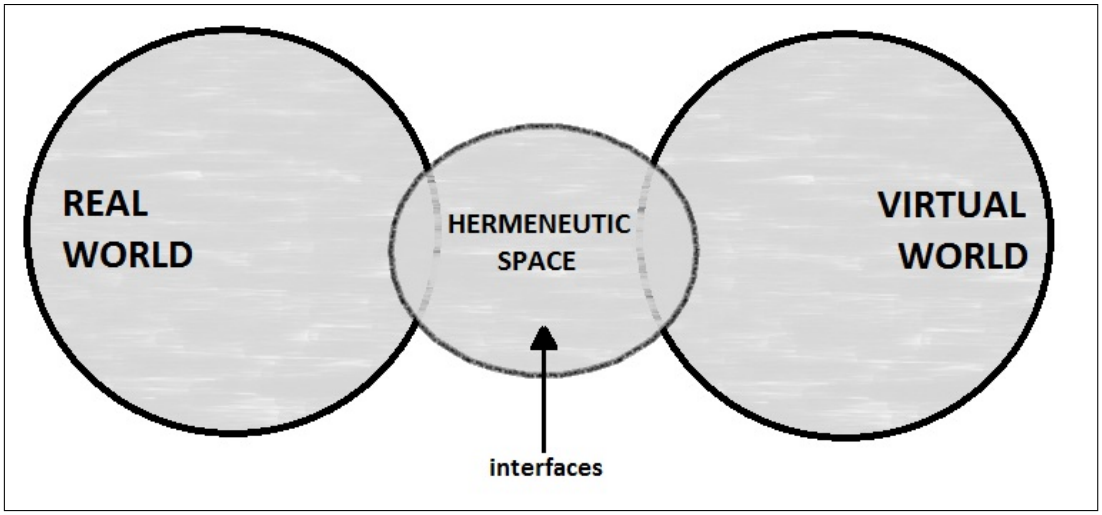

Figure 7:
The hermeneutic space

We have the real world, where the user has real physical presence, and the virtual world, where the user wants to enter, interact and "feel".
In this scenario is obviously needed some sort of layer capable of allowing the user in the real world to comprehend what is happening in the virtual world.
Since these two world are and always will be different from each other, this layer can be

seen as an "hermeneutic space" where the user is able to understand part of the sensory information about what is happening in the virtual world, so this layers is an intersection between the two worlds because of this.
The concept of interfaces extends to multiple subjects, as controls (inputs), understanding (graphical user interfaces) and, more generally, the wide subject of Human-Computer Interaction.

6.2 The User Interface

The interface layer is made up of the heads-up display that usually adorns the edges of videogame screens, overlain on the graphical world, and the input device.
The interface, of course, makes these games playable, and we've already seen that gamers are often willing to ignore the inconsistency (at the cost of some degree of presence) of these tropes in order to participate in the game.

Expert gamers display great agility in distinguishing between the supposed gameworld and the interface, though they both appear on the screen simultaneously.
That is not a native skill: it is common that not experienced players of a first-person shooter position themselves poorly by misinterpreting the HUD elements as all relating to the center of the character, rather than representing the right-hand.
Though it is easy, today, to program an avatar-based game with no additional information on the screen than the physical gameworld as would be seen by the player/character, this is usually not enough information to play the game effectively.

The videogame avatar is a simulated person in a simulated world, but the player does not (with today's technology) have direct access to their sensorium.
The videogame has to simulate the collected awareness that a game character would have, primarily about the avatar's body, and the general "gamestate".
Relaying information from the gameworld to the player is the first job of a videogame's interface.

Since the possible gamestates are virtually unlimited in an avatar-based game world, heads-up displays often flag contextual situations where a particular action button can be used to perform myriad actions, from opening doors, pressing buttons to lighting fires or untying a captive NPC.
These flags will alert the player to the possibility of interacting with the gameworld, which invokes the interface's second function: converting button-presses or other input methods into gameworld actions.

One of the abstract artefacts between player and the game world is often the avatar's ammunition, weapon status, health and other statistics.

This is a slowly fading artefact, in that ammunition is shown on the weapon itself in Halo and Dead Space, for example.
That's because the UI-design has evolved greatly in the last years.

6.3 Types of UI

User interface design in games differs from other UI design because it involves an additional element: fiction.
The fiction involves an avatar of the actual user, or player. The player becomes an invisible, but key element to the story, much like a narrator in a novel or film.
This fiction can be directly linked to the UI, partly linked, or not at all. Historically games didn't have any real link to the game's narrative, most likely because early games rarely had strong story elements.

Erik Fagerholt and Magnus Lorentzon introduce[7] terms for different types of interfaces depending on how linked to the narrative and game geometry.
They are:

- **Non-Diegetic**
- **Meta**
- **Spatial**
- **Diegetic**

Non-diegetic UI elements are the most traditional and have been used since the first videogames.
These elements have the freedom to be completely removed from the game's fiction and geometry and can adopt their own visual treatment, though often influenced by the game's art direction (so they still inherit the visual style associated with the game world).
These elements are best used when the diegetic, meta and spatial forms provide restrictions that break the seamlessness, consistency or legibly of the UI element.
To make an example *World of Warcraft* uses a mostly Non-diegetic UI (except for the Spatial-UI player names).
It allows the user to completely customise it, hopefully ensuring a familiar experience.
Most of the UI elements in World of Warcraft sit on the 2D hub plane, some elements sit within the world's geometry such as the player names however the character isn't aware of any of the UI.

Figure 8:
Non-Diegetic UI: World Of Warcraft. Note the non-diegetic elements on the screen (the skill/actions selectors), while the Player's names are spatial-UI

Meta UI elements: sometimes UI elements, for one or more reasons, don't fit within the geometry of the game world.
They can still maintain the game's narrative but sits anyway on the 2D hub plane; these are called Meta elements.
A common example of a Meta UI element is blood the splatters on the screen as a form of health bar, as in *Call of Duty: World at War*.
Blood splashing on the screen within the 2D HUD plane to tell the player that the character is losing health.
Interacting with the phone in Grand Theft Auto 4 is an interesting example.
It mimics the real world interaction: you hear the phone ringing and there is a delay before the character and player answer it.
The actual UI element itself appears on the 2D hub plane though, so it's actually a Meta element, though the start of the interaction is Diegetic.

The character is answering the phone but the actual UI element is placed within the 2D HUD plane that only the player sees.

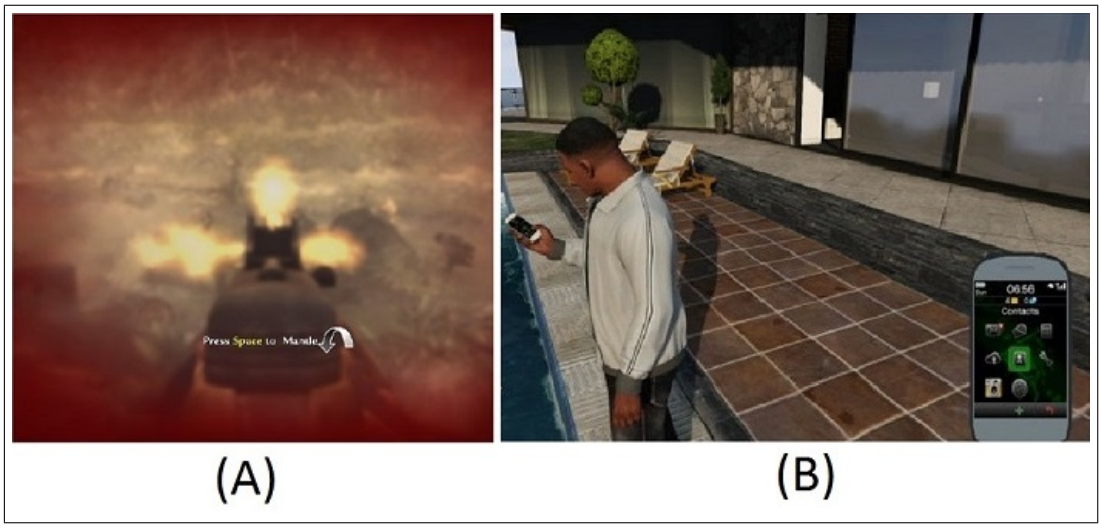

Figure 9:
Meta UI
- (A) Call of Duty World at War
- (B) Grand Theft Auto 5
As meta-UI we can see the blood effect occluding the visual to simulate being shot in (A) and an interactive 2D phone popping out when our avatar operate his in (B)

Spatial UI elements are used when there's a need to break the narrative in order to provide more information to the player than the character should be aware of.
They still sit within the geometry of the game's environment to help immerse the player and prevent them from having to break the experience by jumping to menu screens.
The closer these follow the rules of the game's fiction the more they can help immerse the player.
A good example can be seen in the game *Splinter Cell Conviction*: it adopts Spatial elements in the form of projections that illustrate objectives within the game world.
Type is overlaid in to the environment to communicate messages to the player rather than the character.
Spatial elements can be beautiful pieces when they work with the geometry of the world.
An example of this came from *Forza 4*: it demonstrate that a simple style can contrast the rich 3D qualities of the game.
Bold iconography combined with strong typographic layouts help establish a beautiful art direction for *Forza 4*'s UI.

Figure 10:
Spatial UI
- (A) Splinter Cell Conviction
- (B) Forza 4.
As spatial UI we can see the game informing the player of a possible action by writing the info on a world 3D object in (A) and info drawed around a 3D object in (B)

Diegetic UI elements exist within the game world (fiction and geometry) so the player and avatar can interact with them through visual, audible or haptic means.
Well executed diegetic UI elements enhance the narrative experience for the player, providing a more immersive and integrated experience.
For example, *Far Cry 2* uses a lot of Diegetic UI with only a few HUD elements to help to support the game's narrative.
It runs the risk of frustrating the player though slow response times but this forms part of the game mechanic: when the player wants to look at the map, he has to wait for the avatar's animation of pulling it out.
Many games get away with using Diegetic patterns because their narrative is set in the future, where UI overlays in some sort of hologram in daily life are commonly accepted. *Assassin's Creed* manages to use a lot diegetic patterns even though it's set in a historical world because the player of the player is using a virtual reality system in the future. So the story is in fact futuristic rather than historical.
If the story was set in a different time period the UI elements would be probably be considered Spatial instead of Diegetic. The game uses it's "eagle vision" to highlight enemies and their patrol track.
The player and the character see the same thing. There are cases when diegetic UI elements aren't appropriate, either because they aren't legible in the geometry of the game world, or there's a need to break the fiction in order to provide the player with more information than the character should or does know.

Figure 11:
Diegetic UI
- (A) Far Cry 2
- (B) Assassins Creed.
As Diegetic UI we can see the map and the gps (they're actually 3D objects, manipulated by the avatar and coherent with the environment reality) in (A) and the eagle vision overlay of Assassins Creed (it's 3D particle effect, coherent with game environment since it's supposed to be viewed through a computer) in (B)

6.4 UI through HMD

How will these kind of UI perform inside a virtual reality environment experienced through a head mounted display?
Physical interaction methods and immersive technology such as the Oculus Rift promise to challenge game UI design, allowing for a stronger connection between the avatar and character as both engage in similar actions at the same time.
Game UI has a key advantage (or disadvantage from some perspectives) in that players are often engaged with the narrative and/or game mechanic enough for them to learn new interaction patterns, or forgive bad ones. This is likely the reason so many games have bad UI, as testing needs to encompass the core game mechanic while UI is seen as secondary.
This can no longer be allowed in the case of virtual reality headsets, and one of the goals of this study will be to examine all these different kind of UI and determine what will work, what won't, where are the weak points and where are the advantages in using different kind of UI designing a good interface for the user to be experienced through an HMD.

7 Devices and Software Used

Since the introduction of computers to our world, we have been witnessing creative inventions in the science of human-computer interaction.
For a long time, the term "input device" evoked mainly two specific devices: the keyboard and the mouse, the main instruments used to provide user input to a personal computer.
Nowadays thanks to an increasing research on human-computer interaction technologies, we have a large set of input devices that changed the way of interaction.
The new approaches of human-computer interfaces will facilitate a more natural, intuitive communication between people and all kinds of sensor-based input devices, thus more closely mimicking the human-human communication.
These innovative technologies empower users to be more natural and spontaneous when dealing with them, allowing increased efficiency, speed, power, and realism factor.
However, many users feel comfortable with traditional interaction methods like mice and keyboards to the extent that they are often unwilling to embrace new, alternative interfaces.
A possible reason for that might be the complexity of these new technologies, where very often users find it disturbing to spend a lot of time learning and adapting to these new devices.
Gesture-based human-computer interaction could represent a potential solution for this problem since they are the most primary and expressive form of human communication thus they're able to increase the "presence" factor if used inside virtual realities.

All tests in this study were performed using a specific set of software and hardware input/output devices which aims at increasing this "presence" factor by mimicking human natural ways of interaction.
In this section we're going to examine what they are and how do they work.

7.1 Oculus Rift DK2

The Oculus Rift DK2 is a virtual-reality headset that act as an input-output device.
It takes the input from its internal sensors and the external infrared camera, sends them to a computer for the scene elaboration and gets in return the output of the computer's video card to display the elaborated 3D images back at the user.

7.1.1 History of Oculus Rift

The Oculus Rift was invented by *Palmer Freeman Luckey*, born in 1992 on September 19 at Long Beach, California.

Around the age of 15 Palmer started to get interested in the area of virtual reality, collecting various early attempts at virtual reality headsets.
Around the age of 16, unhappy with the existing head mounted virtual reality set in his collection due to high latency, low field of view and high contrast, Palmer started building his first prototypes of an headset of his own. He developed a series of prototypes, starting by piercing together old devices and eventually crafting something by himself, until he reached his 6th generation unit called "rift", which was intended to be sold as a do-it-yourself kit on the Kickstarter crowdfunding website with a target of about 100 customers.

Palmer regularly posted updates of his work on *MTBS3D*, a forum website frequented by virtual reality enthusiasts, where he was contacted by *John Carmack*, co-founder of *id Software*, which was very interested in his project and requested one of the prototypes. As he told *Eurogamer*[5] on 2013:

> *"... He ended up seeing my head-mounted display work and asked me, 'Hey, what you have looks interesting – is there any chance I could buy one?' He's John Carmack," Luckey snorts, "I just gave him one instead – you can't turn him down."*

A few month later, Carmack was at the *Electronic Entertainment Expo (E3) 2012*, where he was demoing a modified version of id Software's *Doom 3 BFG Edition* running on Palmer's Rift prototype, slightly improved.
With the resulting attention of thousands of people suddenly drawn to the Rift, Palmer (just past halfway to his journalism degree) dropped out of college to start a company: **in June fo 2012, Palmer formed *OculusVR***.

When Palmer first started thinking about Kickstarting a virtual reality headset, long before he met Carkmack or even formed a company, he hoped he'd get about 100 backer for his project.
In April 2012 he wrote[6]:

> "*I won't make a penny of profit off this project, the goal is to pay for the costs of parts, manufacturing, shipping, and credit card/Kickstarter fees with about $10 left over for a celebratory pizza and beer.*"

[5]Article "*Happy Go Luckey: Meet the 20-year-old creator of Oculus Rift*" readable at: *http://www.eurogamer.net/articles/2013-07-11-happy-go-luckey-meet-the-20-year-old-creator-of-oculus-rift*

[6]Post readable on on MTBS3D forum at: *http://www.mtbs3d.com/phpBB/viewtopic.php?f=120&t=14777*

After the Carmack's show at E3 2012 and the foundation of OculusVR, however, other industry titans (like *Gabe Newell* of *Valve*) threw their support behind the project.
On August 1st of 2012, Oculus launched their Kickstarter campaign with the modest goal of $250,000, way less than some earlier 90's VR headset that have flopped in the market.
Within 24 hours, they'd raised $670,000 from 2,750 people.
Within three days, they'd broken a million dollars.
At the end of the campaign, they managed to raise over $2.4 million.

Figure 12:
Oculus Rift DK1

Thanks to the Kickstarter campaign, OculusVR managed to create the **Oculus Rift DK1**.
This product wasn't intended to final consumer, but rather for developers, to get people building things in VR. The DK1 gave most people their first glimpse at Oculus Potential, and it made one thing clear: this little $350 headset was already better than everything that came before it, but it wasn't perfect:

- The low resolution screen made even modern 3D rendered environments looks dated. It was like sitting too close to an old TV, or staring at the display through a screen door (the "screen door effect").
- The lacking of positional tracking. While the headset's sensors could keep tabs of

how your head was angled, it was impossible to determine where your head was in space (e.g. you could look down to and item but couldn't lean in for a closer view).

- The points above contributed to the sensation of motion sickness on players and their low feel of "presence".

Despite the flaws, Oculus made around 65,000 units of DK1 which got officially sold out on February 21st of 2014.

Figure 13:
Oculus Rift DK2

On March 19th of 2014 Oculus began accepting pre-orders for their second hardware release, the **Oculus Rift DK2**, which is coming close to the product Oculus intends to ship to final consumers (but still isn't).
With this improved version they have:

- Increased the resolution to 960 X 1080 (increasing the overall pixel count by over 100%), making the "screen door" effect much less noticeable.
- They changed the old LCD display with an OLED, which offers brighter screens with less motion blur (reducing the motion sickness effect on players).
- Decreased the latency of the movement of the headset from 60 ms to 30 ms.

- They introduced a new piece of hardware. an external InfraRed camera able to see where the headset is thanks to a series of LEDs (adding support for movements like leaning).

In late March 2014, Oculus VR was acquired by *Facebook* for $2 billion, and in September 2014, during the *Oculus Connect Conference* in Los Angeles they presented an updated version of the Rift DK2 code named **Crescent Bay**.

Figure 14:
"Crescent Bay" prototype

This version has a greater resolution than the DK2, a lower weight, built-in audio and 360-degree tracking (thanks to the presence of tracking LEDs in the back of the headset but it won't be available for sale, not even for developers, since the company is currently aiming at the delivery of a final product, **consumer oriented**, before 2016.

7.1.2 Dev Kit Specs Comparison

Since the current *Crescent Bay* prototype is not available for use, we will analyze and compare the specifications only for the previous developer kits:

	DK1	**DK2**
Price	$300	$350
Weight (headset)	380g	440g

Screen Resolution	640 x 800 (1280 x 800 split between each eye)	960 x 1080 (1920 x 1080 split between each eye)
Total Pixels (per eye)	512,000	1,036,800
Pixel Layout	RGB	Pentile
Screen Type	LCD	OLED
Screen Size	7”	5.7”
Screen Manufactorer / Model	Innolux HJ070IA-02D LCD	Samsung Galaxy Note 3
Latency	50ms – 60ms	20ms – 40ms
Low Persistence	No	Yes
Refresh Rate	60 Hz	75 Hz
Orientation Tracking	Yes	Yes
Positional Tracking	No	Yes
Gyroscope, Accelerometer, Magnetometer	Yes	Yes
Field of View	110°	100°
3D	Stereoscopic	Stereoscopic
Inputs	DVI, USB	HDMI 1.4b, USB, IR Camera Sync Jack

Table 1: Oculus Rift Developer Kits Comparison

7.1.3 Oculus DK2 Technology Overview

Since the Development Kit 2 is the one used for the tests, we will concentrate only on its features, ignoring the other prototypes.

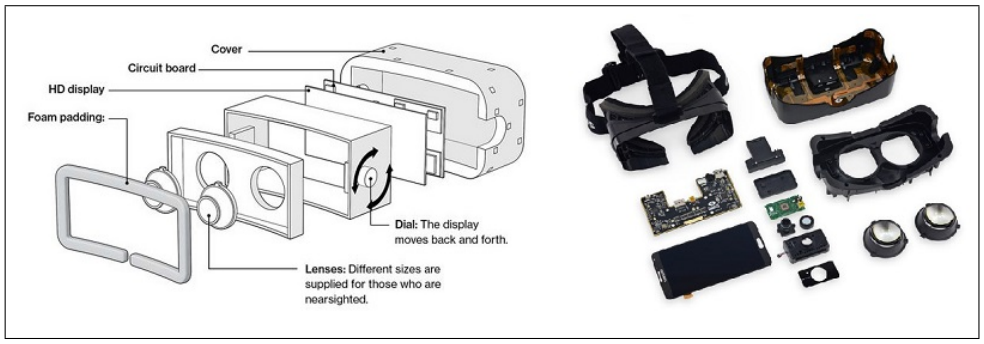

Figure 15:
Oculus Rift DK2 Internal Components

Lenses:
The Oculus DK2 comes with three different set of lenses, coded “A”, “B” or “C”.
Lens pair “A” is to be used by people who have excellent long sighted eyesight as the rift is focused at infinity.
The pairs B and C are to be used by people having problems with nearsightedness, though cannot be used by all, especially people with major vision complications.
Furthermore glasses can also be worn along with the Oculus Rift headset, provided that the glasses are not huge.

Head Tracking:
Head tracking lets the user look around the virtual world just in the manner they would in the real world.
The DK2 head tracker constantly analyzes the player’s head movement with a high rate and uses it to control the view, instead of relying on other inputs to turn your view in the virtual reality.
This makes for a completely natural way to observe the world, which is a major factor in immersion.

The headset consists of a Oculus Tracker v2 board consisting of chips controlling the head tracking device. These chips are:

- *STM microelectronics 32F103C8 ARM Cortex-M3* Micro-controller with 72MHz CPU.
- *Invensense MPU-6000* six-axis motion tracking controller.
- *Honeywell HMC5983* with three axis digital compasses used in conjunction with the accelerometer to correct for gyroscope drift.

The Oculus VR sensor support sampling rates up to 1000 Hz, which minimizes latency (the time between the player's head movement and the game engine receiving the sensor data) to roughly 2 milliseconds.
The increased sampling rates also reduce orientation error by providing a denser dataset to integrate over, making the player's real-world movements more in-sync with the game.
As previously said, Oculus VR sensor includes a gyroscope, accelerometer, and a magnetometer, and when the data from these devices is fused it helps in determining the orientation of the player's head in the real world and synchronize the player's virtual perspective in real-time.
The Rift's orientation is reported as a set of rotations in a righthanded coordinate system.

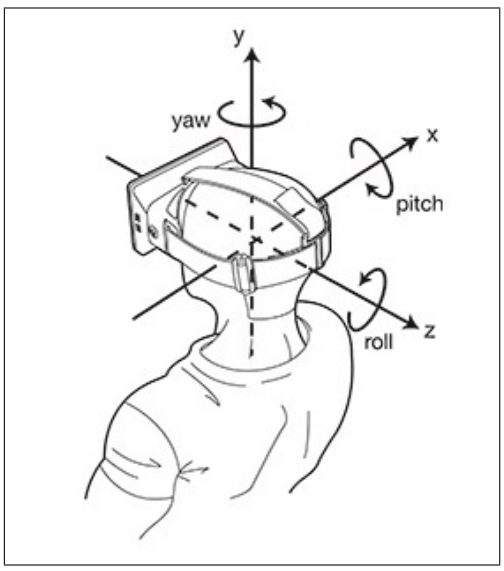

Figure 16:
Oculus Rift DK2 Head-angle Tracking

The gyroscope, which reports the rate of rotation (angular velocity) around X, Y and Z axes in radians/second, provides the most valuable data for head orientation tracking. By constantly accumulating angular velocity samples over time, the Oculus SDK(system development kit) can determine the direction of the Rift relative to where it began. Although the gyroscope provides orientation relative to the starting point, it creates two difficulties: it cannot provide the original orientation of the headset and it's subject to a small amount of drift over time (imagine reorienting your head back to perfect center but in-vr you're now looking slightly left or right).
These issues affect the VRs with a fixed reference point (e.g. a game where the player is fixed inside a cockpit, where the players head's orientation does not affect the position of vehicle being piloted).

The accelerometer is leveraged to estimate the "down" vector and the magnetometer measures the strength and direction of the magnetic field.
Combined, these allow for correction of drift in all three axes.

Positional Tracking:
The Oculus DK2 introduced the support for position-tracking of the headset in space, which allows the user, for example, to lean in for a closer look at an in-game object or to peek around a wall by only moving his head and upper body, exactly like in real world. This is possible thanks to a new hardware device, an InfraRed Camera, which can track the headset position through to a series of LEDs hidden below the Rift frontal cover.

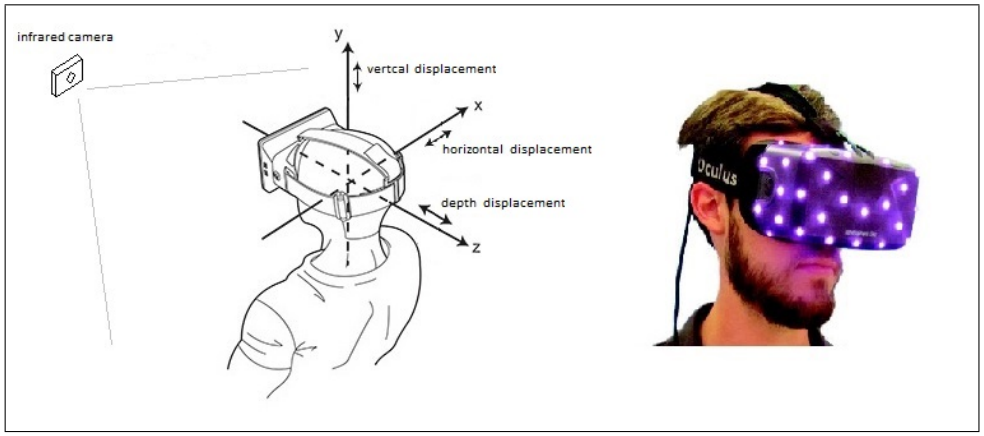

Figure 17:
Oculus Rift DK2 Head-movement Tracking through infrared camera+LEDs

This added positional tracking helps with reducing the dizziness since the brain doesn't get confused by the missing degree of motion, hence greatly increasing the immersion

factor of the user inside the virtual reality.

7.1.4 Known Oculus DK2 Problems

The Oculus Rift DK2 is an amazing piece of hardware, allowing to experience virtual reality and presence factor like never before, but it's clear that it's still in its beta-phase, since it still shows a lot of youth-related problems.

Wires:
While wearing the Oculus Rift the user obviously can't see the real world.
That means that by exploring the virtual environment he's experiencing, he will lose orientation in the real world, risking to wallow or stumble in the wires.
This is a temporary problem though, since the final, consumer version of the Oculus Rift is supposed to be wireless to allow total freedom of movement.

Screen Door Effect:
The "Screen Door Effect" can be described as a black grid over the original image.
This occurs because of empty spaces in between the pixels. Every display has a characteristic known as the *Pixel Fill Factor* that is responsible for this effect.
On any LCD display every pixel is made of three sub pixels, namely red green and blue (RGB). Human perception of different colors on the display is a result of the varying intensities of these sub pixels.
The distance between these sub pixels is called as the pixel pitch which ultimately decides the pixel fill factor.
Higher the pixel pitch higher is the pixel fill factor. The Oculus Rift DK2 (unlike the DK1) has a fair pixel fill factor, however since it is worn close to the eyes (only few centimeters) it still gives rise to the screen door effect.

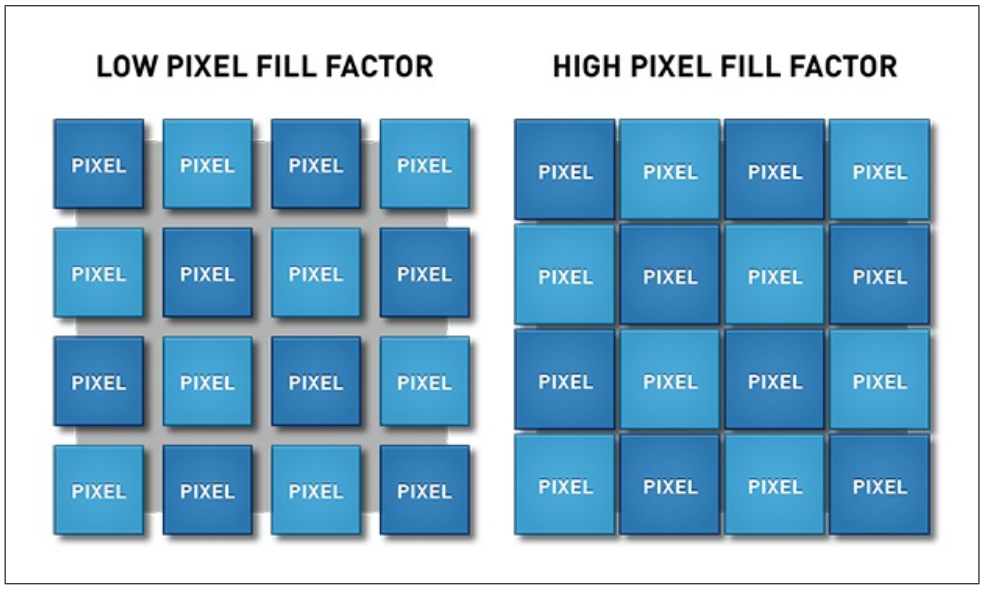

Figure 18:
Display pixel fill factor

Ghost Effect:
Ghosting is the appearance of faded trails behind any moving object.
In Oculus Rift, the slow pixel switching time causes ghosting, meaning the pixels take a fair amount of time to change intensities as compared to the motion of the head.
Faster the movements greater the ghosting, since the pixel switching lags behind. Thus ghosting persists until the head stops moving. This causes blurring of the scene and, consequentially, decreased immersion degree for the user inside the virtual reality and an elevated sense of nausea and motion sickness.
Ghosting can be avoided with a higher switching rate for the pixels, and despite the improvements from Oculus Rift DK1 to DK2, this effect is still present in the DK2 prototype.

Simulation Sickness
This is probably the biggest problem the Oculus Rift has yet to overcome.
The solution is complex since it depends on multiple factors:

- Personal sensitivity
- Virtual Reality method of interaction

- Hardware Technological level

The personal sensitivity is different from person to person (e.g. some people can feel a sense of nausea after a boat ride, others don't), so we can only slightly lower this sensitivity by working on other factors, but we can't overcome it completely.
The different possible ways of interaction inside the virtual world can make a big impact on that matter: studies have shown[18] that different controls can lead to different degree of simulation sickness, even forcing the users to take off the headset because of this.
The hardware technological level is probably the easiest part where things are gonna be improved: the final, consumer grade version of the Oculus Rift will be improved under every aspect compared to the DK2, so when the new hardware will be available we'll be able to determine how much the technological level really impacts the aspect of simulation sickness.

7.2 Leap Motion

The Leap Motion controller is a 3D optical input device that tracks the hand of the user in space and passes that data to his software on a computer.
Another technology with similar goal is the Microsoft's Kinect: this controller was initially developed to allow the user to interact with the Xbox console without any controllers, however it was used further as a vision platform in many different applications.
The Leap Motion controller was chosen against the kinect because, with the goal of integrating an input technology in a virtual reality environment, the Leap Motion is a better since it allows a gesture and position tracking with sub-millimeter accuracy[15], while the Kinect controller showed that it has an approximately 1.5 cm standard deviation in depth accuracy and the full-body movement tracking (available with Kinect and not with the Leap Motion) isn't a necessity as only hands tracking is required.

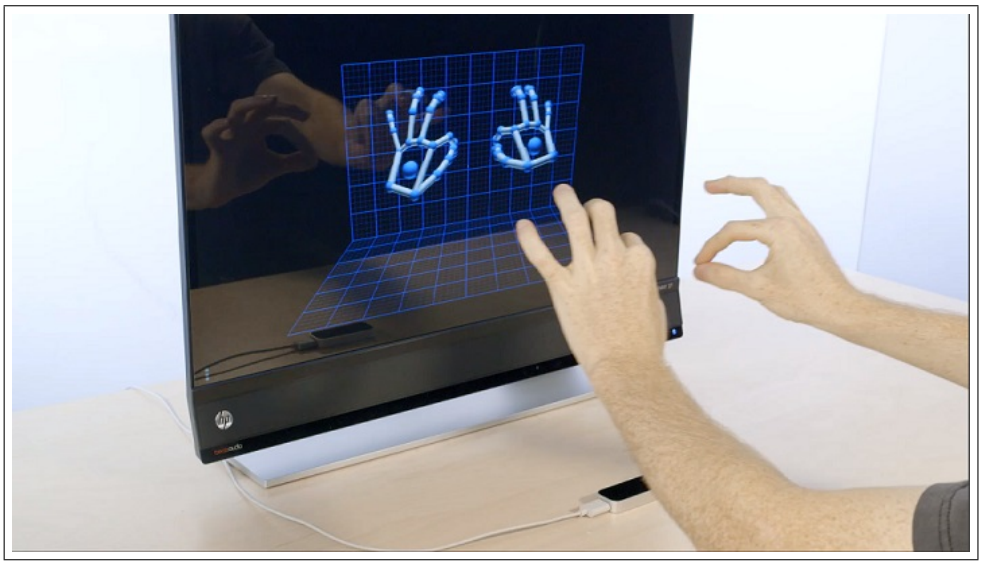

Figure 19:
Leap Motion in action while connected to a personal computer

7.2.1 Leap Motion Technology Overview

In the last few years, different optical sensors have been developed, which allow the mapping and acquisition of 3D information.
Various applications also have been introduced, which exploit the increasing accuracy and robustness, and the decreasing cost over time of 3D sensors.
The applications range from industrial use, object tracking, motion detection and analysis, to 3D scene reconstruction and gesture-based human machine interfaces.
These applications have different requirements in terms of resolution, frame-rate throughput, and operating distance.
Especially for gesture-based user interfaces, the accuracy of the sensor is greatly considered a challenging task.
The Leap Motion Controller is considered a breakthrough device in the field of hand gesture controlled human-computer interface.
The new, consumer-grade controller introduces a new novel gesture and position tracking system with sub-millimeter accuracy.
The controller operation is based on infrared optics and cameras instead of depth sensors, and at the moment its motion sensing precision is unmatched by any depth camera currently available.
It can track all 10 of the human fingers simultaneously, with an accuracy in the detection

of each fingertip position of approximately 0.01mm (as stated by the manufacturer), with a frame rate of up to 300 fps.

The controller is an optical tracking system based on stereo vision. Within its surface area of 24 cm^2, the controller has three infrared light emitters and two IR cameras. The field of view of the controller is very wide, up to 150°, which gives the user the opportunity to move his hand in 3D, just like in real world.

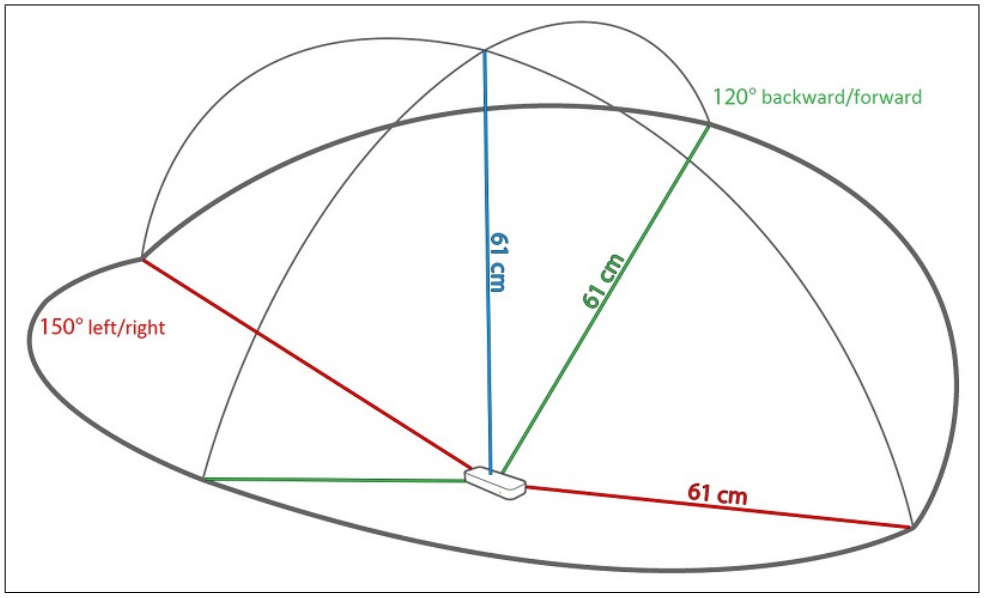

Figure 20:
Leap Motion tracking zone

The Software Development Kit (SDK) supplied by the manufacturer delivers information about Cartesian space of predefined objects such as the finger tips, pen tip, hand palm and position.
Also, information about the rotations of the hand (e.g. Roll, Pitch, and Yaw) are available as well.
All delivered positions are relative to the Leap Motion controller's center point, which lies between the two IR cameras, just above the second IR emitter.

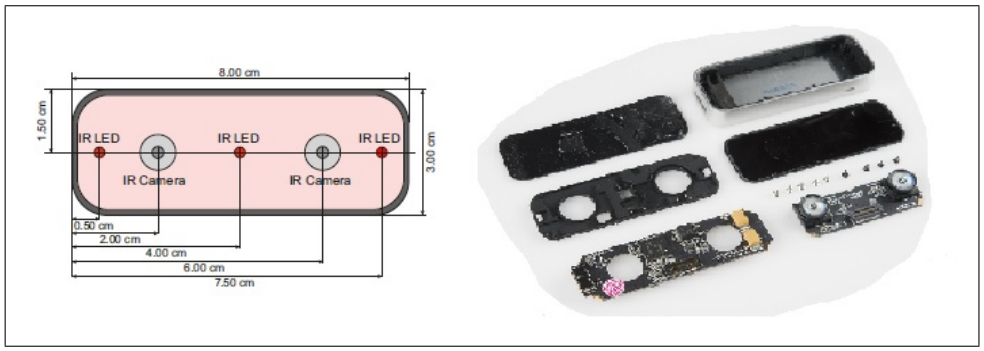

Figure 21:
Leap Motion internal components

7.2.2 Known Leap Motion Problems

The Leap Motion controller really have a sub-millimeter accuracy (which is more than enough for a convincing hand movement and gesture tracking), but it has some problem that will impact his performance in virtual-reality use (as we'll see in section 9).

Oculus Rift mount:
When the Leap came out, it was supposed to be used by being positioned on a flat, fixed surface (e.g. below the computer keyboard) and by tracking the user's hands from below, looking mostly at the user's palms (see figure 19).

For our tests it has been mounted on the front side of the Oculus Rift, so when the user will be using his hands in the virtual reality, the Leap controller will view the back of the hands for most of the time, as seen in figure 22.

Figure 22:
Leap Motion fixed on the Oculus Rift front side

How this will impact on its tracking capability will be tested in section 9.

Angle, range and quality of hand tracking:
Another problem with the usage of the Leap controller in the virtual reality will be its range and angle of hand tracking: this aspect will be important since it will not only affect the presence factor on the user, but the overall gameplay and available interactions inside the virtual reality must be designed by keeping these limits into account.

7.3 Razer Hydra

The Razer Hydra can be seen as the first real motion controller developed exclusively for PC devices, and offers two 3D magnetic trackers with declared resolution of 1mm and 1 deg when closer than 0.5m from the base station.
This tracker works extremely well in good conditions. The downside is that the tracking becomes unstable when too far from the base, limiting the available workspace.
It was originally designed to be used in a typical seated environment in front of a PC, and does not require a line of sight to operate.

Figure 23:
The Razer Hydra Controller. We can see the two motion controller and the base station

The Razer Hydra base station, to detect the controllers spatial position, uses a magnetic field which is 20 times weaker than the Earth's natural magnetic field, allowing it to operate without adversely affecting objects in its surrounding, such as credit cards, hard disk drives and speakers.

Technical specification per controller:

- Thumb-ergonomic analog stick for fluid control
- Four Hyperesponse action buttons
- Rapid-fire trigger and bumper for faster in-game response
- Non-slip satin grip surface
- True six degree-of-freedom magnetic motion tracking
- Lightweight, anti-tangle braided cable

Technical specification of the base station:

- Low-power magnetic field, low power consumption
- Ultra precise sensor for 1mm and 1 degree tracking
- No line of sight to controllers required
- Low latency feedback

7.3.1 Known Razer Hydra Problems

The Razer Hydra wasn't developed with Oculus Rift usage in mind, so a few problem are to be expected.

Wires:
As for the Oculus itself, wires will surely be a problem.
Since the user can't see the real world, he risks of wallowing himself with the wires or to reach their maximum length without noticing.

Tracking quality: The wires are 2 meters long, but the optimal tracking distance stated is of only 0.5 meters, which means that the further we go from the base the worst will be the tracking quality.
This degradation will be tested in section 9.
Another aspect about quality is the interference: since the hydra works on the principle of magnetic tracking it must be used in an environment free of possible interference, such as other objects that emit their own magnetic fields.

7.4 Software Used

To use the input/output devices, to create the 3D virtual environment and to code all the interactions, different software were used.

7.4.1 The game engine: Unity3D

Unity3D is a cross-platform game creation system developed by *Unity Technologies*, including a game engine and integrated development environment (IDE), which supports C#, UnityScript (a Unity's version of Javascript) and Boo languages.
First announced only for Mac OS, at *Apple's Worldwide Developers Conference* in 2005, it has since been extended to target more than fifteen platforms, such as *BlackBerry 10*, *Windows Phone 8*, *Windows*, *OS X*, *Linux* (mainly *Ubuntu*), *Android*, *iOS*, *Unity Web Player*, *Adobe Flash*, *PlayStation 3*, *PlayStation 4*, *PlayStation Vita*, *Xbox 360*, *Xbox One*, Nintendo *Wii U*, and Nintendo *Wii*.

For the making of this project, it was used the version *4.6.1f1 PRO*, with no extra plugins and all the code was written in C# language.
Both the free version and the Pro version can be downloadable at the url *http://unity3d.com/*.

7.4.2 The Oculus Rift software

To correctly work on our system, the Oculus Rift DK2 headset rely on various software, all downalodable at url *https://developer.oculus.com/downloads/*:

- **Oculus Rift Runtime for Windows**, version *0.4.4-beta*:
  this allows our system to recognize and use the hardware.
- **Oculus SDK for Windows**, version *0.4.4-beta*:
  this installs the software and the documentation needed to develop with the Oculus Rift.
- **Unity 4 Integration**, version *0.4.4-beta*:
  this is the integration for the game engine Unity3D, which gives us the gameobjects and the scripts needed ready for use inside the Unity editor.

7.4.3 The Leap Motion software

To correctly work on our system, the Leap Motion controller rely on various software:

- **Leap Motion Runtime for Windows**, version *2.2.3*, downloadable at url *https://www.leapmotion.com/*:
  this allows our system to recognize and use the hardware.
- **Leap Motion SDK for Windows**, version *2.2.3.25970*, downloadable at url *https://developer.leapmotion.com/*:
  this installs the software and the documentation needed to develop with the Leap Motion.
- **Unity Pro Asset**, version *2.2.0.23475*, downloadable at url *https://developer.leapmotion.com/downloads/unity/*:
  this is the integration for the game engine Unity3D, which gives us the gameobjects and the scripts needed ready for use inside the Unity editor.

7.4.4 The Razer Hydra software

To correctly work on our system, the Razer Hydra controller rely on various software:

- **Sixense SDK for Windows**, version *062612*, downloadable at url *http://sixense.com/windowssdkdownload/*:
  this installs the software and the documentation needed to develop with the Razer Hydra.

- **Sixense Unity Plugin**, version *1.0.2*, downloadable at url *https://www.assetstore.unity3d.com/en/#!/content/7953/*: this is the integration for the game engine Unity3D, which gives us the gameobjects and the scripts needed ready for use inside the Unity editor.

8 Creation of the test room

Now we can finally try to experience all inputs method and graphical user interfaces interactions discussed in previous sections.
To do so, we must create a virtual environment, our "test room", able to support all of our hardware devices and use them to test different way of interacting with the environment's objects.

The aim of these tests will be to answer the following questions:

- can these hardware devices be easily integrated in a virtual environment with Unity3D?
- can their performance and usability be considered acceptable?
- which kind of graphical user interface works better and which is to be considered wrong and why?
- what input device is better at performing various task and why?
- is there a relation of the points above regarding presence factor?

8.1 Integrating Oculus Rift capability in Unity

First of all, we want to integrate the Oculus Rift inside Unity3D, since all we're going to do will be experienced through this headset.

After the installation of the *Oculus Runtime* and the *Oculus SDK* on our system, we can import the *Oculus Unity 4 Integration*.
After extracting the file *ovr_unity_0.4.4_lib.zip* we have, among other things, a file called "OculusUnityIntegration.unitypackage".
We can import this file in Unity3D, and we'll have the object "OVRPlayerController" ready for use inside our virtual scene.

This object already have a basic keyboard and gamepad input controller scripts able to move the player around and 2 already calibrated cameras, which allows stereoscopic vision through the headset.
While this technology can be considered still in its "prototype phase", the software already works great, a lot of documentation is already available and its integration in the game engine has been proven to be surprisingly easy.

8.2 Integrating Leap Motion controls

We will now integrate Leap Motion hand tracking inside our Unity project, which will allow us to have "virtual hands" in our virtual environments, tracked by our real hands movements.

First we have to install the *Leap Motion SDK* in our system, then we can import the Leap's *Unity Pro VR Asset* inside Unity.
This will give us the object "LeapController", and we will insert it in our scene as a child of the "CenterEyeAnchor" object, which is a child of the Oculus's "OVRPlayer-Controller" object.

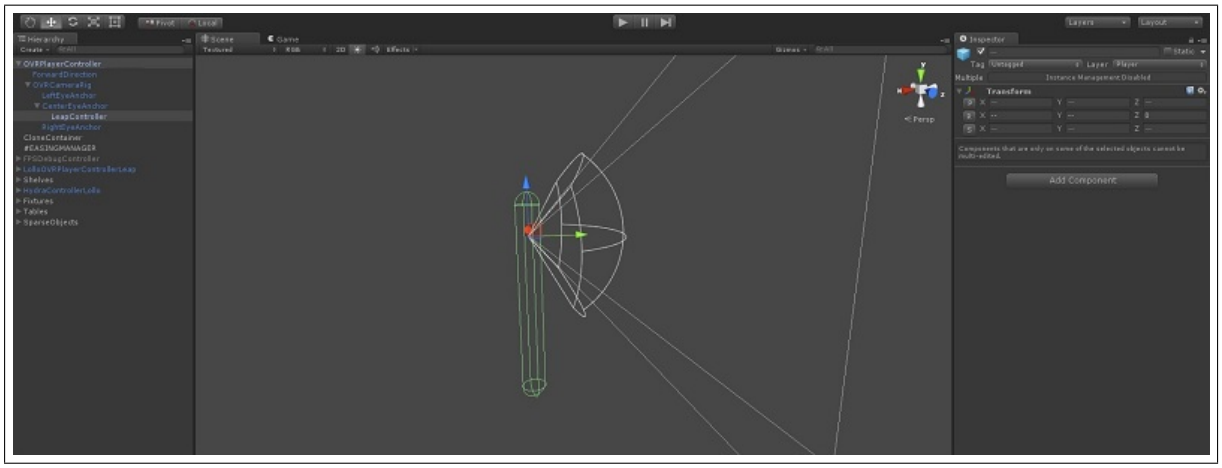

Figure 24:
Unity Editor: Leap Controller integration

As we can see in figure 24, we must move and rotate it to reflect its real position on our headset in the real world.
The white zone viewable in the editor represent its tracking zone, where our hands as 3D-objects will be created.

This controller is almost ready to use as it is: when we wave our hands in front of the tracking zone it starts tracking our hands, creating 2 new clone objects inside our scene (placed in front of the *LeapController* object) which are going to mimic our real hands movement.

As we can see in figure 25, these two objects are "GlowRobotFullRightHand", which

is the 3D model of the hand that we have chosen, and "RigidHand".
These comes already filled with *rigidbodies and colliders*, which means that our virtual hand can already physically interact with other objects inside our scene.

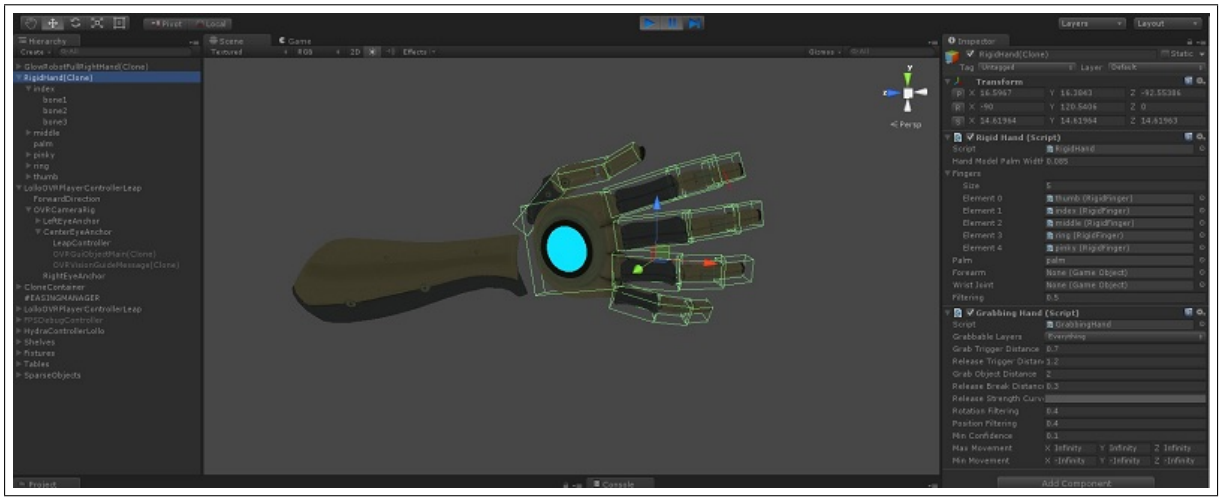

Figure 25:
Unity Editor: Leap Motion 3D hand

When we hide our hands from the Leap's sensor, the *LeapController* object destroys the virtual hands from the scene.
Since it's obviously impossible to move our avatar solely through our hands, we will rely on other devices (like Hydra / Gamepad) to execute movements.

This technology too have proven to be really easy to use in development, allowing us to add its support to our Oculus Rift in-game controller without difficulty.

8.3 Integrating Razer Hydra controls

Now we're going to integrate Razer Hydra support inside our Unity project, which will allow us to have another pair of virtual hands tracked by the Hydra's controllers.

First we need to install the *Sixense SDK* in our system, then we can import the *Sixense Unity Plugin* inside Unity.
This time we have to work a little harder to make the virtual hands work properly.

By using the new imported objects, we have to create a parent-child structure: we

create an empty object called “HydraController” as a parent, and put inside him as children the objects “HydraInput” and “HydraHands”.

Now we have to make our virtual hands appear in front of our avatar’s view since now they move only in front of our “HydraController” object, so we create a simple C# script called “FollowPosition.cs” and attach it to the “HydraController” object, putting the script’s target as the “CenterEyeAnchor” object seen before.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class FollowPosition : MonoBehaviour
5  {
6      public GameObject target;
7
8      void Update ()
9      {
10         // every render step, set this object position and rotation as the ones of
               the target object
11         transform.position = target.transform.position;
12         transform.rotation = target.transform.rotation;
13     }
14 }
```

Listing 1: FollowPosition.cs

Now we have our virtual hands correctly positioned, but they still don’t have any colliders, which means that they will simply pass through other scene’s objects.

To fix that we must add a “RigidBody” component to each hand, and in both of them add (for every part of the hand) some “Box Collider” components shaped in the right way, until we obtain what is shown in figure 26.

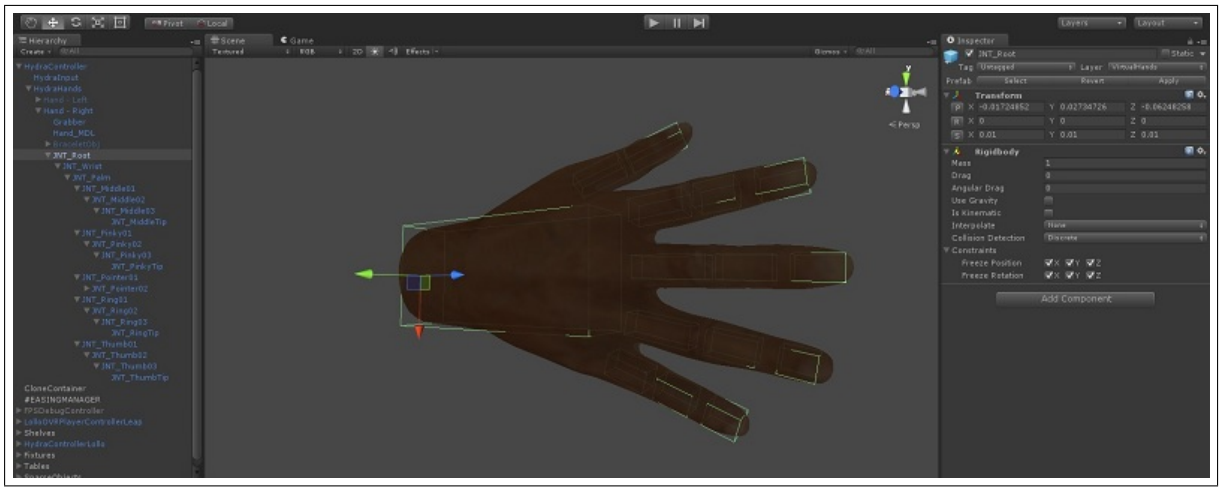

Figure 26:
Unity Editor: Hydra 3D hand

Finally, since with our hydra controllers we have two usable sticks, we want to be able to use them to move our avatar in the virtual environment.
To do that we create the script “HydraMovement.cs” and attach it to the “HydraController” object.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class HydraMovement : MonoBehaviour
5  {
6      public float movementSpeed = 0.1f;
7      public float rotationSpeed = 3f;
8
9      private SixenseInput.Controller leftController = null;
10     private SixenseInput.Controller rightController = null;
11     private float leftJoyX = 0f;
12     private float leftJoyY = 0f;
13     private float rightJoyX = 0f;
14     private float rightJoyY = 0f;
15     private Vector3 newPos;
16     private float yPos;
17
18     void Update ()
19     {
20         // get controllers once they're activated
```

```
21          if (leftController == null)
22              leftController = SixenseInput.GetController(SixenseHands.LEFT);
23          if (rightController == null)
24              rightController = SixenseInput.GetController(SixenseHands.RIGHT);
25
26          // if the left controller is set
27          if (leftController != null)
28          {
29              leftJoyY = leftController.JoystickY;
30              // if left stick was used forward/backward
31              if (leftJoyY != 0f)
32              {
33                  // move avatar forward/backward
34                  yPos = transform.position.y;
35                  transform.position += Camera.main.transform.forward * leftJoyY *
                        movementSpeed;
36                  transform.position = new Vector3 (transform.position.x, yPos,
                        transform.position.z);
37              }
38
39              leftJoyX = leftController.JoystickX;
40              // if left stick was used left/right
41              if (leftJoyX != 0f)
42              {
43                  // make avatar move sideway left/right
44                  yPos = transform.position.y;
45                  transform.position += Camera.main.transform.right * leftJoyX *
                        movementSpeed;
46                  transform.position = new Vector3 (transform.position.x, yPos,
                        transform.position.z);
47              }
48          }
49
50          // if the right controller is set
51          if (rightController != null)
52          {
53              rightJoyX = rightController.JoystickX;
54              // if right stick was used
55              if (rightJoyX != 0f)
56              {
57                  //rotate avatar
58                  transform.rotation *= Quaternion.Euler(0f, rotationSpeed*rightJoyX,
                        0f);
59              }
60          }
61      }
62  }
```

Listing 2:
HydraMovement.cs

As we've seen this technology has proven to need a little more work to reach the same usability level of the Leap Motion inside our game engine, but we can still state an overall simplicity of development even considering its different capabilities (e.g. we are now using its analogical sticks to move our avatar).

8.4 Creating the room

Now that we can see through the Oculus Rift what our avatar sees and control it in multiple ways (keyboard, gamepad, Leap Motion and Razer Hydra), we must place him inside a dedicated virtual environment.
To create this environment, 3D models, textures and sound were used.

The 3D models was partially made by me and partially taken (free of charge and royalty free) from the site *http://www.turbosquid.com/.*

The textures and materials came with the 3D models, and some were made by me.

The sounds was taken (free of charge under Creative Commons License) from the site *http://www.freesound.org/.*

8.4.1 Fixtures

Firs of all, the scene's geometry. Using textured 3D parallelepiped we create floor, ceiling and walls, adding in all of them *rigidbodies* and *box colliders* components to allow them to stop other objects and player from passing through and marking them as "static and kinematic", so that no matter the physical force applied on them, they won't move.

After that, with the same properties, we add shelves, sword-holder and lights on the walls, pedestals on the floor and a light with a rotating fan on the ceiling.
Obviously, in every light-point of the 3D models we have to add one or more child objects with a *point-light* or *spotlight* component to actually have lights.

8.4.2 Sparse Objects

After the fixtures we can add sparse object which the user can interact with.

So we add some lamps, tables and crates on the floor, some books, cans, swords, guns, tablets, monitors and abatjour on the tables, some more books and cans on the shelves and some vases on the pedestals.

To allow interaction every object needs to have its *colliders* and *rigidbody*, where we set the correct value of its supposed *mass* (e.g. a table has a mass of 60, while a book has a mass of 2).

Figure 27:
The Test Room, full of interactive objects

8.5 Preparing the interactive objects for use

Now all our objects inside the room react to physical touch, but nothing more.

This is alright for general objects, but some of them must be wielded in exact positions (think of swords or guns) and others must perform special actions (e.g. tablets must be operable, guns must be able to shoot bullets and so on).

Before doing that, though, our virtual hands must gain the ability to pick, hold, op-

erate and release/throw objects since at the moment they can only push other objects aside.

8.5.1 Adding virtual hands actions

Now we create a new, empty object called "Grabber" as a child of each hand.
We add to him a *RigidBody* component marked as "kinematic" (since we don't want to push objects with this one) and a *Sphere Collider* marked as "Trigger".
We're going to use this collider as a "picking up radius": when the user wants to grab something, we check what objects are inside this zone and we try to grab the closest one.
To do so, we must write a C# script called "HydraHandInteraction.cs" ant attach it to the "Grabber" object.
We can now see a simplified version of this script without the called functions:

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  public class HydraHandInteraction : MonoBehaviour
6  {
7      // HYDRA DATA
8      public GameObject     handColliderParent;
9      public SixenseHands   hydraHandType = SixenseHands.UNKNOWN;
10     public GameObject     grabPoint;
11     public float          maxGrabbableMass = 50f;
12     private SixenseInput.Controller controller = null;
13
14     // GENERAL DATA
15     Vector3               collisionPoint;
16     GameObject            grabHandle;
17     GameObject            grabbedObj;
18     Transform             grabbedOrParent;
19     bool                  grabbedOrKinematic;
20     int                   grabbedOrLayer;
21     List<GameObject>  objsInGrabRange = new List<GameObject>();
22
23     // hand movement calculation
24     Vector3    curHandPos = Vector3.zero;
25     Vector3    prevHandPos = Vector3.zero;
26     Vector3    prevPrevHandPos = Vector3.zero;
27     Vector3    curGrabbedObjPos = Vector3.zero;
28     Vector3    prevGrabbedObjPos = Vector3.zero;
29     Vector3    curHandRot = Vector3.zero;
30     Vector3    prevHandRot = Vector3.zero;
31     float      handVelocity, throwForce=2f;
```

```
32      Vector3 handRotOffset = new Vector3 (0f, -45f, 0f);
33      Vector3 handPosOffset = new Vector3 (0f, -2f, 0f);
34
35      // BUTTONS
36      private SixenseButtons grabButton = SixenseButtons.BUMPER;
37      private SixenseButtons shootButton = SixenseButtons.TRIGGER;
38      private SixenseButtons customButton1 = SixenseButtons.FOUR;
39      private InfoBracelet.BraceletType braceletType = InfoBracelet.BraceletType.None;
40
41      void Start ()
42      {
43          // look for bracelet on hand and set type
44          if (hydraHandType == SixenseHands.LEFT)
45              braceletType = InfoBracelet.BraceletType.Left;
46          else if (hydraHandType == SixenseHands.RIGHT)
47              braceletType = InfoBracelet.BraceletType.Right;
48      }
49
50      void Update ()
51      {
52          // get the controller reference
53          if (controller == null && hydraHandType != SixenseHands.UNKNOWN)
54              controller = SixenseInput.GetController(hydraHandType);
55
56          // if we're holding something
57          if (grabbedObj)
58          {
59              // update general data
60              UpdateHandData();
61              prevGrabbedObjPos = curGrabbedObjPos;
62              curGrabbedObjPos = grabbedObj.transform.position;
63
64              // send messages to holded obj
65              if(controller.GetButtonDown(shootButton))
66                  grabbedObj.SendMessage("Fire", SendMessageOptions.DontRequireReceiver);
67              if(controller.GetButtonDown(customButton1))
68                  grabbedObj.SendMessage("CustomAction1",
                        SendMessageOptions.DontRequireReceiver);
69          }
70      }
71
72      // when an obj enters our grabbing zone, we add it to our Grabbable list
73      void OnTriggerEnter(Collider c)
74      {
75          GameObject grabbable = isObjGrabbable(c.gameObject);
76          if(grabbable && !objsInGrabRange.Contains(grabbable))
77              objsInGrabRange.Add(grabbable);
78      }
79
```

```
80      // when an obj exits our grabbing zone, we remove it from our Grabbable list
81      void OnTriggerExit(Collider c)
82      {
83          GameObject grabbable = isObjGrabbable(c.gameObject);
84          if(objsInGrabRange.Contains(grabbable))
85              objsInGrabRange.Remove(grabbable);
86      }
87
88      void FixedUpdate()
89      {
90          if (controller == null)
91              return;
92
93          // if user is trying to grab, disable hand colliders (to avoid pushing away
                items)
94          if (controller.GetButtonDown(grabButton))
95          {
96              if (grabbedObj == null)
97                  SetCollidersStatus(handColliderParent, false);
98          }
99          else if (controller.GetButtonUp(grabButton)) // execute pick up action
100         {
101             if (grabbedObj == null)
102             {
103                 GrabObject(GetNearestGrabbableObj());
104                 if (!grabbedObj)
105                     SetCollidersStatus(handColliderParent, true);
106             }
107             else    // execute release action
108             {
109                 ReleaseObject(); // release and throw item
110                 StartCoroutine("DelayedEnableCollidersCoroutine");
111             }
112         }
113     }
114
115     // FUNCTIONS
116     // [ . . . ]
117 }
```

Listing 3:
HydraHandInteraction.cs (no functions)

Thanks to this script our hands are now able to pick and hold items, operate them (sending them messages about what buttons the user has pressed) and release them (or throw them, if they were released with during hand movement).
The figure 28 shows how our “Grabber” object will look like in our hand.

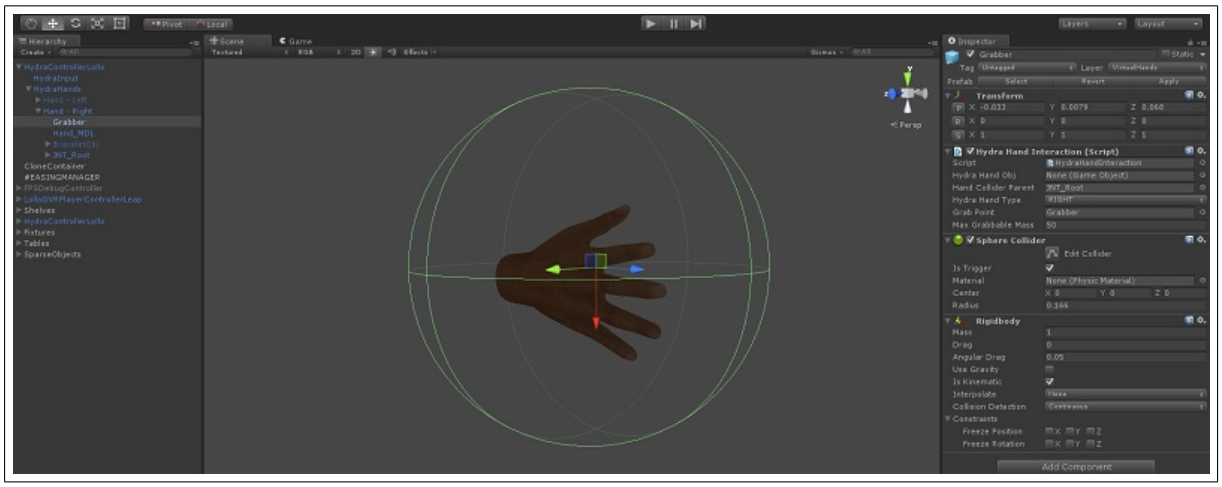

Figure 28:
Unity Editor: The grabber object in our virtual hands (Hydra)

To add the same abilities on the Leap Motion's hands we followed the same procedure but using a different C# script called "LeapHandInteraction.cs", which rely on gestures instead of buttons: making a fist to grab items or moving the thumb to shoot a weapon.

8.5.2 Preparing the Main Menu

Since we want to test different UI interactions we're going to make two different kind of main menu: a standard, 2D non-diegetic menu and a 3D, interactive, spatial menu.

For the 2D version we simply create a 2D GUI in unity, place in front of everything inside the scene and the user can press the corresponding keys for each action.
As we'll see in section 9.2 this design is far from optimal for virtual reality usage, so we'll create a 3D version too.

For the 3D version we must use a different approach: first we create different gameobjects in the 3D space (one for every line of the menu).
Then we add *rigidbody* and *boxcolliders* components to each of them, so our virtual hands will be able to interact with them, move the lines and "play" with them.
Thanks to a little C# script called "ComeBack.cs" each line will come back in its place after a while, since we don't want the user to be able to definitively break the menu usability.

To allow our hands to select the chosen action, we must add some buttons on the side of each line: by pressing these buttons with our virtual fingers, the corresponding line will be executed.

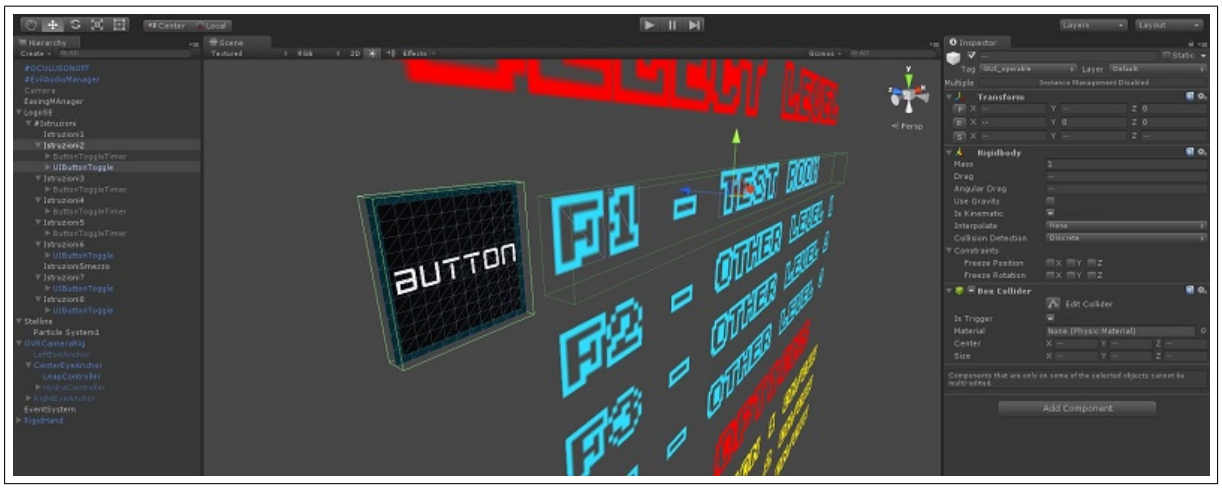

Figure 29:
Unity Editor: The 3D main menu.
Each lines now has a 3D physical object and has a button on the side

After creating the buttons we must attach a new C# script on the fingertip of the pointer finger for both hands (Leap's and Hydra's) called "GUIOperator.cs".
This will use a short-range raycast to tell the buttons if they're being hovered so they can create a glow effect.

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class GUIOperator : MonoBehaviour
5  {
6      public GameObject  rotator;
7      public float       castDistance = 25f;
8
9      private GameObject lastHit = null;
10     private Vector3    castDirection;
11     RaycastHit         hit;
12
13     void Update ()
```

```
14      {
15          if (rotator) castDirection = rotator.transform.forward*castDistance;
16          else castDirection = Camera.main.transform.forward*castDistance;
17
18          // raycast at short distance and check if something is hit
19          if (Physics.Raycast(transform.position, castDirection, out hit) &&
                isRaycastHittable(hit.transform.gameObject))
20          {
21              if (hit.transform.gameObject != lastHit)
22              {
23                  // stop hover state on old hit
24                  if (lastHit != null)
25                  {
26                      lastHit.SendMessage("HoverStop",
                            SendMessageOptions.DontRequireReceiver);
27                      lastHit = null;
28                  }
29
30                  // start hoverstate on new hit
31                  lastHit = hit.transform.gameObject;
32                  hit.transform.SendMessage ("HoverStart",
                        SendMessageOptions.DontRequireReceiver);
33              }
34          }
35          else if (lastHit != null && isRaycastHittable(lastHit))
36          {
37              // stop hover state on old hit
38              lastHit.SendMessage("HoverStop", SendMessageOptions.DontRequireReceiver);
39              lastHit = null;
40          }
41      }
42
43      // FUNCTION: hover only on GUI_operable objects
44      bool isRaycastHittable(GameObject rcHit){
45          if (rcHit.tag == "GUI_operable")
46              return true;
47          return false;
48      }
49  }
```

Listing 4:
GUIOperator.cs

8.5.3 Preparing 3D objects

Our room is full of interactive objects which all can be picked up, held in hand, swung around an dropped/threw away.

To make them look good once grabbed in our hands, we must move the center of they're *pivot point* in the right place where we want to hold them and with what rotation.
The figure 30 shows some objects (a vase, a gun and a sword) with their pivot points which tells where and how they will be held.

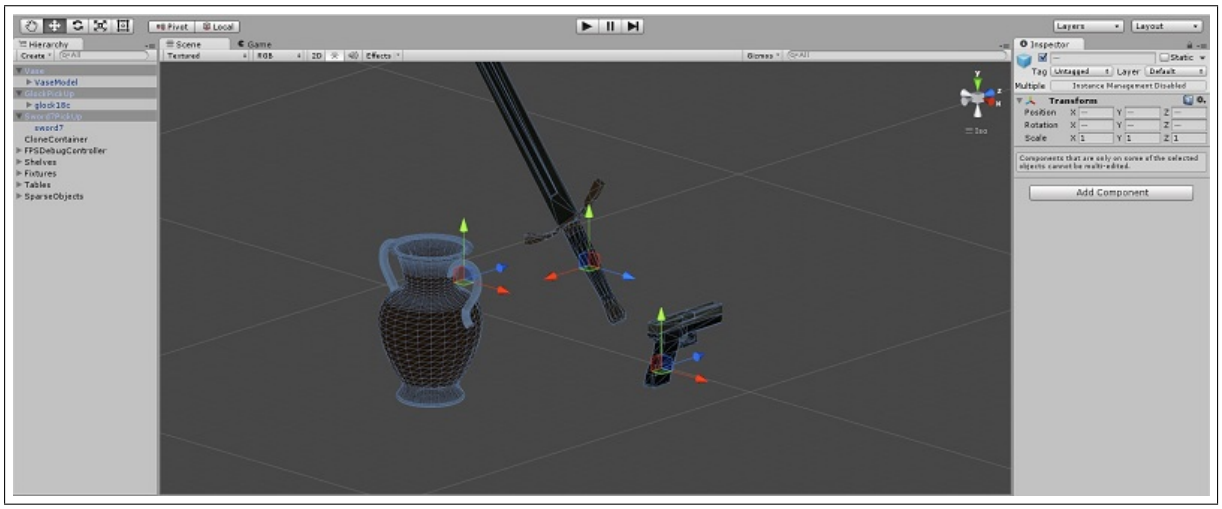

Figure 30:
Unity Editor: Objects Pivot Point

8.5.4 Preparing the in-game UI

As for the main menu, we're going to implement two different types of in-game informational displays: a standard, 2D non-diegetic HUD (as we can see in almost all FPS videogames) and a 3D, interactive object.

Being this is a simple test we must think about what we want to show in these displays: since we're going to use our two hands to grab and use objects, we decided to show info about what is being held and, if it's a gun, how much ammo we have left. If nothing is held, we show a clock since the user has an headset on and he has no way of checking real time without taking it off, so it can be considered useful.

About the 2D HUD we will use the same approach of the 2D main menu: they will be a 2D layer, drew above anything on the bottom left and right of our screen.
As we'll see in section 9.3 this UI design can't work at all inside a virtual reality context, so a 3D version is required.

For the 3D version I've chosen to make it a double smart watch, one for each hand, wore on each wrist.

For both version, to be able to show the information we want we can simply attach a C# script called [InfoDisplay.cs].

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class InfoDisplay : MonoBehaviour
5  {
6      private static InfoBracelet braceletLeft = null;
7      private static InfoBracelet braceletRight = null;
8      private TextMesh         text;
9
10     public enum BraceletType
11     {
12         Left,
13         Right,
14         None
15     }
16     public BraceletType braceletArm = BraceletType.Right;
17
18     // display info
19     private string holdedObject = "";
20     private string additionalInfo = "";
21     private float seconds, minutes, hours;
22
23     void Start ()
24     {
25         // start with clean display
26         braceletLeft = null;
27         braceletRight = null;
28         ResetDisplay();
29         text = GetComponentInChildren<TextMesh>();
30         // display time
31         if (text)
32             StartCoroutine("UpdateDisplayTimeCoroutine");
33     }
34
```

```
35      void Update ()
36      {
37          if (!text)
38              return;
39          // if we're holding something, update the display
40          if (holdedObject != "")
41          {
42              text.text = "-- HOLDING --\n"+holdedObject;
43              if (additionalInfo != "")
44              {
45                  text.text +="\n\n"+additionalInfo;
46              }
47          }
48      }
49
50      private IEnumerator UpdateDisplayTimeCoroutine()
51      {
52          while (true)
53          {
54              if (holdedObject == "")
55              {
56                  text.text = "--------------\n" + GetTimeFormatted() +
                        "\n--------------";
57              }
58              yield return new WaitForSeconds(0.5f);
59          }
60      }
61
62      // FUNCTIONS
63      // [ . . . ]
64  }
```

Listing 5:
InfoDisplay.cs (no functions)

Figure 31 shows how our smartwatch will look like once finished.
It will be wore with the display facing down on the wrist allowing the user to glance at it even when operating objects (like guns).

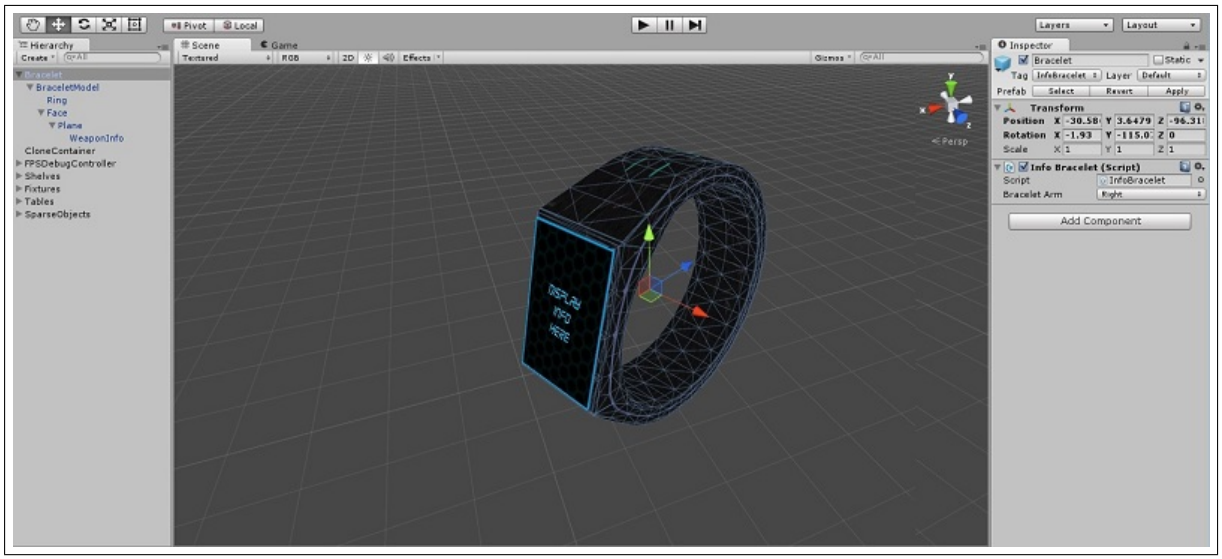

Figure 31:
Unity Editor: The smartwatch as 3D spatial UI

8.5.5 Preparing guns

We have guns in our scene, and we want to be able to use them other than simply holding them.

Since our hands interaction scripts already sends messages to grabbed objects (see section 8.5.1) we simply need to use a script able to intercept them.
We're going to attach to our guns a script called "GunBehaviour.cs" which will:

- inform our UI to show a new information (the ammo count)
- catch messages to about hands actions to shoot and enable/disable crosshair
- actually shoot our gun, decreasing ammo count and adding impact force on the hit object
- give feedbacks: on shoot we want to simulate weapon recoil, add graphical effects on the gun and on the hit object (fire effect and impact decal), show a faint bullet trial, play shoot and reload sounds.

8.5.6 Preparing in-game menu

As for the in-game UI we're going to implement two different kind of in-game menu: a standard, 2D non-diegetic menu (as the one we're already used to) and a 3D, interactive diegetic object.

Since the 2D version, implemented in a very similar way of the 2D UI, will show in section 9.5 all the problems of the 2D main menu, a 3D version is required, and ti integrate it in our virtual environment we're going to create an usable 3D talbet.

After creating the 3D gameobject we need to attach to it the C# script called "TabletBehaviour.cs" which allows it to work and show the menu, operable through our virtual hands.

```
using UnityEngine;
using System.Collections;

public class TabletBehaviour : MonoBehaviour {

    // PUBLIC
    public string            objName = "";
    public GameObject     infoPanel;
    public GameObject     usablePanel;

    // PRIVATE
    private InfoBracelet.BraceletType handHolding = InfoBracelet.BraceletType.Right;

    void Start ()
    {
        // correct obj name if left empty (for interface display)
        if (objName == "")
            objName = gameObject.name;
    }

    void ButtonClick(string clickValue)
    {
        // do the action of the clicked option
        switch (clickValue)
        {
            case "Reload":
                        StartCoroutine("DelayedLoadCoroutine", "PlayRoom");
                        break;
            case "Menu":
                        StartCoroutine("DelayedLoadCoroutine", "Intro_OCU");
                        break;
            case "Quit":
```

```
33                          Application.Quit();
34                          break;
35          }
36      }
37
38      public void Grabbed (InfoBracelet.BraceletType brac =
            InfoBracelet.BraceletType.Right)
39      {
40          // on pick ustart booting up to show the menu
41          handHolding = brac;
42          SetNewLayer(gameObject, LayerMask.NameToLayer("Default"));
43          InfoBracelet.GetBracelet(handHolding).SetHoldedObjectName(objName);
44
45          StartCoroutine("BootingCoroutine");
46      }
47
48      public void Released()
49      {
50          // on release, start the shutdown sequence
51          StartCoroutine("ShuttingDownCoroutine");
52      }
53
54      // show the "booting up" sequence for some seconds before showing the menu
55      private IEnumerator BootingCoroutine()
56      {
57          if (usablePanel)
58              usablePanel.SetActive(false);
59          if (infoPanel)
60          {
61              infoPanel.SetActive(true);
62              infoPanel.GetComponentInChildren<TextMesh>().text = "... Booting ...";
63          }
64          yield return new WaitForSeconds(1.5f);
65          if (usablePanel)
66              usablePanel.SetActive(true);
67          if (infoPanel)
68          {
69              infoPanel.SetActive(false);
70          }
71      }
72
73      // show the "loading" screen for some seconds berofe executing the requested
            action
74      private IEnumerator DelayedLoadCoroutine(string level)
75      {
76          if (usablePanel)
77              usablePanel.SetActive(false);
78          if (infoPanel)
79          {
```

```
80              infoPanel.SetActive(true);
81              infoPanel.GetComponentInChildren<TextMesh>().text = "... Loading ...";
82          }
83          yield return new WaitForSeconds(1.5f);
84
85          Application.LoadLevel(level);
86      }
87
88      // show the "shutdown" display for some seconds before turning off the screen
89      private IEnumerator ShuttingDownCoroutine()
90      {
91          if (usablePanel)
92              usablePanel.SetActive(false);
93          if (infoPanel)
94          {
95              infoPanel.SetActive(true);
96              infoPanel.GetComponentInChildren<TextMesh>().text = "... Shutting Down
                    ...";
97          }
98          yield return new WaitForSeconds(1.5f);
99          if (infoPanel)
100             infoPanel.GetComponentInChildren<TextMesh>().text = "";
101     }
102 }
```

Listing 6:
TabletBehaviour.cs

9 Usability analysis

After the creation of the test room, it's finally time to enter it and see how the ideas and implementations of section 8 will "feel" when experienced, and if our input devices are up to the task of granting high levels of presence.

9.1 Preliminary test: hardware devices

First of all, we can finally try inside a virtual environment the accuracy of our test hardware devices: the Oculus Rift DK2 headset, the Leap Motion and the Razer Hydra.

9.1.1 Oculus Rift

While this device shows immediately all the problems and limitations described in section 7.1.4, his presence-delivery factor is immediate as well.
The first thing one experience by putting the headset on is the concept of spatial presence: even in a simple virtual environment like our test room, with the Oculus Rift the user is clearly able to state depth and distance of objects inside the environment with only little error, which was impossible through a normal display.
The head rotation and translation tracking feels already good enough thanks to the low latency, and lot of remaining previously described problems are going to be resolved with the final product on the end of 2015.

9.1.2 Leap Motion

On paper, Leap Motion states the ability of sub-millimiter accuracy in hands tracking with a maximum range of 61 cm.
As we've seen, though, it was born to be used from a "under your palms" position, not Oculus-mounted like we're using it.

With this use, its accuracy immediately seems much lower: while it works great if the user try to examine his palms at close distance, it starts quickly to lose tracking accuracy the further the hands go, and it's even worse if hands are not kept forcefully open, with fingers wide spread.
In figure 32 on the right we can see how the tracking fails to determine the correct hands orientation and finger position at only 30 cm of the controller when the user doesn't keep hands widely open.

So, on the field, Leap Motion tracking quality when Oculus-mounted has shown to be:

- **with hands wide open and clearly visible fingers:**
  the tracking quality is pretty close to the one stated on paper. Hands translation

and position in 3D space are well tracked, as well hands orientation.
Tracking quality degradation and jitter starts at a distance of 40 cm or more from the controller.

- **with hands in arbitrary pose, with fingers partially or totally hidden:** in this case the tracking doesn't work well at all. Gesture are not recognized even at close distances, and making them causes immediate heavy jitter and positional and orientation errors on virtual hands as well.
  As we'll see later, this will have a great impact on gameplay possibilities and natural way of interactions with this device.

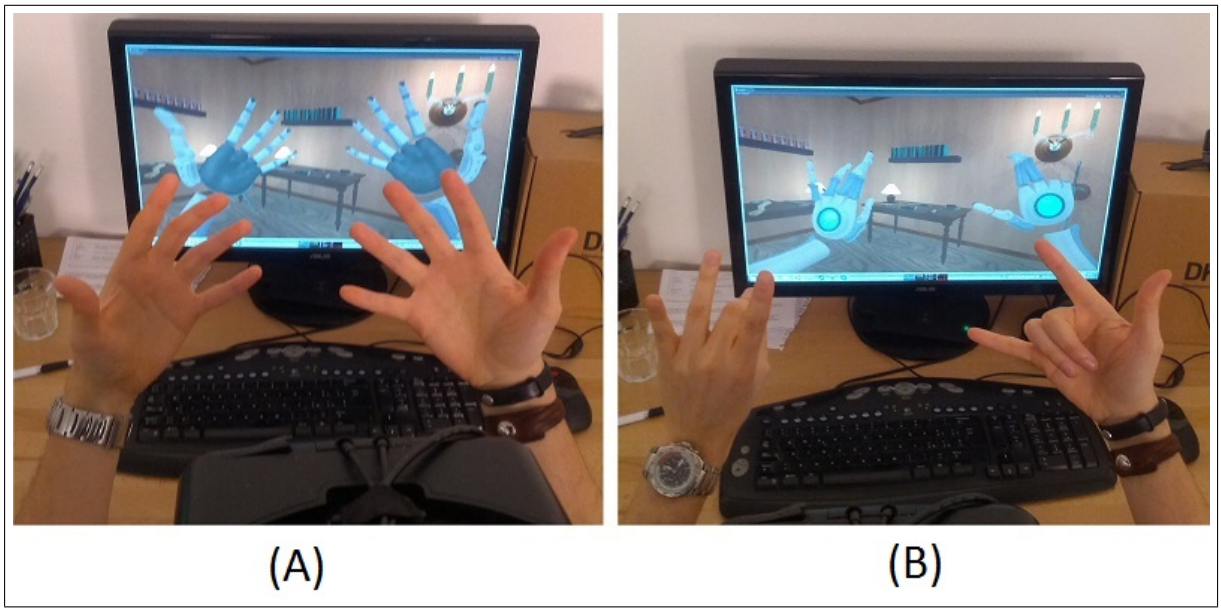

Figure 32:
Leap Motion tracking quality in different hands position:
- (A) Good tracking quality
- (B) Bad tracking quality

Even with this problem, though, the possibility of being able to "see your own hands" inside the virtual environment has a great impact on the user, who starts to empathize with his avatar much more quickly, granting great presence sensation.

9.1.3 Razer Hydra

The Hydra, on the other hand, delivers another quality of experience.
On paper it states that the optimal distance of controllers is 50 cm from the base.

Testing it on the field, with no interference from other magnetic fields, has shown that the tracking remains of good quality even at greater distances:

- **under 85 cm:**
  translation and orientation on virtual hands is near perfect, with a little jitter of virtual hands (of around 0.5 cm of virtual world space) that starts over 75 cm.
- **between 85 and 120 cm:** translation and orientation tracking still works fine, but the jitter effect increase to about 4 cm of virtual space.
  This means that while still usable, performing delicate, precision tasks (e.g. aiming with a gun or selecting items on a graphical interface) will be much harder.
- **over 120 cm:**
  While translation and orientation tracking still works good, the jitter effect in virtual space is now around 40 cm, making impossible to accomplish any kind of task.

Obviously Hydra, unlike the Leap Motion, doesn't track user fingers, so to make gestures the user must rely on controller's buttons.

9.2 The main menu

Before entering the room, the player is exposed to a classical main menu, where he's able to select different levels to load or to configure options.
Since this is a simple test, only one level is really available (the test room), so this main menu is useful only for interface-testing purpose.

The first implementation is the classical way of doing a main menu: as seen in section 8, the first test will be done with a standard, non-diegetic 2D menu, where the user can select different choices with the use of the mouse, the keyboard or a gamepad.

Figure 33:
The Main Menu: non-diegetic

In a virtual reality context, this design shows immediately some problems:

- **incoherence:**
  while the animated starfield in the background is in 3D space, with particles moving from afar towards the player, the menu is a 2D layer on a fixed position.
  This means that by looking around the starfield will react correctly, while the menu will always be at the center of the view.
  This leads to confusion and loss of control, and as stated by Llorach, Evans, and Blat[18], this quickly provide a sense of simulator sickness.

- **bad control:**
  selecting different choices can be made only through input like keyboard or gamepad. Since the user has an headset on, he can't see these devices in the real world, so it's a bad decision by default, and they add a layer of interface able to lower immersion factor.

Wanting to place this situation in Csíkszentmihályi's flow concept as in figure 5, we would be in "apathy" or even "boredom".

We can now transform from 2D to 3D the menu, as shown in section 8.5.2.
By making every line of text a 3D object, selectable with virtual hands the experience is now different: coherence is maintained, since the user can look around and the menu (being now a 3D object in space) will move out of the view accordingly.
Selecting with virtual hands grants a great sense of immersion, destroying the artificial layer on interaction provided with standard inputs.
The scene, however, still gives a sense of "stillness": by touching text with the virtual hands it gets selected but it doesn't move like a real, 3D object would do.

Another step is to add physics to every lines of text, allowing the user to use virtual hands to punch, push and move around the text (see figure 34-A), providing a great sensation of usability-coherence, and pressing the virtual buttons to select the corresponding options (see figure 34-B).

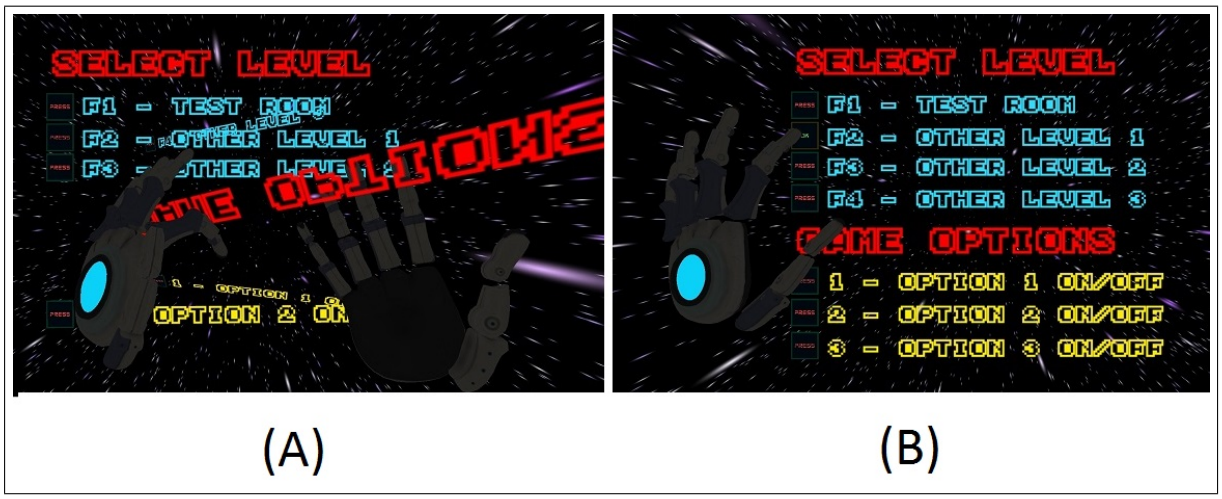

Figure 34:
The Main Menu: spatial
- (A) pushing lines of text with Leap Motion
- (B) selecting buttons with Leap Motion

After the lines are pushed away they come back to their original position (thanks to a C# script called "ComeBackObject.cs"), so that the user can't "break" the menu usability.

Since the 3D objects of the menu are close to the camera, the Leap Motion controller behaves pretty well: the range of its usage is optimal and selecting buttons as shown in figure 34-B is done through a gesture well recognized by the controller.

The Hydra controller performs great in this test too, giving the extra ability of selecting a menu button by pressing a controller button when the hand hovers the menu button, other than just touch the menu button with the virtual hand.

With both input devices we can put the experience in the "control" section of Csíkszentmihályi's flow, and we already now by definition that higher user control grants higher level of presence.
The best interaction method regarding presence, however, is the Leap Motion.
That's because by following the Norman's concept of *natural mapping*, the user interface operations are best suited to be done with free hand and not by wielding a controller.

Obviously with this kind of menu design, the concept of "suspension of disbelief" must be applied: a main menu like this one is nothing the user can experience in real life, and the player must accept it.
The main menu could have become a diegetic component, for example through a real-like interface (e.g. a tablet), as we'll do instead for the in-game menu (see section 9.5).

9.3 The in-game user interface

For the in-game UI we will follow the same path of the main menu (see section 9.2).

The first implementation (figure 35) is the standard, non-diegetic 2D HUD seen in many videogames.
This solution have all the problems stated for the non-diegetic main menu and adds a new one: it forces the user to heavily strain his eyes.
While the 2D main menu was centered in view, this non-diegetic UI adds readable details on the peripheral vision of the user: this means that to read the UI, the user must rotate his eyes on the bottom-left or bottom-right of his vision (remember that the Oculus Rift display is only a few centimeters from the eyes), since he can't turn his head to center the UI in the view (being a 2D element, it will always be anchored in peripheral vision).

Figure 35:
In-game UI: non-diegetic

As seen in section 8.5.4, we've created 3D smartwatch as substitute of this 2D UI: in this way the UI become a real 3D object inside our virtual world, and can be considered diegetic UI.

As we can see in figure 36 , a diegetic UI works a lot better inside our virtual environment: the user can experience total control by moving it in view when needed and ignoring it when not needed granting high control of the situation.
Additionally, the concept of "suspension of disbelief" is totally shattered since the player doesn't need to accept fictional items (as usually an user interface in videogame is) but uses real-like items in natural ways.
This applies to Norman's natural mapping too: if you want to check your smartwatch, just do the same movement you usually do in real life.
All these factors keep the player immersed in the virtual environment, maintaining high level of presence factor.
Applying these smartwatch to Hydra's or Leap Motion's hands has no difference on impact factor, but the findings of section 9.1 must be kept in mind.

Figure 36:
In-game UI: diegetic
- (A) checking ammo count on right (virtual) hand
- (B) checking real world time on left (virtual) hand
- (C) experiencing (A) in real world with Razer Hydra

9.4 Virtual object handling

We can now experience the usability of our concept of "grabbing and handling 3D objects" designed in section 8.5.1.

First, we'll test the act of picking up a book on a table inside our virtual environment.
Our fears regarding the tracking accuracy of Leap Motion controller are now proven true: it's almost impossible to mimic the "pick up" gesture at acceptable distances without breaking the Leap Motion's tracking.
That means that to manage to pick up an object with Leap Motion controller, we need to keep our hand really close to the face (see figure 37-A).

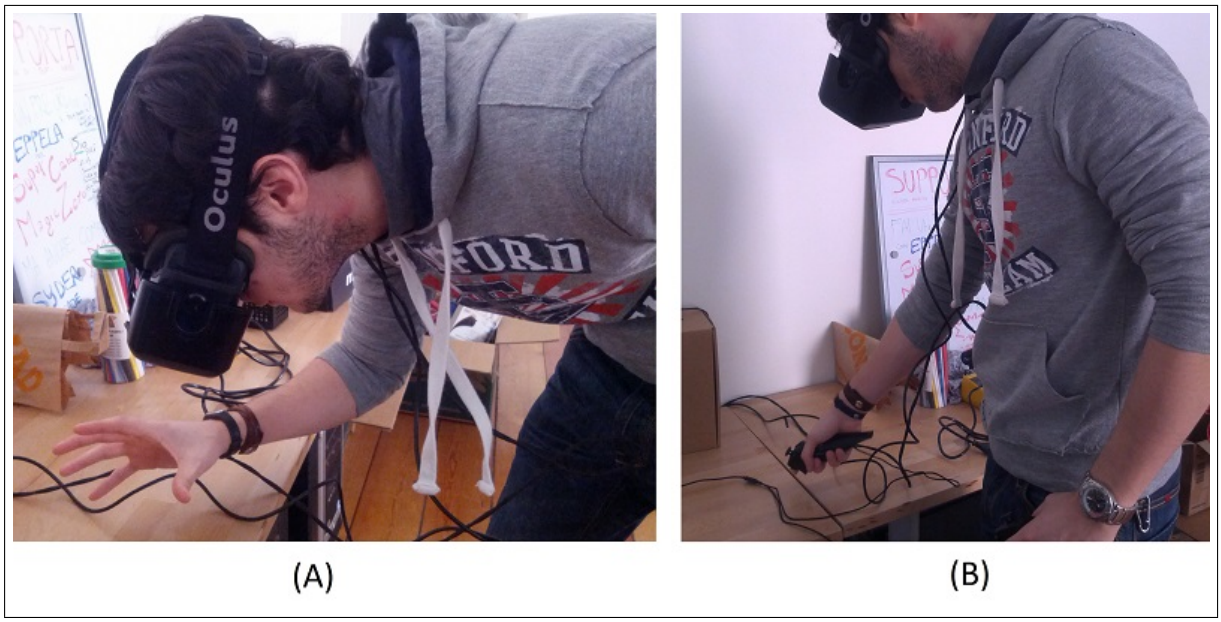

Figure 37:
Picking up items (human view) on a virtual table
- (A) using Leap Motion
- (B) using Razer Hydra

With the Razer Hydra, as shown in figure 37-B, the action is performed in a totally natural way, without the need of following constraints like with the Leap Motion.

This problem persists in handling the grabbed objects, as shown in figure 38:

- **with Leap Motion controller:** the user is forced to maintain unnatural hands position to keep the tracking accuracy at acceptable levels.
  In addition, the hands must be kept in front of the face, since the tracking zone is limited.
  This shatter the player controls and break immersion, since the user is forced to follow usability rules to be able to correctly perform tasks.

- **with Razer Hydra controller:** the action seem more natural; without the Leap Motion's constraint, the user can move around his hands and arms without affecting the tracking capability, which leads to a virtual world able to perform much closer to the real world, granting an higher level of immersion and control for the final user, even if some tasks (as "grabbing") are done through controller's buttons.

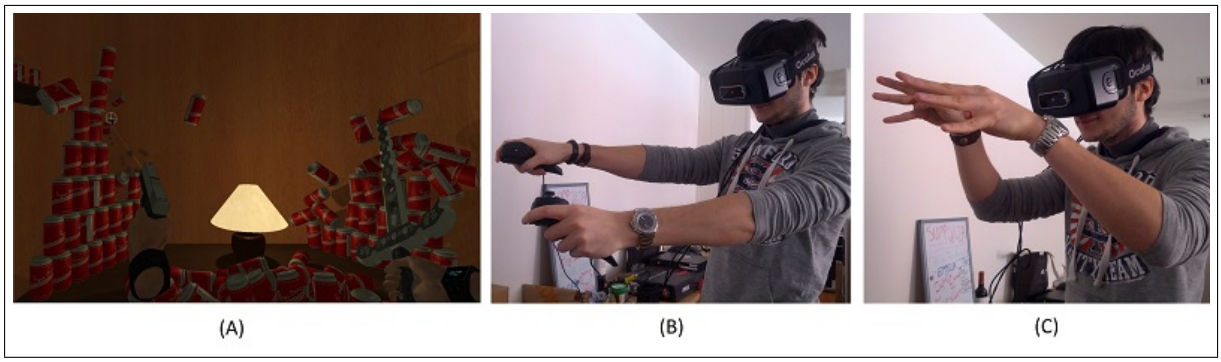

Figure 38:
Handling a gun in left hand and a sword in right hand at the same time while interacting with virtual environment
- (A) scene in virtual environment
- (B) experiencing (A) in real world, using Razer Hydra
- (C) experiencing (A) in real world, using Leap Motion

When handling objects then, the Leap Motion show its limits: by analyzing Csíkszentmihályi's flow, we can state that this controller lowers the skill of the player (since doing correctly even little actions become more challenging in the virtual world rather than in the real world) leading to a loss of control for the player; in addition, by forcing the user to follow movement constraints, it breaks the Norman's barrier of natural mapping.

The Razer Hydra, on the other hand, displays great tracking accuracy on items handling: this leads to a great sense of control on behalf of the player, and follows perfectly Norman's natural mapping principle, since it "feels" better to actually hold something in real world (the Hydra controller) when holding something in the virtual world, even if the shape is different, and maintaining the user experience in the wanted section of flow (see figure 5).

Figure 39:
Grabbing items in virtual environment with the Razer Hydra
- (A) the user can extend his hand to his arm's length to reach an object
- (B) by using the trigger on the Hydra's controller he grabs the item near the virtual hand

9.5 The in-game menu

For the in-game menu we will follow the same path of the in-game UI.
The first implementation (see figure 40) recall the in-game menu we're used to see in many other videogames.

Differently from the non-diegetic in-game UI, this can be considered "usable" since it's in the center of the user view and doesn't create eye strain, but it displays all the problems found for the non-diegetic main menu (see section 9.2), and by being during gameplay it feels even worse: it confuses the user which have control over his virtual hands but he can't use them to operate the in game menu (since it's on a 2D layer drew in front of other 3D objects, like the virtual hands).

We can state that while this design could have been acceptable (but far for optimal in virtual reality) in a main menu, during gameplay it has more issues:

- confuses the player by changing the rules of the virtual reality (suddenly he's not able to operate everything with his hands anymore), a big reminder of "you're not really there"

- shatter user control factor, forcing him to switch input device to operate the menu
- since a "main menu analogy" doesn't exists in real world, the user must rely on the "suspension of disbelief" factor
- ultimately breaks presence

Figure 40:
The in-game menu: non-diegetic

As for the in-game UI, we're now going to use the 3D diegetic element (the tablet designed in section 8.5.6) inside our virtual environment.
It's been designed to be usable in a way that the user will be able to feel "real":

- it's been placed on a table, forcing the user to reach it and grab it
- on picking it up it's not immediately usable but it shows a "booting" message before showing the operable menu
- it's used like a real tablet: it's held with one hand and used with the other one
- if the user selects an action on the tablet interface, a brief message of "loading" is shown before executing the requested action

- if the user drops it, a brief message of "shutting down" is shown before turning the tablet's screen off.

Figure 41:
The in-game menu: diegetic
- (A) the user reach the tablet with the virtual hand
- (B) upon grab, the tablet starts the booting sequence
- (C) once operable, the user can interact with the interface with the free hand
- (D) upon selection the interface shows a loading screen before executing the requested action

This diegetic implementation of the in-game menu grants high level of user presence, since the object is real-like (we're all used to the concept of using a tablet), there's no need to rely on suspension of disbelief factor, the engrossment of the player and challenging level are kept high (he needs to commit high focus to correctly operate the virtual device, but in a positive and productive way) so he's in the wanted zone of Csíkszentmihályi's flow.

While figure 36 shown the tablet used only with the Hydra, we can predict that in this case the Leap Motion will work well too.
While using only the Leap Motion controller could be too hard for picking up the tablet (see section 9.4), an hybrid solution can be used: since the Leap Motion has proven good for graphical interface operations (see section 9.2), the tablet can be grabbed with the

Hydra and operated with the Leap Motion.
This means that we hold the tablet with the grabbing virtual hand (where we hold the Hydra's controller in real world) and we operate the device with our free hand (controlled with Leap Motion's controller): this grants the user the correct haptic feedback of having a item in one hand and using the controller-free hand to operate the tablet, further increasing the presence factor.

Figure 42:
The virtual tablet usage
- (A) performing the same action of figure 41-C with Hydra+Leap Motion combo
- (B) experiencing (A) in real world (Leap Motion)
- (C) experiencing figure 41-C in real world (Razer Hydra)

10 Final conclusions

While these technologies are rather new, on the software side they've proven to already be stable and of simple implementation: in section 8 we've managed to create a virtual environment able to support Oculus Rift, Razer Hydra and Leap Motion with no big effort.

While virtual reality brings new levels of user immersion and presence sensation it also brings, however, the need to explore new gameplay, new ways of interaction and new graphical interfaces design and the need to change and adapt pre-existing knowledge of virtual environments design, since a lot of methods and design ideas that have proven to work well before wont' work in the same way in virtual reality.

Regarding graphical user interfaces we've seen that only few grant to keep high presence factor on the player.
Non-diegetic UI have proven to be horrible design choice, in some cases not even able to be considered "usable".
To use meta UI we must keep in mind that the user will be able to see them only through his peripheral vision, so while the first example of figure 9 (Call of Duty) can work in virtual reality, the second one (Gran Theft Auto) surely won't, since it adds too much detail on a fixed-in-view 2D element.
In the context of virtual reality, wherever a diegetic or spatial design is allowed it must be used.

Regarding input devices, when designing a virtual environment we must always remember "what is good for what": while maintaining the general idea of Norman's natural mapping, we need to balance this concept with maintaining high control on the final user.
If he wants to perform a specific action, we must design a way of interaction able to feel "as natural as possible" to maintain presence factor, while at the same time allowing him to reach his goal without the need of extra difficulty (or, in other words, maintaining a balance between player's skill level on doing a task and the challenge level of this task, as we can see in Csíkszentmihályi's flow).
This has been proven in the task of grabbing an object: while it can feel more natural using a controller-free interaction like the Leap Motion, in this case it brings many limits and constraints, so higher presence is maintained using the Hydra, even if it feels less natural (since the user is holding an object in the real world but he's not holding anything in the virtual world).

More generally, there are still some problems with virtual reality.

Oculus Rift DK2 is already an old prototype (preorders starts on March 19th of 2014),

the Crescent Bay prototype is already much better, and a final, consumer version is coming by the end of 2015, bringing the promise of revolutionizing the already-revolutionary Oculus Rift experience.
This means better immersion, easier presence factor and less risk of simulation sickness, allowing for more "daring" gameplay and general interaction choices of design inside virtual environments.

The Razer Hydra successor, the *Stem System*, is already been announced[7], promising better tracking and wireless controllers.

Leap Motion co-founder David Holz states on his blog[8] that a new generation of the Leap Motion sensor, codename "DragonFly", is under development with virtual reality usage in mind.
It promises better tracking distance, angle and overall tracking quality.

The problem of user movement in virtual environment, not tested as part of this study, still persist: for player movement we rely on Hydra's controllers sticks.
As stated in section 5.1, a natural way of body movement should theoretically be the best option regarding player's presence factor, but it brings the problem of "how can one move in virtual world by walking without crashing into a wall in real world".
Some are trying to fill this controller gap, like *Virtuix Omni*[9], a device able to track your body movements (like running, walking and turning around) while keeping you in place, but they weren't yet available during this study and so we were unable to test them.

In conclusion, some of the findings of this study will still be right upon the release of these new devices while some other may be prone to change, but something is certain: this time virtual reality is happening for real, bringing a whole new level of immersion, enjoyment and presence regarding videogames, a new way of interaction with our world in lots of areas (see section 3.3) and new challenges for virtual environments designers.

[7]See url: *http://sixense.com/wireless*

[8]See url: *http://blog.leapmotion.com/leap-motion-sets-a-course-for-vr/*

[9]See url: *http://www.virtuix.com/*

References

[1] Jonathan Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication*, 42(4):73–93, 1992.

[2] Matthew Lombard and Theresa Ditton. At the heart of it all: The concept of presence. *Journal of Computer-Mediated Communication*, 3(2):0–0, 1997.

[3] Emily Brown and Paul Cairns. A grounded investigation of game immersion. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1297–1300. ACM, 2004.

[4] Michael Mateas and Andrew Stern. Interaction and narrative. *The game design reader: A rules of play anthology*, 1:642–669, 2006.

[5] Daniel Mestre, Philippe Fuchs, A Berthoz, and JL Vercher. Immersion et présence. *Le traité de la réalité virtuelle. Paris: Ecole des Mines de Paris*, pages 309–38, 2006.

[6] Ron Tamborini and Paul Skalski. The role of presence in the experience of electronic games. 2006.

[7] Erik Fagerholt and Magnus Lorentzon. Beyond the HUD: user interfaces for increased player immersion in fps games. 2009.

[8] Tilo Hartmann, Christoph Klimmt, and Peter Vorderer. Telepresence and media entertainment. *Immersed in media: Telepresence in everyday life*, pages 137–157, 2010.

[9] Paul Skalski, Ron Tamborini, Ashleigh Shelton, Michael Buncher, and Pete Lindmark. Mapping the road to fun: Natural video game controllers, presence, and game enjoyment. *New Media & Society*, page 1461444810370949, 2010.

[10] Ron Tamborini and Nicholas D. Bowman. Presence in video games. *Immersed in media: Telepresence in everyday life*, pages 87–109, 2010.

[11] David Westerman and Paul D Skalski. Computers and telepresence. *Presence and Popular Media: Understanding Media Users' Everyday Experiences: Telepresence in Everyday Life*, page 63, 2010.

[12] Anthony M. Limperos, Michael G. Schmierbach, Andrew D. Kegerise, and Frank E. Dardis. Gaming across different consoles: exploring the influence of control scheme on game-player enjoyment. *Cyberpsychology, Behavior, and Social Networking*, 14(6):345–350, 2011.

[13] Daniel M. Shafer, Corey P. Carbonara, and Lucy Popova. Spatial presence and perceived reality as predictors of motion-based video game enjoyment. *Presence: Teleoperators and Virtual Environments*, 20(6):591–619, 2011.

[14] Eric Malbos, R Rapee, and Manolya Kavakli. Behavioral presence test in threatening virtual environments. *Presence*, 21(3):268–280, 2012.

[15] Frank Weichert, Daniel Bachmann, Bartholomäus Rudak, and Denis Fisseler. Analysis of the accuracy and robustness of the leap motion controller. *Sensors*, 13(5):6380–6393, 2013.

[16] D. Bassily, C. Georgoulas, J. Guettler, T. Linner, and T. Bock. Intuitive and adaptive robotic arm manipulation using the leap motion controller. In *ISR/Robotik 2014; 41st International Symposium on Robotics; Proceedings of*, pages 1–7. VDE, 2014.

[17] Parth Rajesh Desai, Pooja Nikhil Desai, Komal Deepak Ajmera, and Khushbu Mehta. A review paper on oculus rift-a virtual reality headset. *arXiv preprint arXiv:1408.1173*, 2014.

[18] Gerard Llorach, Alun Evans, and Josep Blat. Simulator sickness and presence using hmds: comparing use of a game controller and a position estimation system. In *Proceedings of the 20th ACM Symposium on Virtual Reality Software and Technology*, pages 137–140. ACM, 2014.

[19] Erik Saar. Touching reality: Exploring how to immerse the user in a virtual reality using a touch device. 2014.

[20] Jonmichael Seibert. *An exploratory study on virtual reality head mounted displays and their impact on player presence.* PhD thesis, 2014.

[21] Samuel Taylor Coleridge and Arthur Symons. *Biographia literaria*, volume 2. JM Dent and sons; EP Dutton, 1996.

[22] Mihaly Csikszentmihalyi and Isabella Selega Csikszentmihalyi. *Optimal experience: Psychological studies of flow in consciousness.* Cambridge University Press, 1992.

[23] Gérard Genette. *Narrative discourse: An essay in method.* Cornell University Press, 1983.

[24] John Chris Jones. *Design methods.* John Wiley & Sons, 1992.

[25] Jesper Juul. *Half-real: Video games between real rules and fictional worlds.* MIT press, 2011.

[26] Donald A. Norman. *The design of everyday things*. Basic books, 2002.

[27] John Ronald Reuel Tolkien. *The Tolkien Reader*. Ballantine Books New York, 1966.