FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Rendering and Editing Tools for Interactive 360º Video

MSc Dissertation

João Manuel da Silva Meira

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: Rui Pedro Amaral Rodrigues

Co-Supervisor: Rui Pedro da Silva Nóbrega

July 22, 2017

© João Manuel da Silva Meira, 2017

Rendering and Editing Tools for Interactive 360º Video

MSc Dissertation

João Manuel da Silva Meira

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

July 22, 2017

"If we knew what we were doing
it wouldn't be called research."

Albert Einstein

Acknowledgements

To my mother and father, and to my brother. To my girlfriend Sara, thank you for your patience and love.
To my supervisors, Rui Rodrigues and Rui Nóbrega, thank you for the guidance and strength.
And to João Jacob, thank you for the support.
To all the friends I made along the way, I may or may not be sorry for what's to come.

João Meira

Abstract

Video editing is essential for collecting and arranging multimedia content to fit a consistent publication, especially prominent in content delivery venues. Orthogonally, Virtual Reality (VR) experiences evoke in their users a feeling known as presence: multi-sensory assimilation of a Virtual Environment (VE). Combining this immersion with established video editing techniques, one arrives at the concept of immersive video editing.

One of the challenges of immersive video is 360° video editing – when the focus is no longer restricted, conditioning users' attention must be done by different means, namely virtual indicators. However, 360° videos are not trivial to edit in current 2D video editors, as immersive journalistic venues find, primarily due to the different image arrangement. Such indicators, called annotations due to their informational nature, are spatialised two or three-dimensional objects that are displayed as an overlay. To solve this issue, a video player capable of rendering these effects in real-time is proposed.

The purpose of this dissertation is to create, evaluate and compare a set of VE–interaction techniques that enable a set of authoring operations within a VR 360° video editor. As such, a prototype was developed with the Unity engine to provide a testing environment. It allows users to annotate 360° video using keyframing mechanisms. First, an annotation paradigm was developed, with each annotation being written in HTML and rendered onto a plane in 3D space. Second, interaction metaphors based on pointing, clicking and grabbing gestures were implemented, with slight modifications for a selected set of input devices. The HTC Vive approach employs the proprietary controller-wands like a laser pointing device, with the trigger button being used to indicate selection or dragging. Finally, the Leap Motion controller provides a more natural interface in which users can use their hands to point and use a pinching gesture to trigger selecting or dragging. Furthermore, within this scope additional questions arise: is a VE the most adequate for 360° video editing? Which data storage techniques are most appropriate to ensure compatibility among potential future implementations? In brief, which interaction techniques are more adequate for use in 360° video editing, in terms of efficiency, fatigue and naturalness?

To assess this, a usability test was conducted requiring users to perform annotating tasks of increasing complexity while telemetry was gathered by the system. Additionally, participants were surveyed by means of questionnaires to score each interaction technique on a System Usability Scale (SUS). The results of these experiments postulate that, when carefully transformed for VR, such interaction paradigms can be as effective as traditional devices. Currently the implemented prototype provides an environment that allows users to apply the researched techniques to the annotating of 360° video effectively.

***Keywords* — Immersive 360° Video, Multi-modal Interfaces, Virtual Reality**

Resumo

A edição de video é essencial para recolher e organizar conteúdo multimédia de modo a ajustar-se a uma publicação consistente, e é especialmente proeminente em cenários de entrega de conteúdo. Ortogonalmente, experiências em Realidade Virtual (RV) suscitam nos utilizadores um sentimento de presença: uma assimilação multi-sensorial de um Ambiente Virtual (AV). Combinando este conceito com técnicas de edição de vídeo já estabelecidas chega-se ao conceito de edição de vídeo imersivo.

Um dos desafios do vídeo imersivo é a edição de vídeo 360° – quando o foco deixa de estar restrito, o condicionamento da atenção dos utilizadores tem que ocorrer de outra forma, nomeadamente por meio de indicadores virtuais. No entanto, a edição de video 360° não é trivial nos editores de vídeo 2D atuais, como se verifica em espaços jornalísticos. Tais indicadores, chamados de anotações dada a sua natureza informativa, são objetos bi- ou tri-dimensionais espacializados que são mostrados como um *overlay*. Para abordar esta questão, propõe-se um *player* de vídeo capaz de renderizar este efeitos em tempo-real.

O objetivo desta dissertação é criar, avaliar e comparar um conjunto de AV-técnicas de interação que possibilitem operações de autoria dentro de um editor de vídeo 360° em RV. Como tal, foi desenvolvido um protótipo em Unity para proporcionar um ambiente de teste. Este permite aos utilizadores anotar vídeo 360° recorrendo a mecanismos de keyframing. Primeiro, foi desenvolvido um paradigma de anotações, com cada anotação a ser escrita em HTML e desenhada num plano em 3D. Segundo, foram implementadas metáforas de interação baseadas nas ações de apontar, clicar e arrastar, com ligeiras modificações de acordo com o conjunto de dispositivos de interação. A abordagem com o HTC Vive aplica o comando-varinha como um apontador laser, com o gatilho a ser usado para indicar seleção ou arrasto. Por fim, o controlador Leap Motion proporciona uma interface mais natural utilizando a mão do utilizador para selectionar ou arrastar com um gesto de belisque. Adicionalmente, surgem mais questões no âmbito deste protótipo: será um AV o mais adequado para edição de vídeo 360°? Que técnicas de armazenamento de dados são mais apropriadas para garantir a compatibilidade com implementações futuras? Em suma, quais técnicas de interação são mais adequadas para utilização em edição de vídeo 360°, em termos de eficiência, cansaço e naturalidade?

Para avaliar estas questões, foi realizado um teste de usabilidade no qual os utilizadores realizam tarefas de complexidade crescente enquanto o sistema recolhe dados telemétricos. Além disso foram entregues questionários para pontuar cada técnica de interação numa escala de System Usability Scale (SUS). Os resultados destas experiencias permitem postular que, quando transformadas para RV, estes paradigmas podem ser tão eficazes como em dispositivos tradicionais. Atualmente, o protótipo implementado providencia um ambiente que permite aos utilizadores aplicar as técnicas estudadas à anotação de vídeo 360° de forma eficaz.

***Palavras-chave* — Interfaces Multimodais, Realidade Virtual, Video 360° Imersivo**

Contents

1 Introduction **1**
- 1.1 Context . . . 1
- 1.2 Motivation . . . 2
- 1.3 Research Questions . . . 3
- 1.4 Objectives . . . 4
- 1.5 Contributions . . . 5
- 1.6 Document Structure . . . 6

2 Related Work **9**
- 2.1 Virtual Reality . . . 10
  - 2.1.1 Background . . . 10
  - 2.1.2 Defining Virtual Reality . . . 12
- 2.2 Augmented Reality . . . 13
- 2.3 360° Media and VR Rendering . . . 14
  - 2.3.1 360° Video, Multimedia Editing and 3D Modelling . . . 14
  - 2.3.2 Virtual Environments and Display . . . 21
- 2.4 Acquisition and Interaction Techniques . . . 27
  - 2.4.1 Hand Recognition . . . 27
  - 2.4.2 HCI and Interaction Interfaces in CAD and Virtual Environments . . . 29
- 2.5 Validation Solutions . . . 33
- 2.6 Brief Consumer VR Device Overview . . . 34
  - 2.6.1 3D Hand Sensors . . . 34
  - 2.6.2 Head-mounted VR Displays . . . 36
  - 2.6.3 Data and 3D Rendering . . . 37
- 2.7 Concluding Remarks . . . 37

3 Architecture of the Immersive 360° Annotation System **39**
- 3.1 Requirement Analysis . . . 39
  - 3.1.1 Key Concepts . . . 40
  - 3.1.2 Overall description . . . 42
  - 3.1.3 Constraints and Specific Requirements . . . 45
- 3.2 Early Prototype . . . 46
  - 3.2.1 General Description . . . 46
  - 3.2.2 Key Takeaways . . . 47
- 3.3 Proposed Solution Overview . . . 50
  - 3.3.1 System Context . . . 50
  - 3.3.2 System Architecture . . . 52
- 3.4 Interaction Techniques and Design . . . 53

3.4.1 Pointing at Objects . . . 53
3.4.2 Selecting Desired Elements . . . 56
3.4.3 Dragging Distant Objects . . . 60
3.4.4 Interface Design . . . 61
3.4.5 Menu Operation . . . 66
3.5 Summary . . . 70

4 System Implementation **71**
4.1 Rendering Techniques . . . 72
4.1.1 Rendering 360° video . . . 72
4.1.2 Positioning Content in 3D Space . . . 73
4.1.3 Content Authoring . . . 75
4.1.4 Rendering Annotations . . . 76
4.1.5 Rendering Keyframes . . . 77
4.2 Object Manipulation Techniques . . . 78
4.2.1 Manipulating Annotations . . . 78
4.2.2 Manipulating Keyframes . . . 79
4.3 Interpolation in Spherical Space . . . 79
4.3.1 Linear Interpolation . . . 80
4.3.2 Spline Interpolation . . . 81
4.3.3 Spherical Spline Interpolation . . . 83
4.4 Metadata Handling and Persistence . . . 85
4.4.1 Data Storage . . . 85
4.4.2 Tracking and Measurements . . . 88
4.5 Summary . . . 89

5 Evaluation **91**
5.1 Experiment and Results . . . 92
5.1.1 Overview . . . 92
5.1.2 Objectives . . . 93
5.1.3 Protocol and Specification . . . 93
5.2 Data Gathering and Assessment . . . 98
5.2.1 Directing User Gaze . . . 98
5.2.2 Measuring Dragging Precision . . . 102
5.2.3 Measuring Selection Precision . . . 104
5.2.4 Measuring Hand Motion . . . 107
5.2.5 Measuring Time Efficiency . . . 109
5.3 Usability Questionnaires . . . 110
5.3.1 System Usability Scale (SUS) Results . . . 110
5.3.2 Likert-scale Preference Survey Results . . . 112
5.4 Discussion and Comparative Analysis . . . 116
5.5 Summary . . . 117

6 Conclusions **119**
6.1 Improvements . . . 120
6.2 Future Work . . . 121

References **123**

A Detailed Component Description **129**
A.1 Core Components . . . 131
A.2 Contextual Menu Components . . . 132
A.3 Main Menu Components . . . 133
A.4 Plugin Components . . . 135

B Notes on Interpolation **137**
B.1 Linear Interpolation . . . 137
B.2 Spline Interpolation . . . 138
B.3 Spherical Spline Interpolation . . . 144

C Whisker-box Plot Reading Guide **147**

List of Figures

2.1 Example of a bystander (right) beckoning the viewer. 16
2.2 Example of synthetic augmentation of a first-person video. 17
2.3 360° video mapped to the inner surface of a sphere. 19
2.4 Comparison between equirectangular and polygon mapping pipelines. 20
2.5 Example of a fixed orbit around a 3D object. 22
2.6 Example of "exocentric" and "egocentric" zooming interaction. 24
2.7 SceneSkim prototype interface. 31
2.8 SketchingWithHands use-case. 32
2.9 HTC Vive and wand-controller usage. 35
2.10 Leap Motion schematic. 36

3.1 Example of traditional annotation process. 40
3.2 Interpolation between keyframes. 41
3.3 Keyframing applied to animate a "blur" property within Adobe Premiere Pro. . . 42
3.4 Coloured spots marking annotation points in a frame of a 360° video. 47
3.5 Annotation video mask. 48
3.6 Example of the early prototype operation. 49
3.7 Example of a textual annotation with a red coloured marker. 50
3.8 System integration diagram. 51
3.9 High-level architecture relationship diagram. 52
3.10 Example of the virtual laser serving as a pointing aid with the HTC Vive controllers. 55
3.11 Example of the virtual laser serving as a pointing aid during Leap Motion usage. 57
3.12 HTC Vive user selecting a distant object. 58
3.13 Leap Motion user selecting an object by pinching. 59
3.14 User dragging an object using the Leap Motion controller. 60
3.15 Low-fidelity wire-frame for the video controls menu. 62
3.16 Low-fidelity wire-frame for the annotation menu extension within the video controls menu. 63
3.17 Low-fidelity wire-frame for the keyframes' contextual radial menu. 64
3.18 High-fidelity wire-frame for the expanded video controls menu. 65
3.19 High-fidelity wire-frame for the keyframes' contextual radial menu. 66
3.20 Example of Menu operation using the HTC Vive controllers. 68
3.21 Example of Menu operation using the Leap Motion controller. 69

4.1 Usage of the implemented solution. 71
4.2 360° video in the context of the proposed solution. 74
4.3 Annotation allowing proper viewing. 75

4.4 Examples of visible and partially occluded objects. 77
4.5 Examples of annotation rendering. 78
4.6 Example of annotation interpolation. 80
4.7 Interpolation on a spherical surface. 81
4.8 Example of Catmull-Rom interpolation with three keyframes. 83
4.9 Example of Catmull-Rom interpolation with two keyframes. 83
4.10 Example of Catmull-Rom interpolation seen from outside the sphere. 85

5.1 Initial scenario for T1. 94
5.2 Annotation to delete during T2. 94
5.3 Example of a new annotation placed during T3. 95
5.4 Expected scenario upon rigorous completion of T4. 95
5.5 Distribution of annotations in the scene. 99
5.6 Keyboard and mouse user gaze heatmap. 100
5.7 Oculus Rift user gaze heatmap. 101
5.8 HTC Vive user gaze heatmap. 101
5.9 HTC Vive and wand-controllers *y* axis dragging during T1. 102
5.10 Desktop screen plus keyboard and mouse *y* axis dragging during T1. 103
5.11 Oculus Rift and Leap Motion *y* axis dragging during T1. 104
5.12 Oculus Rift average error per ten clicks over time. 105
5.13 Mouse average error per ten clicks over time. 106
5.14 HTC Vive average error per ten clicks over time. 107
5.15 Centred right hand horizontal and vertical positions for T4. 108
5.16 Centred left hand horizontal and vertical positions for T4. 108
5.17 Distribution of SUS scores per environment. 111

A.1 Architecture diagram 130

B.1 Linear interpolation. 139
B.2 Set of arbitrary collinear points and one outlier. 139
B.3 Fitting eight-order polynomial. 140
B.4 Natural cubic spline interpolation. 142
B.5 Catmull-Rom interpolation. 143

C.1 Example of a whisker-box plot. 147
C.2 Example of a whisker-box plot with point labels. 148

List of Tables

2.1 List of 360° multimedia and editing systems. 21
2.2 Comparison of 360° multimedia & editing systems. 21
2.3 List of 360° and VR visualisation systems. 26
2.4 Comparison of 360° and VR visualisation systems. 26
2.5 List of HCI, CAD and virtual reality interaction systems. 33
2.6 Comparison of HCI, CAD and virtual reality interaction systems. 33

5.1 Gender distribution. 92
5.2 Age distribution. 92
5.3 Previous video editing experience. 92
5.4 Previous virtual reality experience. 92
5.5 Education levels. 92
5.6 Participant handedness. 92
5.7 Permutation of environment order. 96
5.8 Computer specifications. 97
5.9 Globally gathered data. 98
5.10 Comparative precision scores for selection actions 104
5.11 Possible SUS grades per score 112
5.12 Average SUS scores per environment. 113
5.13 Dunn test results for SUS scores. 114
5.14 Metric score medians and quartiles per environment. 115
5.15 Kruskal-Wallis test results per metric. 115
5.16 Dunn test results per metric. 115

Listings

4.1 Example *.srt* file 86
4.2 Extended *.srt* file 87

Abbreviations and Symbols

2D	Two Dimensions
3D	Three Dimensions
5G	Fifth Generation Mobile Networks
API	Application Programming Interface
AR	Augmented Reality
CAD	Computer-Aided Design
CG	Computer Graphics
CSS	Cascading StyleSheets
EIT	Electrical Impedance Tomography
FOV	Field Of View
GPS	Global Positioning System
HCI	Human-Computer Interaction
HMD	Head-Mounted Display
HTML	HyperText Markup Language
ISAR	International Symposium on Augmented Reality
ISMAR	International Symposium on Mixed and Augmented Reality
ISMR	International Symposium on Mixed Reality
IWAR	International Workshops on Augmented Reality
JSON	JavaScript Object Notation
SUS	System Usability Scale
USB	Universal Serial Bus
VE	Virtual Environment
VR	Virtual Reality
VRML	Virtual Reality Modelling Language
X3D	Extensible Three-Dimensional Graphics
XML	Extensible Markup Language

Chapter 1

Introduction

When one considers multimodal integration in multimedia-rich environments – such as Virtual Reality (VR) – one finds that the classical single-camera approach that allows interlocutors to be framed within the action becomes insufficient. This does not necessarily imply that such pieces require multiple cameras or concurrent points-of-view. Rather, it means the user is immersed, surrounded by content.

In this format, space and time unfold around the users, and not just in front of them. This means that content creators – especially video producers – must look to alternatives for conditioning viewer focus. The apparent viewing freedom users gain from immersive 360° videos in a VR environment can be detrimental to the value of a piece of content if creators, producers and editors fail to guide viewers through the scenario.

A meaningful analogy is perhaps that of Journalism, a field that has refined attention conditioning techniques down to a science. Traditional news content is already augmented with multimedia information – tickers, oracles and even sound cues all aid the user in absorbing the most out of a specific message. For benefiting from a strong sense of presence – transporting viewers to the action – journalistic content is a prime example of the usefulness of applying these traditional bi-dimensional annotation techniques to an immersive 360° scenario. This is not exclusive to Journalism, though. Popular user-driven content-delivery platforms like YouTube and Facebook have been bringing 360° video playback to mobile devices. And with head-mounted display cases for smartphones costing as little as a replacement charger, immersive environments and 360° video content are quickly becoming an unified phenomenon accessible to the masses.

1.1 Context

As one deviates from the specific case study of Journalism and focuses on other areas, especially considering applicability – simulation, training, entertainment – one finds the same concerns. By generalising the problem, the need to conceive new narrative and interaction paradigms around user attention becomes clear, as stated by Zyda (2005). This type of challenge, Feng Zhou et al. (2008) reiterate, is one that has already been studied in other virtual environments that face the

same issues, such as Augmented Reality (AR). During previous work, Carvalheiro et al. (2016) state that these problems will share known solutions precisely because 360° video content, while not being real-time content as in AR, is a representation of reality comparable to that of AR. Similarly, the need to introduce content authoring and editing mechanisms arises. The playback of such content also needs to be coherent, especially in platform-independent contexts, such as the Web. It is important that these features are able to be adapted to the available control and interaction interfaces in VR, since standard controllers – such as mouse and keyboard – may not be available or feasible to use in such a context.

When one approaches realistic content in VR – such as 360° video – with a narrative focus, the ambiguity of some points of interest becomes more noticeable. As mentioned, the apparent freedom of 360° content means that users fail to accurately pinpoint actors or highlighted points in a video piece. To aid the viewers in identifying points of interest and other narrative helpers, it is necessary to augment the video with layers of information that are adapted to the content and the viewer's focus of attention. One way to do this is by use of "dynamic annotations" – multimedia elements superimposed on the existing content. Such paradigms have been studied and documented in-house as well, by Marques (2016). Some of these include billboards, Point-Of-Interest lists, subtitles, tickers, arrows, and a compass or radar. These are not necessarily attached to the video itself, and can accompany user motion, such as head movement. To manage this new type of object motion data, applications resort to metadata stored externally.

This represents a new set of communication and interaction metaphors, hereby defined as "augmentations" or "annotations". The addition of such elements during video editing can also be called "annotation". To avoid confusion, the latter will be referred to as "annotation process", "augmentation process" or "annotating" throughout this document. Furthermore, one should consider the means by which annotations are superimposed on top of video content in real-time. These will be referred to as "annotation rendering" or "augmentation rendering".

1.2 Motivation

After analysing the set of solutions that Augmented Reality can provide – as explored in chapter 2 – one verifies that the process of dynamically annotating virtual environments is one of the most efficient interaction and content-enhancement paradigms implementation-wise. This paradigm allows external multimedia elements to enrich said content with more information, and furthermore allows users to influence their surroundings, by providing additional interaction interfaces.

However, playback of this type of augmented content is not trivial because it hinges on a distorted video to achieve a 360° effect, as Cruz and Fernandes (2011) point out – contrary to Augmented Reality, which works in real-time with monoscopic video image. As such, this type of annotations must be performed in real-time, since only then is it possible to eliminate the image distortion variable.

The distorted nature of 360° imagery – regardless of format – poses an authoring issue that is twofold: first, playback software is required to render the specified annotations on top of 360°

content; secondly, annotation content needs be tailored specifically for such software, which warrants the design of new authoring facilities. Additionally, the fact remains that for the specific case of immersive 360° video content, i.e. content rendered within a VR environment, both playback and editing facilities should incorporate input devices adapted to this medium, whether these are specific controllers or natural interfaces, like human hands.

For instance, natural input devices like the Leap Motion controller[1] or the Microsoft Kinect [2] can be paired with visualisation devices, namely head-mounted displays like the Oculus Rift[3] or the HTC Vive[4] to allow users to interface with the virtual environment, even if a keyboard and mouse setup is not adequate. In the context of this thesis, three sets of input and output devices were studied. Each of these sets will be referred to as an "environment", always applied to the context of the editing procedure. The solution supports the following three environments:

- Desktop screen with keyboard and mouse
- HTC Vive head-mounted display and HTC Vive wand-controllers
- Oculus Rift head-mounted display and Leap Motion controller

1.3 Research Questions

Even two distinct self-contained areas – such as 360° video editing and interactive multimedia content – raise important questions when brought together. For instance, to provide tools for a producer to direct user attention to an action sequence in a 360° video one must start by understanding why viewers are not focusing on said sequence in the first place. Furthermore, video editing staff might not be versed in real-time rendering engines. As such any interaction paradigms that are developed must work as functional black-box components – they must not require additional specialisation and yet they must allow editing staff to add style and multimedia elements to them. This allows the producers to include such paradigms in their workflow most trivially, which would otherwise require extensive research and video editing skills.

As mentioned, significant effort was put into studying and documenting diverse interaction paradigms during the initial preparatory phase of this dissertation. And while some of them were implemented and tested using rudimentary editing techniques, there are still some research questions that need to be refined. For this thesis, these include the following:

Interaction: Can traditional 2D video annotation techniques be translated to immersive Virtual Reality environments? Can producers conceive and create annotated 360° content using such techniques?

[1] Leap Motion input device homepage – https://www.leapmotion.com/. Last accessed: 2016-12-26

[2] Kinect for Windows – https://developer.microsoft.com/en-us/windows/kinect. Last accessed: 2016-12-26

[3] Oculus Rift homepage – https://www.oculus.com/. Last accessed: 2016-12-26

[4] HTC Vive homepage – https://www.vive.com/eu/. Last accessed: 2016-12-26

Usability: Is it feasible to rely on hand gesture detection as the sole input for an editing system? Which devices do users prefer to use in such a system?

Technology: How can the compatibility gap between different systems and software platforms be minimised?

1.4 Objectives

Despite the symmetry between immersive 360° content production and playback, and the set of tools associated with each, it is not trivial to delimit a problem space without regarding the desired approach.

This dissertation focuses on the development and study of interaction paradigms for 360° content editing within Virtual Reality and their prototyping, with the ultimate objective of assessing the adequacy of a set of input and output devices and interaction mechanisms for authoring annotation content, without the need for specific user technical skills.

While there exists the domain-specific problem of conditioning user attention during playback of immersive 360° video, it is not as overreaching as the spectrum of content produced for this medium. Even when one factors in dynamic annotations, there might not necessarily exist a clearly defined set of focus points in produced 360° video content that warrant attention span conditioning. Consider, for instance, a VR experience consisting of an immersive 360° video of natural scenery, such as the Grand Canyon. It is conceivable that in this context there is no action taking place, but annotations and other informational content are still relevant to enhance the piece.

A more broadly applicable approach is then the study of multimodal interfaces, although always within the context of immersive 360° video. The specific target is to reach a conclusion regarding the comparative appropriateness of each environment – i.e. each set of input and output devices – for specific annotating tasks. This study can be performed with recourse to a prototype – an implementation that serves both as a proof of concept for the aforementioned interaction techniques and as a framework for conducting tests on usability. With it, conclusions can be derived from mechanical considerations such as precision and task completion time, and subjective data such as user comfort or perceived naturalness.

As such, the following objectives are defined:

- Reach a conclusive delimitation and documentation of interactive 360° video paradigms, especially dynamic annotations and subtitles;
- Elicit and document interaction metaphors that allow for the intuitive editing of interactive 360° videos;
- Implement a software solution for referred metaphors that supports editing, using Virtual Reality devices and tools;
- Extend the aforementioned software solution to accommodate different types of devices, both classical and VR-compatible.

- Create a test suite capable of assessing user satisfaction and ability to use the implemented solution with each of the referred devices, and answer the aforementioned research questions.

The majority of development time was devoted to developing the proposed solution and ensuring its compatibility with the aforementioned devices. After this, user tests were carried out with the intent of gathering both mechanical and subjective data. The results of such a study are presented in this document.

1.5 Contributions

The contributions of this dissertation start with the research and documentation of interaction techniques, some of which were developed particularly for this solution, or as a consequence of it. These novel techniques come in two forms: practical strategies that describe the functioning of selected paradigms for particular aspects of rendering and information handling, among others; and analytical descriptions that allow previously known techniques to be applied to new scenarios, such as the translation of 2D interaction paradigms and interpolation techniques to a three-dimensional immersive space.

These efforts were integral parts of the development of the previously mentioned software solution to facilitate the annotation process of 360° content, compatible with multiple standard and VR input and output devices. This system presents a significant advance both in interactive video editing and towards streamlining the augmentation process for professional and everyday use.

For instance, the foremost feature of the software is its full VR integration – it allows users to navigate a virtual environment, and work on video editing from it as they would in a regular setup. As mentioned, natural interfaces like hands are key in maintaining the immersion, and the implementation reflects this.

There are three main areas of impact regarding the context of this dissertation: one relating to editing systems in the context of 360° media; one surrounding Human-Computer Interaction (HCI) applied to VR; and another pertaining to the augmented viewing of 360°media. These are presented in the following list, along with the leading contributions in each area.

360° multimedia & editing systems: There is currently a dependency between pre- and post-production of 360° video content, to the point that the process of adding annotations has to be taken into account during the whole production phase. It is common for video editing applications to fail to provide tool-sets that account for distortion of the video file, unless the producer can control the whole pre-production pipeline as well. By researching real-time video editing in VR, significant effort is put into implementing and providing multiple meaningful tools within the same application.

HCI and virtual reality interaction systems: Another development is the implementation of natural interaction paradigms that the producer is able to deliver to the viewers. After gathering and identifying such paradigms in literature, their subsequent implementation facilitates both editorial control – by which the annotation's style is changed – and multimedia content navigation, which will simplify the annotation process.

360° and VR visualisation systems: In an attempt to provide a consistent experience – that is, assuring that both the viewer and the producer have the same outlook on the content – novel display techniques were employed or developed when not available. The main contribution is the addition of gesture-based navigation techniques, which take advantage of the aforementioned natural interfaces to provide an intuitive control of virtual space, as well as make the most out of the user's field of vision and head orientation.

The work presented in this dissertation is both the result of previous research work and the precursor of future application development. While not directly related to the academic scope of this thesis, an exploratory prototype was developed with the single purpose of implementing a 360° video player in Virtual Reality which allowed for the processing of annotation and subtitle metadata. These data would be extraneously managed, and therefore independent of the implementation, providing a framework that could be easily assimilated into existing workflows. The chosen method for generating such data within this exploratory prototype was a computer vision approach using existing traditional video editing tools. The details of this prototype are explicit in section 3.2.

Additionally, a peer-reviewed paper that details this prototype was published:

Meira et al. (2016). Video Annotation for Immersive Journalism Using Masking Techniques. In *2016 23° Encontro Português de Computação Gráfica e Interação (EPCGI)*, pages 1–7. IEEE, Nov 2016.

Furthermore, this dissertation is inserted in the *AV360 - Augmented Video 360*[5] research project, in collaboration with the Institute for Systems and Computer Engineering, Technology and Science (INESC TEC) of Porto, funded by Google's Digital News Initiative (DNI).

1.6 Document Structure

This initial chapter summarily presented the problem of translating already established two-dimensional interaction paradigms to the context of immersive 360° video. More specifically applied to the authoring of annotations and other informational overlays. The remainder of the document aims to deepen the understanding of the problem, and subsequently propose a solution. The details of such a solution require a segmented analysis, semantically hermetic and divided into chapters. Therefore, for the rest of the document the structure is as follows:

[5] AV360: Augmented Video 360 project page – https://dei.fe.up.pt/gig/projects/av360-augmented-video-360/. Last accessed: 2017-22-06

Chapter 2 Related Work

Presents a literature review of state-of-the-art publications. A comparison is drawn between these research projects and equated to this dissertation's objectives. Furthermore, a technological review of the currently available devices is described.

Chapter 3 Architecture of the Immersive 360° Annotation System

Summarises the problem in the form of a requirements analysis. All design work prior to implementation is also detailed, namely with regards to an early prototype. In an attempt to formalise the proposed solution, an overview of the system's structure is provided. Furthermore, the researched and designed interaction techniques are presented.

Chapter 4 System Implementation

Presents the outcome of research – either conducted or developed – in terms of techniques that are meaningful to the system. These are rendering and object manipulation paradigms, as well as interpolation techniques employed in the solution. It further presents the details of information persistence methods.

Chapter 5 Evaluation

Describes the studies conducted with users to assess the system, and presents the results of gathered data processing. Furthermore draws conclusions regarding the statistic frame inferred from the results.

Chapter 6 Conclusions

Elicits practical considerations gathered during the course of this work, and further reports possible improvements and other pointers for future work on this system.

Additionally, appendices A through C contain useful information that will be referenced throughout the document, namely in chapters 4 and 5.

Chapter 2

Related Work

Looking at the concept of 360° video, one realises that playback of such content is not trivial. This comes naturally as human field-of-view is limited – even if one majorates it to 180° it will still only encompass half of the 360° video area. If one takes into account other vision and attention conditioning aspects, such as focus or depth perception, this usable range decreases significantly. It is therefore safe to assume that experiencing 360° content requires a new type of control – an interface with which to pan or rotate the content, but also specially tailored to inform, instruct and guide the viewer.

Conceptually, earlier prototypes achieved this on Web browsers by use of the mouse to display panoramic photos. Much like in a first-person video-game, users would move the mouse to control the viewport's direction. This proved to be a non-immersive experience after a brief period of novelty. Interacting with content through a non-natural interface such as the mouse detracts from the sense of presence.

As such, new approaches focusing on dedicated hardware emerged. Virtual Reality (VR) devices take over the user's vision and sense their head movements through head-mounted displays (HMD). In recent years – with such displays becoming increasingly cheaper, coupled with state of the art mobile sensing and computing power – 360° video has begun to come across as an extremely rich content format which warrants its investment cost. This further reinforces the idea that 360° interactive content and VR are linked in symbiosis.

Still within the device-oriented frame, another type of technology becomes relevant – Augmented Reality (AR) is the set of techniques by which representations of the physical world are enhanced by superimposing virtual objects or data. The most interesting features of AR when dealing with 360° are two – the augmentation of content by means of real-time rendering of virtual objects, and the technology by which user focus is tracked and influenced. Using a hand-held device, such as a smart-phone, to pan around a 360° panorama feels more natural than using the mouse. Furthermore, the touch-screens in most of these devices ensure that this medium continues to support interaction.

That same metaphor, however, is lost in VR. Assuming the viewer is "sealed" from the real world by means of a HMD, interfaces such as keyboards and mouses are either not accessible, or

their usability is limited. Complementary to this is the head-tracking capability of such setups: the user does not need to explicitly pan the video, but simply move their head to look somewhere else, and their virtual perspective will follow.

That said, this chapter is dedicated to a qualitative literature review in the context of 360° video-based content, its virtual enhancement, and rendering, as well as brief argumentation on hardware and software choices regarding this thesis. Firstly, an introduction to Virtual Reality history and core concepts is presented in section 2.1. A section on Augmented Reality is also included in section 2.2, and finally, an analysis on a set of past research works is performed in sections 2.3, 2.4 and 2.5. To conclude, a summary description of available technology is presented in section 2.6.

A brief introduction to VR and AR separately is presented in the next section, before the full scope of this dissertation is de-constructed and analysed separately. However it is not without cause to consider VR and AR simultaneously, since there is a valid relationship between the two as proved by Milgram et al. (1995). Arguably, with regards to currently available technology, reality and virtuality are not antithetic. Rather, they are extrema in a spectrum – a reality-virtuality continuum. While it is possible to regard this continuum as a one-dimensional line, discriminating between exclusively real and virtual environments, other characteristics can be parametrised into a multi-dimensional metric. For instance, reproduction fidelity and extent of presence are also continua and not discrete points as Milgram and Kishino (1994) propose. When analysing VR as a technology, one must not forget how integrated it is with AR from a human perception and interaction standpoint.

The purpose of such a qualitative analysis was to gather information on the state of current research projects, so as to further understand which metrics the industry and academic standards provide, and to possibly elucidate any doubts regarding integration between different components.

2.1 Virtual Reality

One can trivially argue that "virtual reality" is a combination of different technologies, in which virtual environment rendering, interface, and system integration are key. However, the term "virtual reality" was popularised by Jaron Lanier of VPL Research in 1989, and yet, experiments with virtualisation and simulated reality far preceded that date.

2.1.1 Background

In 1929, Edwin Link had completed his Link Trainer [1] – a flight simulator that resorted to air pumps and valves to provide accurate instrument-based training – demonstrating that mechanical virtualisation preceded computer-controlled one. Later, after mid-century, Heilig (1962) unveiled his Sensorama that, while still being a mechanical device, was able to display stereoscopic 3D

[1] Link Simulation and Training history page – https://www.link.com/about/pages/history.aspx. Last accessed: 2016-12-27

imagery through a wide field-of-view (FOV), provided stereo sound, and enabled the user to feel vibration, smells and wind. In the same decade, three years later, Sutherland (1965) presented display concepts that are still in use today. Expanding on Heilig's design, Sutherland was able to theorise the use of multi-sensory displays, tracking devices, immersion and new, intuitive input models. Subsequently, Sutherland and E. (1968) designed the so-considered "first virtual reality system" – called *The Sword of Damocles*, an allusion to the Greek myth of a sword hanging over Damocles' head, held only by a single thread of horse hair. This system featured a head-mounted display and head tracking. Due to its translucent nature, it is also often regarded as a precursor to augmented reality devices.

In his paper, Foley (1979) began theorising about the foundations of computer graphics – application and device layering and independence, definition of primitive attributes, input interfaces, among others. This and other computer graphics articles and journals became increasingly popular in the following decade.

In 1994, the first version of the Virtual Reality Modelling Language (VRML) was specified – for the first time, a Web-oriented solution was available and standardised. It featured an annotation format for describing polygon vertices as well as surface colour, textures, shininess, transparency and other graphical properties. Designed specifically for the Web, it allowed for the creation of virtual worlds with which the user could interact as they would a regular Web page. It also supported animations, sounds, lightning and scripting hooks. In 2004, it was superseded by X3D, a similar XML-based annotation format [2]. In the same year, a book by Burdea and Coiffet (2003) established VR as the integrated trio of immersion, interaction and imagination, arguing that a simulation is only real as long as the human mind can imagine it so. This groundwork proves that core concepts of VR, such as user perception of non-existent or abstract things, were already being researched over two decades ago.

As the 1990s progressed into the 2000s and onward, computer hardware and software began developing at a fast enough pace to allow for new Human-Computer interaction metaphors – first as novelty or research, then as mass-produced consumer products, as is the case of the video-game industry. Starting in 1991 the first arcade systems supporting VR started to emerge. One such system was called Virtuality which, powered by an Amiga 3000, supported networked multiplayer and a novel glove and magnetic ring input system. In 1994 it was rendered obsolete by a more powerful version, the SU2000 – as seen on the technical manual written by Giles and Kerry (1994) – which relied on a Intel processor and had dedicated Motorola graphics processors. On the consumer front, 1995 saw the launch of the Forte VFX1 headgear device. While it still suffered from poor performance and cost nearly as much as a brand new computer, the design is unmistakably a premature version of modern headsets – supporting one screen for each eye, head tracking, built-in stereo sound, microphone – and compatible with video-games such as *DOOM* and *Quake* [3]

[2] Virtual Reality Modelling Language standards page – http://www.web3d.org/standards/all. Last accessed: 2016-12-27

[3] VFX1 Official distributor page – http://www.mindflux.com.au/products/iis/vfx1.html. Last accessed: 2016-12-27

In the early 2000s, PC-based 360° content experiments such as those presented by Román et al. became increasingly common, as Google introduced its Street View technology in 2007, based on early Stanford University research projects starting in 2001 and concluded in 2006. This technology shows panoramic views of landscapes and city blocks, and features a stereoscopic 3D mode as well as an API since 2010 [4]. Also in 2010, the first prototypes of the Oculus Rift surfaced, resulting in the founding of Oculus VR, LCC two years later, with Palmer Luckey at the helm. Shortly after, in the same year, the company garnered over two million dollars through crowdfunding. In 2014, Facebook purchased Oculus VR, LLC for a reported two thousand million dollars. Sony announced Playstation VR in the same year, and Google unveils their accessible do-it-yourself Cardboard project. In 2015 Valve and HTC partnered to create the HTC Vive, an alternative to Oculus Rift.

2.1.2 Defining Virtual Reality

While VR research and development was driven mostly by entertainment and simulation applications, it has come to grow out of its initial domain to now encompass novel research areas, such as engineering, architecture, medicine, as well as defence and education training programmes. This broader spectrum implies that one cannot give a universal definition of "virtual reality", and must instead rely upon qualifying the definition within more detailed parameters. Of the multiple lenses through which one can define "virtual reality", the technological and immersion perspectives offer satisfactory explanations. While technical definitions are more objective with regards to interaction, immersion-based ones focus on user perception of the experience.

2.1.2.1 Technology-related Definition

The schema for a technological definition of Virtual Reality is generally the most widely accepted. One approach under this outline is that of human-computer interaction: Aukstakalnis et al. (1992) defend virtual reality to be defined as a means by which people can see, operate and interact with complex or non-trivial data through computers. Even Sutherland (1965) described "kinaesthetic displays" – devices that serve as many senses as possible, theorizing about head-mounted displays as well. This is to say that "virtual reality" can be realised as a hardware and software symbiosis. Then again, interaction is paramount – utilising devices to control a virtual world, rendered through optics, seems to be the common motif.

Furthermore, with the increasing demand for content, VR has come to also include authoring technologies, such as tracking and rendering APIs and video-game engines that enable the generation and manipulation of virtual worlds in real-time.

[4] Google Street View service page – https://developers.google.com/maps/documentation/javascript/streetview. Last accessed: 2016-12-27

The ability to manipulate intangible objects and the abstraction it suggests when representing knowledge are the core features of this VR definition.

2.1.2.2 Immersion-related Definition

Ever since Hungarian psychologist Csikszentmihalyi (1975) introduced the concept of "flow", immersive media have been harnessing the power of motivation. User engagement can be classified as "immersion" when the user chooses to be completely involved in an activity for its own sake and loses other unrelated perceptions. One will see this concept is similar to the "sense of presence" found in literature, as stated by Zhou and Deng (2009) in their survey. In fact, when considering that a person's understanding of reality is dependant on personal perception, an immersion-based definition could be leveraged by the aforementioned "flow" theory. When experiencing a world constructed in their minds – that is, being immersed in such a world – people will consider that physical and mental space a "virtual reality".

Whether or not this constructed world view reflects the real one becomes irrelevant under this scope. As long as a certain space can be reproduced by means of an arbitrary medium – such as a virtual reality head-mounted display – it can be considered a virtual space, provided the users feel immersed in it.

Curiously, this type of definition complements the technological one, as the latter can be seen as a digital, tangible implementation of this theoretical definition.

2.2 Augmented Reality

Augmented reality (AR) is the set of techniques by which virtual data is layered on top of captured physical data, such as video – constrained, of course, by technological challenges. Nóbrega (2013) states that augmented reality systems face two problems: registration, when it is necessary to acquire the location where the virtual information will be placed, by means of GPS and compass usage, or physical markers; and superimposition, by which virtual information is rendered in 2D or 3D using pose estimation on the correct spot. This real-world aspect is what allows AR to provide the solution for many VR problems when the latter's content tend to realism. As mentioned, Sutherland and E. (1968)'s *The Sword of Damocles* is often regarded as the precursor to AR for its innovation. In the subsequent decades, technology allowed pioneering applications to emerge in areas such as medicine and mechanical engineering, even if such applications were not yet fit to be usable due to inconsistent or low-quality visualisation systems. It was not until AR became a more independent research field in the 1990s that these issues started to be approached systematically. Bowman (1999) would argue that many visualisation issues arise from the lack of standardised interaction and visualisation techniques. Furthermore, Bowman suggests that VR formal frameworks can borrow from existing computer graphics research, as this taxonomical isolation is far more common in those areas – the benefit of a modular approach is that it reduces the space of the problem.

Like VR systems, AR systems utilise position and orientation tracking at their core, although the latter typically resorts to GPS positioning and gyroscope to also provide geographical data. This overflow of data has been the subject matter of many recent research articles, from which sprouted solutions such as the "bubble metaphor" employed by Takeuchi and Perlin (2012) or dynamic label positioning introduced by Grasset et al. (2012) which take advantage of geographical data. Liu et al. (2008) propose attention analysis algorithms which focus on positional tracking as well.

According to Feng Zhou et al. (2008), tracking and interaction papers are both the most published and the most cited among ISMAR, ISMR, ISAR and IWAR conferences, with display techniques following close behind. It is arguable that this tendency towards technology research is beneficial to VR research as well, seeing how similar they are at a technical level.

It would then be trivial to see how AR metaphors can be applied to VR, especially (but not solely) when content tends to realism. As the line between the two areas dims, one will witness more and more paradigms being shared between the two, such as is the case of gaze-directed steering and viewport distortion presented by Carvalheiro et al. (2016).

2.3 360° Media and VR Rendering

When analysing VR projects one should first define the problem's domain. This is because often times research will tend to overly technical subjects, and the focus on the problem at hand will be lost. It is sometimes better to expand the criteria to include AR articles. For this specific case, novel hardware is not a necessity. In fact, targeting currently-available consumer hardware yields access to a much larger code base, providing the support and tools required to perform more in-depth tests. Furthermore, as it will be shown in this section, this approach allows for the application of AR techniques to VR environments, without loss of generality.

2.3.1 360° Video, Multimedia Editing and 3D Modelling

Both VR and AR devices have the ability to be pivoted around a point, either by using head movement or dragging the device through the air. This, coupled with recent advances in technology, potentiates panoramic content, such as 360° video.

This type of content is not trivial to produce and distribute. As will be discussed in this chapter, the process of producing 360° content is split into two phases. Firstly, raw footage must be captured – either resorting to specific camera lenses, multiple-camera setups, or a combination of the two. This footage must then be corrected to compensate for colour and overlapping issues. The process of mapping multiple-camera footage to a single panorama is known as stitching. Typically, camera manufacturers will provide proprietary stitching software, but third-party solutions also exist.

One of the main areas of focus when studying 360° video is the problem of controlling the users' attention. When inside a virtual environment, the freedom to look around impacts the film

makers' ability to direct the viewer's attention – a serious impediment to telling a story. Below is a list summarising the main challenges and desired features of this type of environments.

- 360° video stitching
- 360° video playback
- Pre-production authoring
- Post-production authoring
- Dynamic post-production content
- Video scrubbing
- User gaze directing
- Gaze interaction

360° video enables a stronger sense of presence by correlating closely to real life, in the sense that the user's view is not constrained *a priori*. This also means that the director is forced to work with a diminished domain of techniques – controlling camera angles and transitions have reduced effect in 360° environments. More so, rather than guiding the user through the narrative path, applying such techniques in a VR environment can wrest control away from the viewer, and even induce discomfort. Brown et al. (2016) defend that since the benefits of 360° are a function of the viewer's control over their own gaze, directors and film makers need to employ more subtle, unobtrusive techniques in order to allow the viewer to retain agency. However, they suggest that these techniques should rely on movement, sound and lighting – all of which are pre-production static assets that are directly recorded into the video.

On a similar note, Akkil and Isokoski (2016) present a system to augment regular first-person video using gaze data. The system captures the user's gaze and later superimposes it onto the video. This approach proved to enable the users to perceive intention and deictic gestures better.

Both of these studies focus on different approaches to the same problem: conveying a message to the viewer when the medium allows for more freedom. While the former features the effects of in-narrative characters guiding the user's gaze directly, the latter relies on synthetic artefacts at a post-production level.

The first study has the additional objective of assessing whether the sense of presence, as Brown et al. (2016) suggests that immersion and enjoyment of an experience could be manipulated by the actors in the scene. It was shown that, by introducing different cues to both challenge and direct the viewer's attention, pre-production techniques can be very effectively not only in augmenting the experience but also as a tool to control the viewer's attention. Remarkably, audio cues have proved to be the most effective considering they are agnostic of the camera's position and orientation. Coincidentally, these audio cues can be introduced at a post-production level, which makes them a valuable asset for VR or AR content production. Another remark regarding

the feeling of immersion is the distance from the action. This immersive quality is a property of VR content. It was demonstrated that viewers find the optimal distance away from the action is 3 meters. It was reported that 4 meters was too distant, as if the scene took place across the street, and 2 meters was too close, almost in the viewer's personal space. However, perhaps the most novel is the study on presence – when presented with a clip in which the actors actively engage the viewer during their dialogue, users will experience significantly superior immersion. This is also the case when passers-by "looked" at the viewer.

While it is clear that using both visual and audio cues to direct user attention is effective, these techniques rely on the assumption that the editor has full control over the authoring process. With the exception of sound, none of the techniques – such as the addition of extras – are plausible during post-production, as is the case of Figure 2.1.



Figure 2.1: Example of a bystander (right) beckoning the viewer. Picture from an article by Brown et al. (2016).

The second study introduces the concept of synthesised annotations in the form of a red dot over a focus point. Akkil and Isokoski (2016) consider this type of annotation an enhancement as it is superimposed on the regular video either at post-production or at run-time. While this system was designed for regular video it has wide application in 360° video in order to create awareness – both spatial and contextual. The augmentation of video content has a twofold effect on user experience: for one, it adds extrinsic data to an already multimedia format; secondly it subtly influences the viewer's gazing behaviour. This gaze synchronising ability has proved to be a collaborative tool as it allows participants to jointly focus on the same targets. In VR, especially within realistic authored content, gaze control helps the directors ensure a consistent experience for viewers. Naturally – and it remains one of the core semantic definitions of the word – augmentation is not meant to be used in place of other practical modalities for user attention redirection. Instead, augmentation functions as a complement for other communication channels. One such example is that of Figure 2.2.

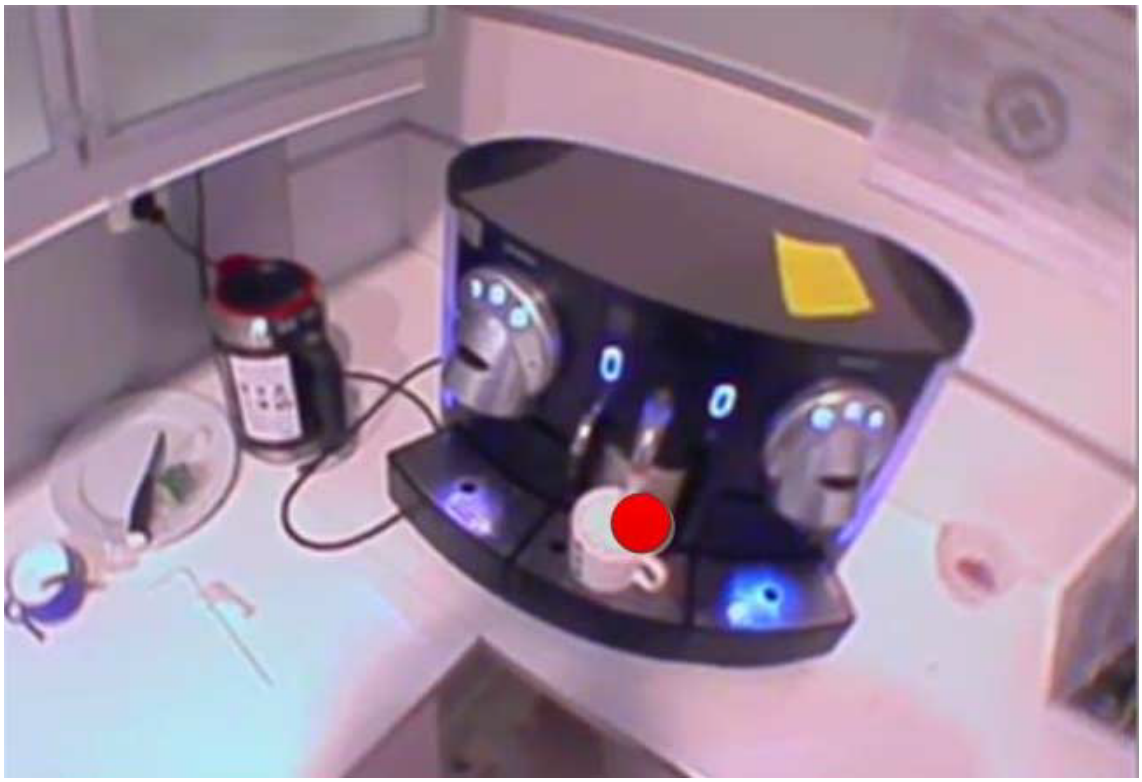


Figure 2.2: Example of synthetic augmentation of a first-person video. Picture from an article by Akkil and Isokoski (2016).

What both these two articles attempt to solve can be reduced to a workflow problem. As Henrikson et al. (2016) argue, planning and executing narratives for VR is a grand challenge – not only do these environments have unique conceptual nuances, such as 360° action, but have also come to incorporate and create new cinematic vocabulary. Most important, however, is the reported lack of tools that support exploratory creative processes in VR. When authoring VR content, one is required to take into account concepts such as presence and peripheral vision, as well as their effective use. To further reinforce the already established points, it is stated that traditional cinematic principles of cuing and staging – although efficient – need to evolve to accompany VR technique standards. It is clear that a workflow solution is as needed as are novel interaction techniques. Furthermore, other variables that are typically taken for granted in regular video production, such as duration and viewer fatigue, are much more limited in VR. The constraints placed in this type of content production mean that directors need to make decisions and technical plans in VR, and not just around it. After all, content production is an industry, and VR-obsolete planning techniques such as storyboarding can quickly become costly for large teams. This is true even for established workflows – the reason being that whatever the current workflow is, it relies on a multitude of professional tools, rather than a single focused and dedicated tool – as such professional tools are not available. The authors propose a sketching system that allows users to draw and experience storyboarding and previewing scenes in VR. One flaw of the proposed system is the lack of an ability to insert 3D models and properly caption scenes without handwriting. This is an issue that

could be solved with multimedia augmentation.

Video augmentation is a research area that serves multiple collaborative networking solutions. The ability to takes notes on top video confers them a space-time quality that greatly enhances the medium. Fakourfar et al. (2016) have developed a prototype for mobile remote assistance that allows users to explain their reasoning by drawing on-screen. In essence it is an augmented reality application, so theoretically it can be almost directly translated to virtual reality. The interesting scope, however, is specifically that of annotations. The study found that stabilising these augmentations provided negligible benefit over those displayed atop live video – once the annotation has been understood, it is no longer needed. Furthermore, with hand-held devices come human induced variables, such as screen shake and distance from target which can make the dynamic annotation process difficult. This is not the case in VR as user perception is controlled, and whichever jitter may come from camera operation is embedded into the video and is therefore deterministic. As such, while different tasks call for different annotation interaction metaphors, trusting the user to use multimedia commands – such as pause and play – while manoeuvring in virtual space will surely mitigate these effects.

Regarding compatibility, storage and delivery, Fautier (2016) presents an analysis of VR video content creation and the technical options for VR display and broadcasting. Arguably – and indeed so stated by the author – VR content creation is still a work-in-progress as far as systematic processes go. While capture is largely dependent on state-of-the art hardware, the software domain has been amassing new techniques. One such example is that of mapping the 2D video texture to a 3D geometric surface, so that the 360° content is available in virtual environments. The trivial approach is to use the inner surface of a sphere. However new schemes have arisen, namely capitalising on the removal of details that are outside the line of sight. An example of this approach can be seen on figure 2.3, which was generated in Unity in the context of this thesis.

The author's focus shift to the encoding domain – both on-line and off-line. Disregarding on-line scenarios, the most straightforward approach is that of equirectangular encoding (seen on Figure 2.4, top row). It consists solely of the raw 360° footage, distorted to fit a 2D rectangle. The main disadvantage of this approach is that only 8% of the total video frame is in the user's field of view at all times.

A similar, but higher-quality approach is that of equirectangular tiles. The raw footage is captured at at least four times the regular resolution, split into four quadrants, each divided into separately encoded tiles. This allows the system to serve whichever tiles are inside the user's field of view, resulting in a much higher resolution experience, at the cost of latency and bandwidth efficiency. Naturally, this system requires a more potent machine (and a capable server, in the on-line scenario).

On-demand transcoding is an alternative that occupies reduced bandwidth by serving only the region of the video within the user's line of sight. This means that these systems require powerful real-time computing features, as well as fast transmission speeds. When compared with an equirectangular solution this type of transcoding will send up to 92% less data, but latency

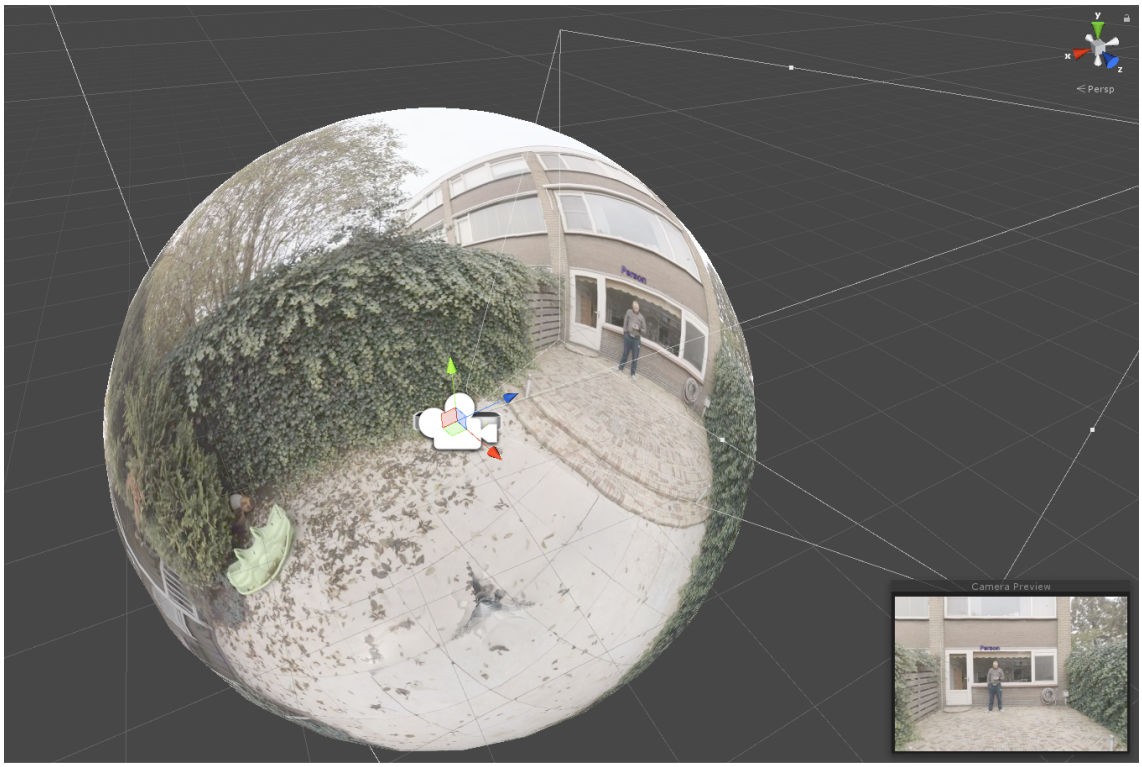


Figure 2.3: 360° video mapped to the inner surface of a sphere. In it, the viewer is placed at the centre, taking the role of the 3D camera.

will rise five-fold, thus drastically reducing the immersion aspect and the illusion of a seamless experience. When designing such a system scalability becomes an issue as the server must tailor its responses to the client's regions of interest. Realistically, this system requires as many transcoders as there are connections, making it economically feasible only for a reduced amount of sessions.

The last option is called Polygon mapping, and it was announced by Facebook in 2016 (seen on Figure 2.4, bottom row). this technique consists of mapping the 2D rectangular video to either a cube or a pyramid. The idea is that since 2D stitched video maps optimally to a sphere, there are areas in the 2D rectangle where information is lost and pixels are stretched. These areas can be compressed when mapped to the net of cubes or pyramids. The latter reduces the file size by 80% with virtually no artefacts, since pixels closer to the viewer's eyes are either clipped by the renderer or compressed by the lenses of the head-mounted-display. While Facebook recommends that servers be able to provide variable-bit rate content for multiple geometries, the author argues that it is still too early to assess the cost-benefit of a lower bitrate, when the heavy real-time computing load and concurrent user requests are considered.

The author theorises that the future of VR video is deeply connected with on-going network investigation, namely regarding new technologies such as 5G. As latency is a core concept in VR, it is trivial to assume that cooperative solutions would rely on the fastest networks. As such, in the near future more over-the-network approaches will become testable in large-scale, allowing for more precise metrics.

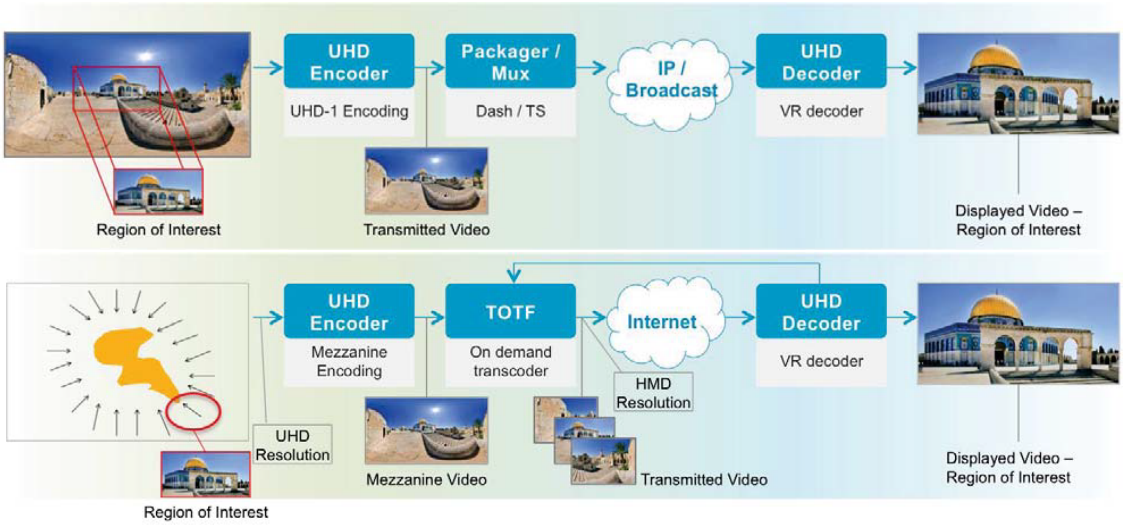


Figure 2.4: Comparison between equirectangular and polygon mapping pipelines. Equirectangular mapping(on the top) is a simpler approach, while polygon mapping(on the botton) is more computationally demanding. Picture from an article by Fautier (2016).

As mentioned, the current 360° video player prototypes either fail to account for vertical distortion or do not provide enough resolution. This may happen because the 360° footage in that particular case is mapped to a cylinder and not a sphere, thus appearing compressed. In fact, obtaining a consistent mapping of a recorded 360° video to a viewable structure in 3D space is not trivial. For a high resolution experience, multiple cameras are used to capture the panorama, and footage from each is projected onto the inside surface 3D and rendered together to an external file – a process known as stitching.

According to Lee et al. (2016) this is still an imperfect process, as the parallax between the capture devices can introduce artefacts. Furthermore, the final rendered piece's resolution is heavily degraded due to the fact that footage from individual devices overlaps greatly. The authors argue that when one has control of both the stitching and rendering processes, it is possible to devise a system capable of greatly reducing artefacts and of increasing level-of-detail by combining editing and rendering techniques.

For the system to work, two novel subsystems are required: a custom stitching protocol that employs computer vision algorithms to reduce parallax artefacts and infer on scene depth, creating a deformed sphere onto which the images are projected and rendered to a separate variable-resolution texture; and a custom rendering software that is able to render this texture to the inside of a UV mapped sphere for viewing purposes.

While this system provides superior stitching and video quality, it necessitates the original raw footage of each camera and requires a proprietary player. This process heavily modifies the workflow. As such, it is not suitable for professional applications where more editorial flexibility is employed. For example, while this system is ideal for fully authored experiences – where

editors control both pre-production and deployment – it may fail to deliver in a heavily pipelined workflow, such as when the editors only control post-production aspects of already-rendered video.

After analysing the aforementioned articles, a feature comparison table was developed and is presented below. An ID is assigned to each exposed research work, so as to identify it in the feature comparison table.

For 360° video, multimedia editing and modelling, the following works were compared based on their ability to play and navigate panoramic content, as well as direct user gaze.
The selected research works are presented in table 2.1.

Table 2.1: List of 360° multimedia and editing systems.

ID	Title	Author(s)
A1	Directing attention in 360-degree video	Brown et al. (2016)
A2	Gaze Augmentation in Egocentric Video Improves Awareness of Intention	Akkil and Isokoski (2016)
A3	Multi-Device Storyboards for Cinematic Narratives in VR	Henrikson et al. (2016)
A4	Stabilized Annotations for Mobile Remote Assistance	Fakourfar et al. (2016)
A5	Rich360: optimized spherical representation from structured panoramic camera arrays	Lee et al. (2016)

In table 2.2, the feature comparison check list is presented. One can see how even though different works share a common topic, the array of features is quite broad. It is safe to assume that this is the case because current solutions are either very specifically tailored for particular functions, or have a purely algorithmic use case without any user interaction. As is demonstrable, this thesis' proposal is one that encompasses multiple application-level needs as well as internal functionality concerns.

Table 2.2: Comparison of 360° multimedia & editing systems.

	A1	A2	A3	A4	A5	This dissertation
360° video stitching	.	.	.	.	✓	.
360° video playback	✓	✓	✓	.	✓	✓
Pre-production authoring	✓	.	✓	.	.	✓
Post-production authoring	.	✓	.	✓	.	✓
Dynamic post-production content	.	✓	.	✓	.	✓
Video scrubbing	.	.	✓	.	✓	✓
Directs user gaze	✓	✓	.	✓	.	✓
Gaze interaction	.	✓	.	✓	.	✓

2.3.2 Virtual Environments and Display

Display and navigation in 3D virtual environments is a field deeply seeded in research. Even when not in VR scenarios, there is a compelling aspect of 3D visualisation, which is that of rendering

to 2D planes. Furthermore, one cannot navigate 2D space with only a 2D plane as a controller, unless multiple functions are available or one degree of freedom is locked. The list explicit below displays the most common features that users and developers alike show concern about.

- Restricted camera movement
- Control of view orientation
- Specific gesture models
- Point-of-view altering commands
- Real-to-virtual object mapping
- Latency compensation

Typically, the aforementioned degree of freedom would be one of the axes, but with advanced translation and rotation, one can parametrise these axes to relative ones, where locking one coordinate makes much more sense. Ortega et al. (2015) present an algorithm to map camera movement around on object by computing an orbit in 3D space, as seen on Figure 2.5. With a focus on user experience, such a prototype maps user 2D touch-screen interaction space to a 3D camera path. When one considers observational tasks of 3D models orbiting becomes one of the most intuitive systems as it combines translation and rotation into a transform that even novices understand. Navigation then becomes a matter of controlling camera position and orientation by means of such a parametrised transform. Arguably, this type of technology is not without application in Virtual Reality – 3D object manipulation tools are important assets, even more so when the objects have restricted freedom (e.g. are attached to the surface of other objects) or when the camera is on a collision course with nearby objects.

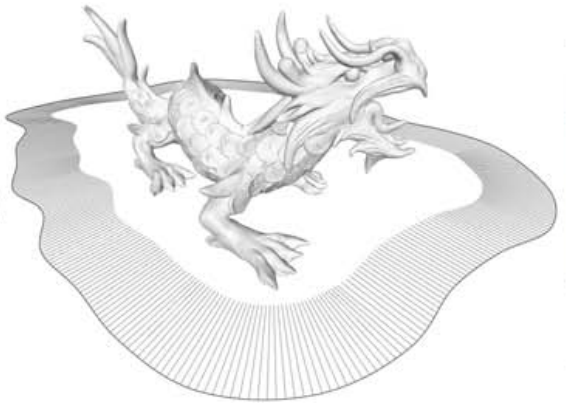


Figure 2.5: Example of a fixed orbit around a 3D object. One can see how this orbit can be rotated to provide different viewpoints. Picture from an article by Ortega et al. (2015).

When manipulating virtual object with one's hands, it becomes necessary to account for inevitable latency. Nancel et al. (2016) have presented a list of metrics capable of assessing user

perception of spacial error when interacting with virtual objects. While this study focuses on touch-screens , its applicability is not lost in VR, as even 3D objects will occasionally be locked to virtual surfaces – thus trading one degree of freedom for a more intuitive, touch-like interface. When capturing user input in the form of gestures, end-to-end latency is the total time between the instant the user makes their intent clear and the moment the system is able to provide feedback. Naturally, in a VR environment with air gestures this feedback is purely visual. Furthermore, the ability for views to distort and condition user perception is a mitigating factor of latency in this scenario. All the while in touch-screens it is common to introduce pointing prediction techniques. However, these can produce visual side effects noticed by the users under non-ideal scenarios. For instance, if the prediction algorithm is poorly weighted or has little influence, the feedback will appear late, as if latency was still present. On the other hand, if prediction is too invasive it will seem to be ahead of the user, potentially conditioning their behaviour. Other artefacts include jitter, jumps, rubber-banding effects and orientation errors.

Perhaps unsurprisingly, users find that prediction lateness (akin to latency) is the less disturbing of these effects. One possible explanation is the association of a physical mass to said objects, which would justify some inertia. This doesn't mean that this delay is preferable to no latency whatsoever. Prediction metrics are still relevant, even if the current state of the art has as noticeable side effects as the unpredicted alternative. In fact, one could estimate which predictor paradigm users would notice more for each scenario, and try to complement it.

On a wider spectrum, namely that of usability and how it impacts application design (rather than interface or interaction), Alessandro et al. (2010) present new field-of-vision and camera views for mobile AR applications. The authors gather that while smartphone hardware has allowed the AR market to flourish, these devices pose serious limitations to usability. Such examples include the reduced field-of-view of mobile cameras (typically 60°) and the restriction to first-person views, while content is spread in 360° around a user. The main issue raised by these limitations is that of overview – with limited screen space, separating focus and context is the ideal approach to allow users more insight. The authors propose two types of zooming interfaces. One of them is called "egocentric zooming", that enlarges the field of view up to 360° and the other is called "exocentric zooming", that transitions the user to a third-person view over all the information. Both of these are triggered in real-time as can be seen on Figure 2.6, without the need for additional hardware or software.

These techniques take advantage of the fact that when browsing AR content, users will typically search for points of interest and, willingly or not, will build a mental map of their relative position, as well as infer spatial relationships between them. For this specific application, panoramic imagery is generated in real-time from already captured footage and stitched using feature tracking techniques. It also features a wireframe grid to support the understanding of distortion. Some usability concerns were stated, namely regarding the display: the system is very sensitive to external lighting conditions, and the real-time stitching is confusing when pitched against a black

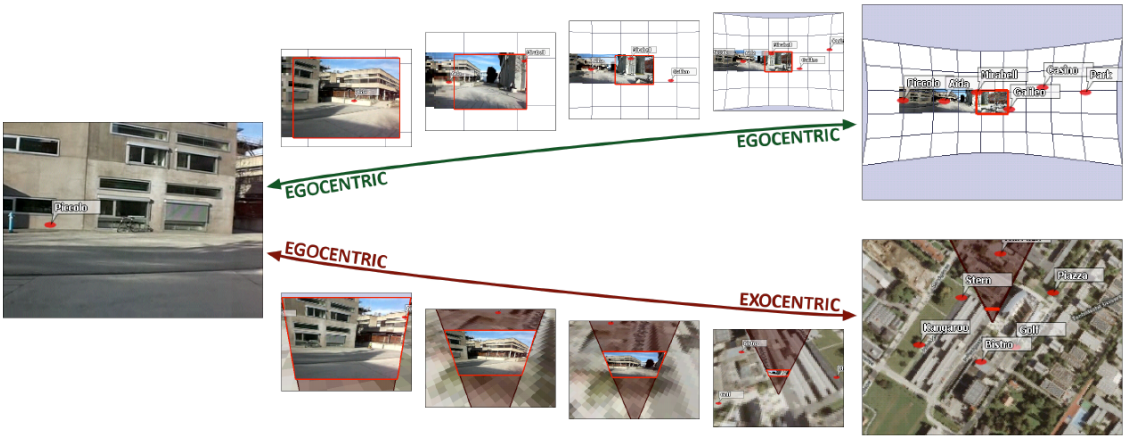


Figure 2.6: Example of "exocentric" and "egocentric" zooming interaction. Picture from an article by Alessandro et al. (2010).

background with incomplete information. Additionally, this technique fails to provide occlusion and distance information. Screen real-estate is a concern, however, as the addition of even more information will induce confusion.

Parallel to that, map imagery is retrieved from satellite data and augmented with text labels for each point of interest. The user's original field-of-view is also represented as a highlighted glyph. This technique is able to convey distance information in the form of a real map, but the authors reveal that users found the transitions to not be smooth enough, causing disorientation. This makes it difficult to use for users who have problems reading real maps. A potential addition to this system is the addition of distance information to each point of interest.

The authors found that different combinations of interfaces have negligible effect on performance when compared to each other. Naturally, these techniques allow users to perform otherwise impossible tasks, so they are always a worthy addition to any navigation system. The authors reiterate that this study was focused on countering the effects of diminished field-of-view and first-person perspective. As such, they prepared a list of limitations regarding their findings: the survey was unable to provide data regarding the user's creation of spatial knowledge, and did not support dynamic positions, requiring the users to be stationary.

Orts-Escolano et al. (2016) present a tele-presence system in mixed reality. The system is capable of capturing and displaying high-quality, real-time 3D reconstructions of an entire scene, including people and objects, and transmit it to remote users. Users can interact with this virtual environment using virtual or augmented reality devices as if they were present in the same physical space. While this publication is very technical in terms of system descriptions, it pertains interesting remarks towards VR applications. These types of tele-presence systems can easily find applicability within immersive VR experiences, whether they are single or multi-player spaces. As the user's body and motions are captured and reconstructed in a shared 3D scene, the sense of presence emerges, especially in future "social" AR and VR applications.

Regarding the user experience, the researchers found consistent reactions across the survey

participants – the users quickly adapted to the shared setups and developed novel communication strategies. Those relied mostly on the intangibility of other users – the hologram-like nature of remote users and objects allows users to intersect other realities. This is also the reason why frames of reference are immediate, contrasting with regular video calls (where displaying an object requires aligning it to the camera and rotating it).

The authors also went to the length of testing multiple device setups, namely both an AR and a VR experience. Users enjoyed the AR experience more as it allowed them to be more in touch with their own physical space, whereas when in VR the users felt they were being transported to a common, disconnected space. Furthermore, in VR one can see a replica of both one's physical reality as well as the remote users', which results in confusing real and rendered objects. the authors state that this can be avoided by filtering or other rendering effects. In a VR environment there was also the issue of latency due to the established pipeline: the user's body needed to be captured and then rendered to the HMD. This caused discomfort for some viewers, although the majority tolerated the latency when picking up objects.

The researchers reported a success in terms of perceived visual quality, with the exception of flickering or distortion artefacts caused when people were at the edge of the reconstruction volume. Additionally, a headset removal feature was proposed – by using wireless cameras attached to the headset, the system could extract textures for the otherwise occluded areas. This technique also detects user features and maps the texture to the 3D model accordingly. This specific setup has a one second start-up time, but is otherwise immediate when used in real-time.

The authors reveal that the main drawback of this type of system is the very heavy computational resource needs, namely two dedicated computers with high-end graphics cards and a 10 Gigabit Ethernet connection for each user, excluding all the state-of-the art capture equipment. The network requirement specifically is one that could be minimised with proper optimization, even to the point of working within the average household's connection speed range. Additionally, reconstruction of interaction tasks is still faulty, especially when fine-grained non-rigid objects – such as fingers – are involved. Nevertheless, they researchers believe that this technology has many yet-undiscovered applications, one such being the ability to record "living memories" and play them back later. This could potentially be useful in 360° immersive scenarios for the recording of actions or displaying tutorials.

Likewise, a feature comparison table was elaborated for the set of research works focuses on virtual environments and visualisation presented above. The exposed literature encompasses substantially diverse domains, and there might not be a direct connection between some of them. Nevertheless, the list presented in 2.3 is comprised of elements which bear some resembling characteristic to this thesis's proposition.

As one can see from table 2.4, while features are by themselves rather isolated, each piece of literature is able to gather more than one under its wing. The domains of such features range from highly technical aspects such as latency to usability concerns such as simpler camera movements by restriction of certain transformation axes.

Table 2.3: List of 360° and VR visualisation systems.

Number	Title	Author
B1	SHOCam: A 3D Orbiting Algorithm	Ortega et al. (2015)
B2	Zooming interfaces for augmented reality browsers	Alessandro et al. (2010)
B3	Holoportation: Virtual 3D Teleportation in Real-time	Orts-Escolano et al. (2016)
B4	Next-Point Prediction Metrics for Perceived Spatial Errors	Nancel et al. (2016)

Table 2.4: Comparison of 360° and VR visualisation systems.

	B1	B2	B3	B4	This dissertation
Restricted camera movement	✓	.	.	.	✓
Control of view orientation	✓	.	.	.	✓
Specific gesture models	.	✓	.	✓	✓
Point-of-view altering commands	.	✓	.	.	✓
Real-to-virtual object mapping	.	.	✓	✓	✓
Latency compensation	.	.	✓	✓	✓

2.4 Acquisition and Interaction Techniques

When approaching user-centric multimedia systems interaction paradigms and a clear definition of interfaces are needed to provide the user with the feedback needed to understand the system. Regarding media production, for instance, Krichenbauer et al. state that AR and VR related publications can be loosely divided into three categories: reviews of attempts to create user interfaces, existing authoring tools for engineering and CAD, and professional 3D modelling or CG software. With each category, different requirements are identified – engineering and CAD are focused on physical correctness and require more precision, while professional computer graphics value artistic expression – even if the dedication of the scientific community to each is not symmetric.

On the consumer front, in the video-game industry, Valente et al. (2016) categorise the current market structure into augmented reality games, pervasive games and alternate reality games, depending on real world interaction. However, a line is drawn between these types of projects and virtual reality games. While AR projects focus on the augmentation of virtual worlds with context-awareness, VR stands out by supporting implicit interaction through controllers or other novel input devices.

2.4.1 Hand Recognition

When researching hand and gesture recognition, one will find pieces pertaining to the most novel of detection techniques. Two are presented below for their prominence and ingenuity. Zhang et al. (2016) present a survey on the use of Electrical Impedance Tomography (EIT) for non-invasive, high accuracy detection of hand gestures. The researchers achieve this by performing a tomography of the users wrist, using an electrode band that is connected to a sensing board, and is worn like a wristband. This device send samples to a computer, which is then able to reconstruct the image in under 3ms, making it suitable for real-time applications. A machine learning algorithm then infers on the hand gesture by analysis of this reconstructed frame.

The researchers report a mean accuracy of over 94%, suggesting that this technique is on par with validated bio-sensing methods such as bioacoustics and electromyography. In fact, this type of sensing, while novel, only works for discrete gestures, and requires a machine learning algorithm to be trained a priori. Furthermore it is incapable of provide hand motion data, as is the case of full-arm motion.

Still regarding hand detection, Chan et al. (2015) introduce a system consisting of a ring-style fisheye wearable camera, that is worn on the hand webbing between the index and middle fingers. This vantage point, coupled with a wide angle lens, allows the device to capture not only the hand's fingers but also the environmental contexts. The system works by first capturing an image and cropping it to include only relevant data. This cropped image is then processed and the identification of skin colour takes place. Binary mask images are generated from this data and then are fed into a decision making algorithm for either training or testing.

The researchers report an accuracy of over 84% when recognising gestures. This result is acceptable considering the system is only using image and colour data. It is the authors' belief that depth data could vastly improve natural hand gesture detection, even more so if a second device is employed in the same hand.

Significant work was also developed regarding different types of interactions. One such is called "on-finger pinch-and-slide" and it allows users to transform their fingers into functional sliders by dragging their thumb across their length. Another feature is called "palm-writing" and takes advantage of the fact that the camera can capture the palm of the hand to allow users to write on their palms using either their other index finger or an object – as they would use a pen. On last relevant technique is called "in-air pinch-and-motion", and it enables the use of pinch gestures for dragging in mid-air. Once the system detects a pinch gesture, it starts estimating hand motion from displacements within the feature tracking of the surrounding environment. This technique is possible only because motion vectors of distorted geometry are eliminated, and only the mean is considered.

The authors also present several AR interaction techniques. One such is the interface with smart objects via gestures. The device camera is able to pick up AR tags, such as QR codes, and trigger commands according to the user's gestures. The authors exemplify this by use of a smart lamp that one can turn on and off by pointing at the tag and pinching. Other theorised applications include human facial recognition, taking pictures, copying or scanning images, interaction with smart watches and controlling of other devices.

It is worth mentioning that none of these devices have position tracking that would allow for a first-person rendering solution with depth.

From a consumer perspective, whether it is a professional application or entertainment, hand detection is regarded as a low-cost interaction method with sufficient precision. A technical review of hand detection products is presented in section 2.6.1. Even though precision is more than acceptable when dealing with the Leap Motion sensor, hand pose and gesture detection are highly contextual subjects that may be hindered by external factors. These might include lighting, sensor orientation, distance from the sensor, among others. When using the sensor in a stationary position, proactive measures can be taken to improve long-term usage performance. This is especially useful when one considers that different hand poses can be estimated with acceptable confidence by simply adjusting the viewpoint of the sensor. Thus, not requiring additional sensing hardware or complicated setups is one of the main advantages of the solution proposed by Hsiao et al. (2016) which uses a low-cost rapid swing arm. Coupled with a machine learning algorithm, the system is capable of adjusting to the most adequate position on a per-user, per-gesture basis. The novelty of such a system resides in its capacity to collect data during usage and effectively complement its off-line training with on-line performance to increase detection confidence. Active positioning of the sensor may not be all there is to it – researchers show that simply shifting the sensor to a random position was not enough, as meaningful motion is necessary to avoid occlusion at all times,

if possible. Since hands are consistent in their structure and movement range, a specific prediction model could theoretically be elaborated to explicitly assist detection. In fact, when only specific gestures are relevant and no fine motion is required one could adapt the detection framework to only respond to specific features, thus implementing a simple detection model by subtraction.

With regards to more focused approach, specifically related to hand gestures and interaction, Farhadi-Niaki et al. (2012) have conducted a user study on the applicability of using hand gestures to control a computer. The authors hypothesise that using gesture-based input is superior to traditional controls (such as keyboard and mouse) when performing typical desktop tasks on a big screen. The performed tasks were increasing in complexity as user satisfaction was assessed, gathering data and participants' impressions of precision, efficiency, ease-of-use, pleasantness, fatigue and naturalness. In a related article, Farhadi-Niaki et al. (2013) compare the use of hand and finger gestures using similar techniques. Since the assessed parameters are the same, an identical research methodology also applies. Users were first exposed to a free-form training session in which they could experiment with the controls. Then, a test session took place, requiring users to perform a set of tasks under different environments. Finally, participants were surveyed using a questionnaire, and data collected by the system was gathered by the examiners. To draw conclusions, a three-way repeated analysis of variances (ANOVA) was performed. The data shows that finger and arm gestures are very similar in terms of user satisfaction, but the authors believe that finger gestures are superior in the long run because they produce less fatigue and feel more natural. When compared to each other, arm and finger gestures do not present significant differences in terms of time easiness, even if finger-based gestures exhibit marginally superior precision. When compared to mouse and keyboard operation natural interfaces like arm and finger gestures lose out on precision and ease of use, while neither of the three exhibit a strong user preference. When inquired, participants reported that the ideal input method would be a combination of arm and finger gestures, with predominant use of fingers for fine tasks.

2.4.2 HCI and Interaction Interfaces in CAD and Virtual Environments

When researching Human-Computer Interaction in virtual environments, one will find a vast array of requirements. Naturally, the terminology will converge to that of virtual and mixed reality – concepts like overlays and radar or compass have increased semantic value in 3D environments. A list of elicited features for this chapter is included below.

- Navigable overlays
- Search feature
- Adjustable interface
- Contextual search
- Natural interaction (hands)

The application of augmentations to video is not new, so it is only natural that applicability to 360° videos has been studied as well. Neng and Chambel (2010) have presented a Web-based hypervideo navigator. While not compatible with head-tracking or VR, the project provides some insight into the challenges of making fully navigable 360° video with hyperlinks. Inside a Web page context, video interaction is typically limited to actions such as play and pause, fast-forward and reverse. Hypervideo is an attempt at creating a high-interactivity format, where video is not just an illustration, but can also comprise links and other high-level structures in time and space, such as text and imagery. This early prototype does not account for vertical distortion, and as such videos appear compressed. Furthermore, the only tool for navigation is the mouse, and one must drag the video to pan the viewport. Despite its limitations, this system incorporates some usability aspects that can be brought to VR, namely a radar-style orientation widget and a mini-map of video content with the location of each annotation. The authors argue that spatial awareness is vital to understanding and finding desired annotations, and as such, fine-tuning of time and space related properties for each annotation is necessary to avoid collisions. Additionally, supporting and clearly identifying different types of annotations helps the users navigate this format. For instance, one type of annotation could be a click-to-expand billboard containing information, and another type could be a link to a different web page. These two must be visually distinct so as to not break the viewer's engagement with the video.

Like this one, other consumer applications – that is, those outside the professional price and quality range – have begun to appear. For instance, Pavel et al. (2015) have developed a tool for video navigation using synchronized captions. While not designed for VR, it demonstrates how navigation between different contexts can be trivialised for users. The system aggregates and integrates information surrounding a video piece, such as captions and subtitles, the script, plot summaries, and video chapter subdivisions – as seen on Figure 2.7. With the advantage of being designed in collaboration with film studies researchers and film professionals from its inception, the tool provides a validated solution for video navigation that can be readily applicable to VR in a 360° video context.

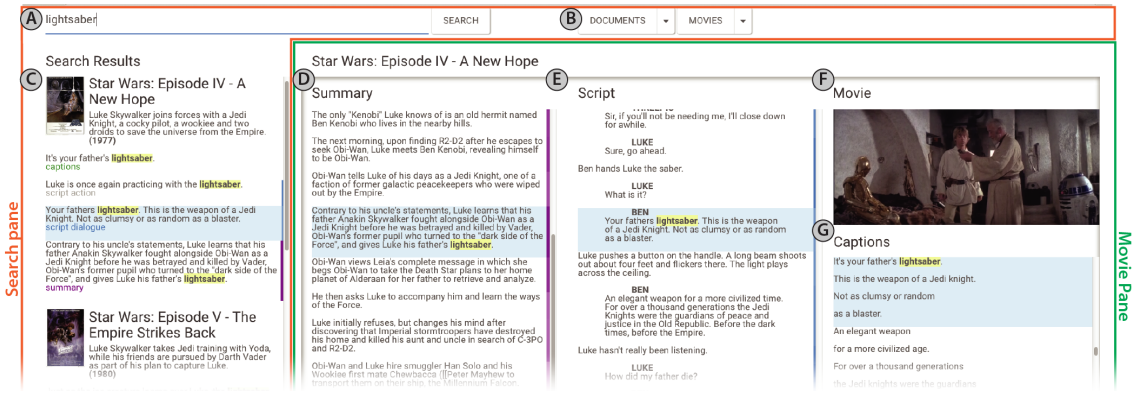


Figure 2.7: SceneSkim prototype interface. Picture from an article by Pavel et al. (2015).

Regarding Human-Computer Interfaces studies specifically, Janzen et al. (2016) present a study on pointing performance and user perception. The authors investigated the impact of target depth on mid-air pointing – in which the user's hand moves freely in the air – considering how common of a task it is in both training and simulation applications as well as real-world scenarios when pointing at a 2D target. The focus of the article was to learn which pointing performance models translate into a virtual environment for different target depths. The testing method would present users with three scenarios, depending on distance to a hypothetical physical screen. In this case, the virtual screen was rendered exactly where it was supposed to be (that is, where the user pointing capture was being made). Other scenarios included the virtual screen being both further away and closer to the user than their physical counterparts.

One of the hypothesis initially proposed by the authors was that the physical distance to this physical screen would have no effect on performance, as the virtual environment would override the illusion of binocular depth. This was proven false – the user ends up perceiving the physical distance mechanically as they move. The authors argue that this might be a result of imperfect calibration, and furthermore propose that a function of the distance between physical and virtual targets can be used to further calibrate the virtual environment. The models that predicted this deviation also predict that it should not exist within a perfectly calibrated virtual environment, which suggests that calibration must be automated on a per-user basis, or that there is a fundamental flaw in stereoscopic 3D rendering that needs to be addressed at a deeper level.

The authors remark that the rates at which the system process different types of information – such as sensory and motor data – will result in a perceived miscalibration, and suggest that validating and measuring this hypothesis is a question to be examined in future research.

Regarding this calibration, the authors state that already accepted models fail when applied to ray-casting solutions in VR, even when the latter is reported as more comfortable. This research work's findings might be able to shed light on new calibration techniques that could be automated. The authors argue that while such techniques exist on an experimental level, they focus on reducing perceived distortion. This means that to achieve perfect calibration, performance metrics must be taken into account. To fully match real-world pointing, binocular depth must be adjusted on a

per-user, per-session basis. However, the fact that the metrics for virtual and physical targets can be measured automatically regardless of the application means that binocular depth values can be tuned within the application, during a one-time calibration phase.

When one conceives applicability scenarios for hand sensing, especially within the scope of computer-assisted tools and environments, one can quickly understand that incorporating user manual dexterity in the design of products can be a valuable asset. Kim and Bae (2016) introduce a 3D sketching and modelling system that incorporates one such hand-tracking sensor – allowing the user to use their hands as reference. By including already established paradigms the prototype is capable of acquiring and modifying hand posture, as well as quickly adjust drawing planes to enable 3D sketching of graspable products, as seen on Figure 2.8. The main advantage being that the designer is able to explore their ideas *with* hand information, *without* the need to draw them – hands are a complex structure and thus very hard to draw. Because this type of product is inserted directly into an already established workflow, one can identify scenarios where special care was taken into providing more comfort for the users. One such case is the use of menus. Since menus break the user's interaction flow they are best left off to gestures, especially those that can be performed in the air. One must always consider user fatigue, however. While this specific prototype avoids shortcuts that conflict with editor features and restricts gestures to pen taps, hand motion in the air can be an useful control mechanism if one utilises it sparingly.

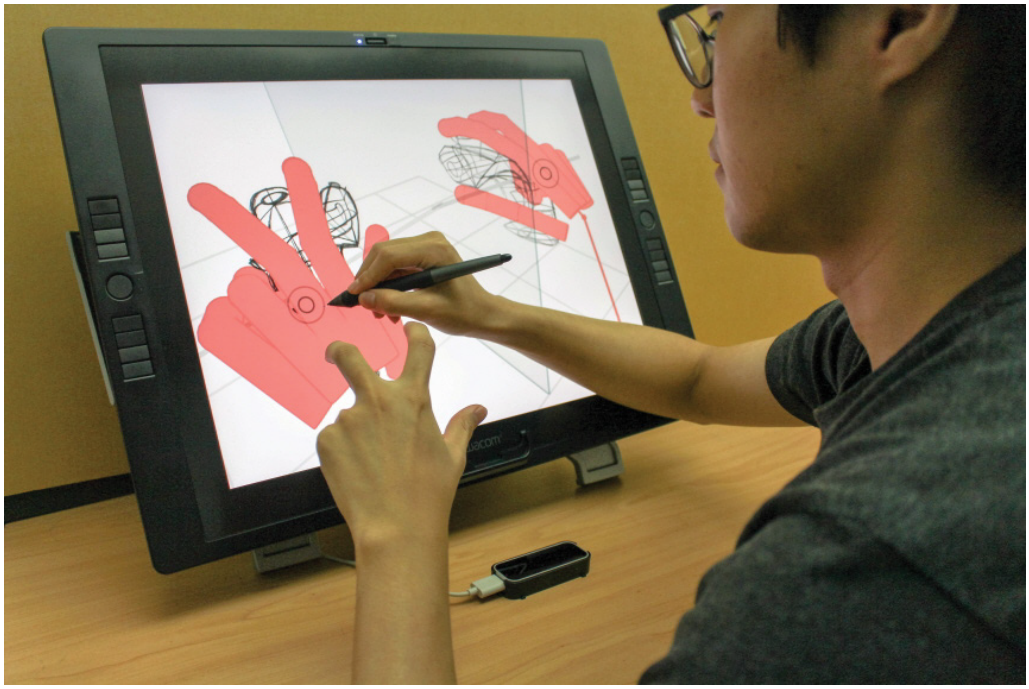


Figure 2.8: SketchingWithHands use-case. Picture from an article by Kim and Bae (2016).

When considering the HCI in CAD and virtual environments literature presented above, one will find full-fledged products and working prototypes, rather than theoretical considerations or

abstract algorithms. The selected works for this section are introduced in table 2.5. For completeness and readability, the section on hand detection was merged into this one. Furthermore, both sections share features involving detection of natural interfaces, such as hands, and their usability as control mechanisms.

Table 2.5: List of HCI, CAD and virtual reality interaction systems.

Number	Title	Author(s)
C1	Get around 360° hypervideo	Neng and Chambel (2010)
C2	SceneSkim: Searching and Browsing Movies Using Synchronized Captions, Scripts and Plot Summaries	Pavel et al. (2015)
C3	Modelling the Impact of Depth on Pointing Performance	Janzen et al. (2016)
C4	SketchingWithHands: 3D Sketching Hand-held Products with First-Person Hand Posture	Kim and Bae (2016)
C5	Proactive Sensing for Improving Hand Pose Estimation	Hsiao et al. (2016)

Even though it is a diminutive selection, the feature list is more consistent as the area of focus gets narrower and more works share the same concerns. As it can be seen, on table 2.6, user interface is a carefully weighed area of any project. It is a component which must be deeply integrated within the system lest its performance be hindered.

Table 2.6: Comparison of HCI, CAD and virtual reality interaction systems.

	C1	C2	C3	C4	C5	This dissertation
Navigable overlays	✓	.	.	.	.	✓
Search feature	.	✓	.	.	.	✓
Adjustable interface	.	.	✓	.	.	✓
Natural interaction (hands)	.	.	✓	✓	✓	✓

2.5 Validation Solutions

Another approach is the statistically-validated study. Busch et al. (2014) have conducted extensive surveys to assess user engagement and learning ability in virtual environments. While the focus of the study is presence in these kinds of environments, other variables naturally become factors in research – virtual environments tend to manifest negative effects and lower user performance. However, habituation and the human tendency to associate presence and usability means that while virtual and real environments cannot be considered statistically similar, virtual environments do not impact usability and learning ability. The pragmatic differences between the two types of environments are rather small, when one considers experienced presence. Thus, selecting a virtual environment over a real one becomes a matter of cost and personal presence, more than performance and usability issues.

As mentioned before, when one realises that a virtual environment inside which reality is simply pre-recorded rendition – rather than a modelled one – is simply a shift in the fidelity and device

awareness within the reality-virtuality continuum, one can argue that such a virtual environment is in fact a mixed-reality environment as proposed by Milgram and Kishino (1994).

Müller et al. (2016) present a study on whether the addition of virtual objects influences collaborative tasks in co-located mixed-reality environments. Curiously, when one approaches video editing and the playback experience as a collaborative task between the producer and the viewer, many of the assumptions and conclusions reached to by Müller et al. hold true. Imagining, for instance, that the producer will want to guide the viewer's attention to a point in space, that space becomes shared. As such, deictic gestures that would be employed in a live scenario stop being available and new virtual ones are necessary. As shown, physical space plays a diminutive role in user perception, and virtual elements such as annotations replace deictic gestures as spatial cues, decreasing user task load. Another interesting takeaway is the usage, by part of the authors, of the NASA Task Load Index (NASA-TLX developed by Hart and Staveland (1988)) test to assess workload. In the context of this thesis, the workload is borne by the producer or editor, who have to use their hands to operate the proposed prototype. With this in mind, implemented solutions will attempt to mitigate fatigue during prolonged exposure.

User satisfaction is a common research topic for computer systems since software that places a high level of burden – or workload – on users can see its adoption and retention negatively impacted. In his paper, Brooke et al. (1996) proposes a questionnaire with the intent of evaluating the appropriateness of a system on an usability scale, named the System Usability Scale (SUS). This system can be used to provide a relative outlook on comparable systems. Since the proposed solution aims to assess the adequacy of different sets of input and output environments, such a scale should provide a comparable measurement of user satisfaction with each. Furthermore it would allow the use of the same scale in future iterations.

2.6 Brief Consumer VR Device Overview

The technological domain of this thesis can be divided into three parts:

- Three-dimensional hand and gesture detection
- Head-mounted VR tracking and image display
- Data processing and rendering

Each of these is analysed and presented below. Opting to separate the components helps to properly compile a state of the art list, and justify each choice.

2.6.1 3D Hand Sensors

When reviewing optical 3D sensors, one can divide the market according to three operating principles: Structured Light, Time of Flight and Stereo-Vision as defined by Weichert et al. (2013).

Structured Light sensors are composed by one or more cameras and one or more light sources. The operating principle is that of a projection of pre-determined light patterns onto the target scene

and processing the distortion caused by objects in the form of a depth channel, as Fofi et al. (2004) state in their survey. One such example is the Microsoft Kinect [5]. While this device presents a suitable alternative, Khoshelham and Elberink (2012) demonstrated that it is not capable of robust sub-centimetre precision with full range of motions. Furthermore, Silberman and Fergus states that the Kinect sensor has a 12V input requirement due to the camera cooling systems, which would make a mobile or head-mounted setup quite heavy. The HTC Vive wand-controllers employ a structured light approach as well. It is capable of tracking the HMD as well as two wand-controllers based using two base tracking stations, as seen on figure 2.9. The system works by sweeping an area with multiple synchronised light pulses and laser lines from the base tracking stations and, paired with inertial measurement units on each controller, pinpoint the position and orientation of each sensor with sub-millimetre, sub-millisecond precision[6]. This technology has been made available to the public as well, and both training and licenses for building original hardware are available[7].

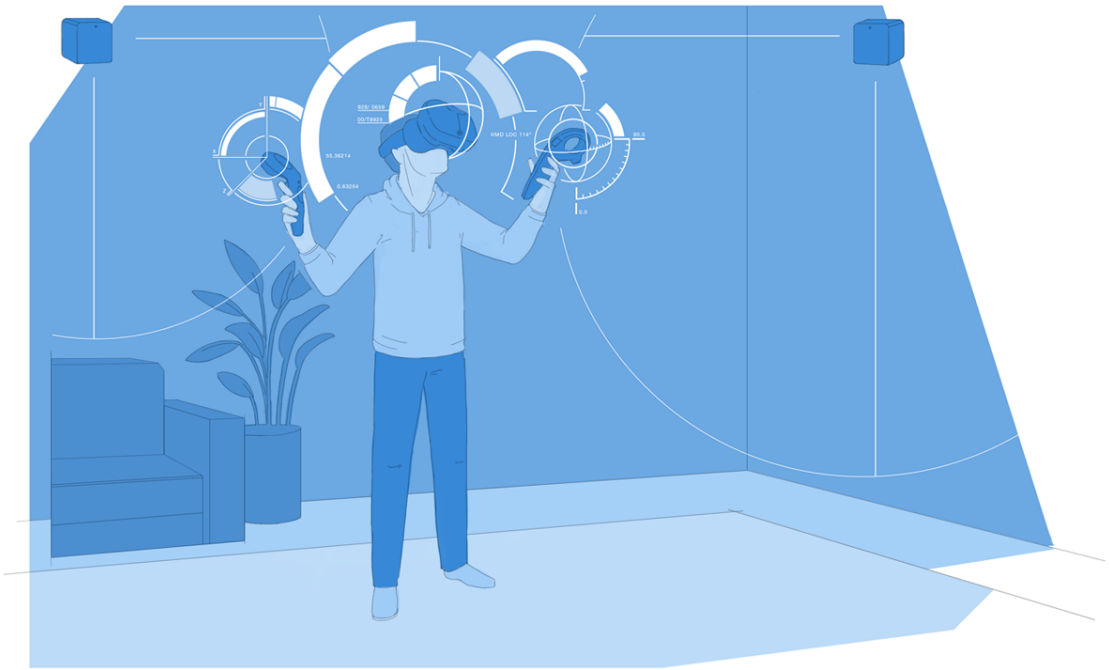


Figure 2.9: HTC Vive and wand-controller usage[6].

Time of Flight sensors utilise modulated light beams in conjunction with camera shutter techniques to obtain and correlate reflected light data as defined by Kolb et al.. The hands' positions

[5] Kinect for Windows – https://developer.microsoft.com/en-us/windows/kinect. Last accessed: 2016-12-26

[6] HTC Vive and SteamVR Tracking – How it works. https://partner.steamgames.com/vrtracking#HowItWorks. Last accessed: 2017-06-25

[7] Synapse – SteamVR Tracking Training – http://www.synapse.com/steamvr, Last accessed: 2017-06-25

would be extracted from this raw data. While these sensors can reach affordable prices, Malawski shows they still are not capable of maintaining errors below 5mm. Furthermore, the lack of an Application Programming Interface (API) would significantly inflate development times. Stereo Vision cameras are devices encompassing two optical cameras that operate on known external conditions. Depth is calculated based on the correlation of points between the two camera images. Additionally, these sensors are capable of providing Cartesian coordinates for the inferred points. The Leap Motion sensor falls into this category, as it consists of 3 infra-red LEDs and 2 infra-red cameras, as pictured below.

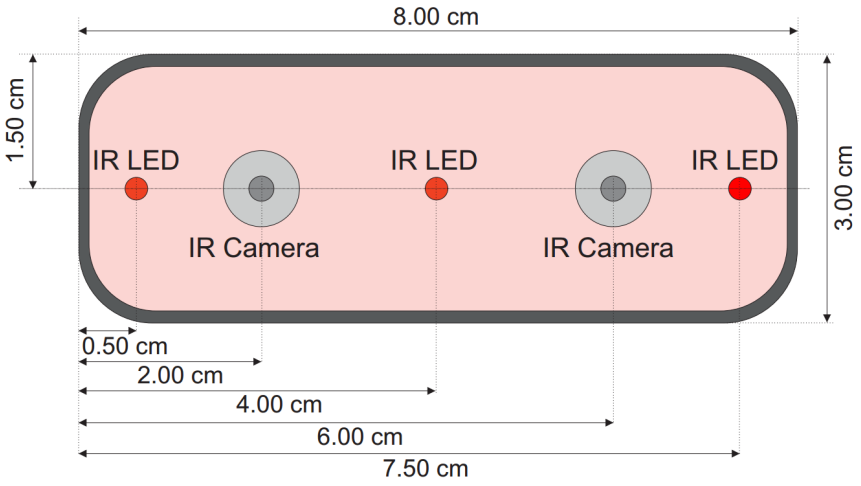


Figure 2.10: Leap Motion schematic. Picture from an article by Weichert et al. (2013).

In this case, Cartesian coordinates are provided relative to the centre infra-red LED. The device functions off a USB connector, and its reduced size makes it suitable for a head-mounted setup. Furthermore, Leap Motion provides an extensive C# API, optimised for hand detection, as well as several Unity 3D engine assets and packages, to be discussed in sub-section 2.6.3.

2.6.2 Head-mounted VR Displays

The Oculus Rift[8] was chosen for a number of reasons, among which are availability, affordability, compatibility with the other technologies (Leap Motion and Unity 3D), and an extensive API. Specifically, the Oculus Rift Development Kit 2 (DK2) was used. Later in development an HTC Vive[9] was acquired, and the prototype was expanded to accommodate it. This device is bundled with two wand-like controllers and as such provides a different type of interaction when compared to the Leap Motion controller. Other options included the Samsung Gear VR – which requires a Samsung phone to achieve full capability – and Google Cardboard, both of which are mobile

[8]Oculus Rift homepage – https://www.oculus.com/. Last accessed: 2016-12-26
[9]HTC Vive homepage – https://www.vive.com/eu/. Last accessed: 2017-05-25

solutions. Therefore, they only support lower resolutions and exhibit an under-performing API (Unity 3D compiled for Android devices).

2.6.3 Data and 3D Rendering

Rendering the 360° video on a VR environment requires an inverted sphere onto which a texture is applied. The dynamic nature of annotations necessitates a rendering engine capable of handling 2D and 3D objects in real-time. Thus – and because this project requires external inputs – a video-game engine was the logical choice. There are currently two free video-game engines with vibrant communities and support – Unity 3D and Unreal. The choice was Unity 3D due to familiarity with the editor and the scripting language C#, as well as previously validated work on this area using this engine.

2.7 Concluding Remarks

This chapter provides a definition for Virtual Reality within the context of this dissertation. Several research projects were then analysed to provide a scientific state of the art background for each of the main contribution areas: 360° video editing and playback, human-computer interaction in virtual environments, hand gesture recognition, and immersive virtual reality rendering and display mechanisms. A group of validation solutions were also gathered, and a feature comparison was elaborated.

Furthermore, a technology review was performed in which adopted technologies were compared to their alternatives.

Chapter 3

Architecture of the Immersive 360° Annotation System

This chapter presents a detailed description of the architecture of the proposed solution. It was possible to provide a virtual environment in which 360° videos could be played back, complete with media controls and annotation path editing interfaces. With the authoring of content being a one-time process isolated from the metadata describing its behaviour, the same information can be used in subsequent interaction with the platform, allowing users to use the proposed solution exclusively for annotating the video, without having to stop to change content.

The requirements of the aforementioned proposal will be formalised and analysed in section 3.1. Subsequently, an early prototype developed in the wake of a previous research project will be described in section 3.2 under the lens of the established requirements and lessons learned. With these in mind, structural decisions will be explicit in section 3.3, with first the contextualisation of the system in terms of work-flow integration, and then from a high-level architectural point of view. Finally, section 3.4 presents the set of researched interaction techniques. These techniques are well known in standard reality computer and mobile phone applications, but they stand to be translated – or transformed – into VR interaction techniques. They set the basis with which to tailor the design side of user interaction. As such, menu interface design is also presented in this section, as well as its operation.

The proposed solution will then be used to assess the feasibility and usability of editing facilities as introduced in the research questions described in section 1.3. It serves as a technological proof of concept for compatibility and data format questions by implementing such paradigms and exposing users to operation scenarios.

3.1 Requirement Analysis

Based on the research questions, an initial set of requirements for an immersive editing tool for 360° videos was derived. However, before detailing these requisites, a group of key concepts will be described in section 3.1.1. After it, the requirements that shaped the development of the

solution are described in section 3.1.2. In addition to these requirements a set of constraints that were needed to ensure the adequacy of the implemented prototype are described in section 3.1.3. These are not necessarily requirements, but still need to be taken into consideration for the sake of quality.

3.1.1 Key Concepts

Before producing the requirements, there are two core concepts that must be clearly defined. The first is that of **annotations**. As primarily informative objects, annotations deviate from simple subtitles due to the capability of displaying multimedia content. Furthermore, they exempt from the static, fixed-position nature of subtitles and are free to occupy any point in space relative to the video, especially in 360° environments. Figure 3.1 shows the traditional annotation process of a 2D video. The most immediate applications of annotations are the use of imagery and rich text overlays to enhance the already existing content. While arguably a subtitle could also display a simplistic rendering of rich text, a direct implementation of this paradigm would still not contemplate images. Annotations can then satisfy both the content and placement necessities that subtitles can not. A possible approach is one that unifies the different content types – images, text, etc. – under a single structure, and gives free rein to the author in terms of content.

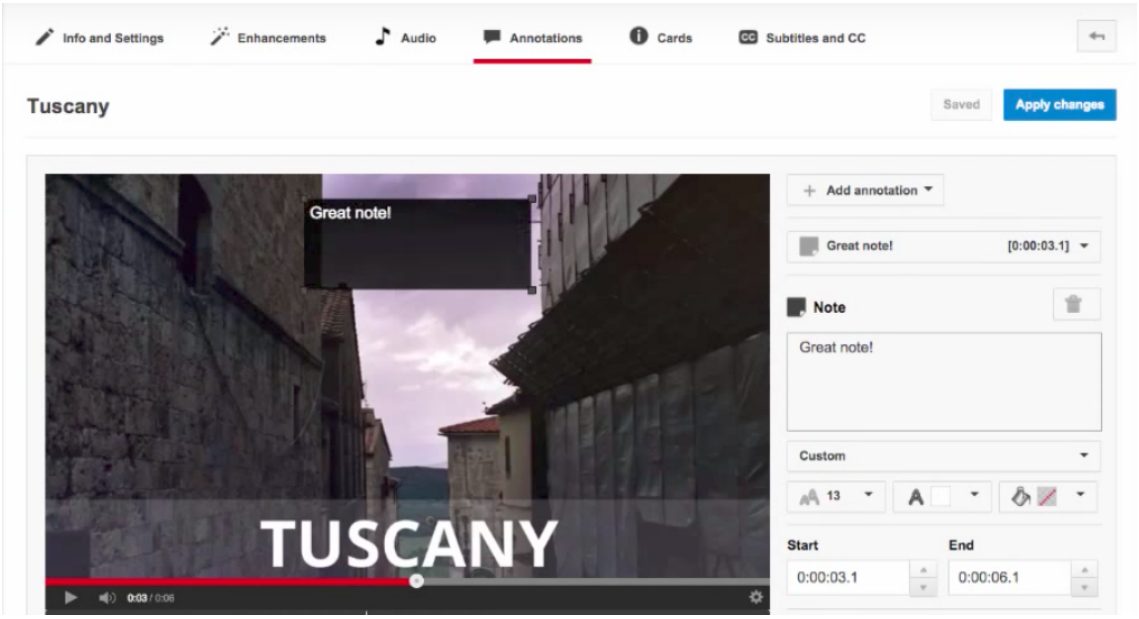


Figure 3.1: Example of traditional annotation process. Users select the type, content, position and timing of each annotation using a post-production editing method. Picture from YouTube[1].

[1] YouTube Help: Annotation types – https://www.youtube.com/watch?v=g2P-jxDhjCo. Last accessed: 2017-06-18

The positions of annotations in 3D space can vary. Based on the a priori knowledge that there exists a set of known 3D point-time pairs that fully describe "key" points of the annotation's position life-cycle, then all remaining positions for arbitrary values of time can be interpolated from those "key" points, known as **keyframes**. An illustration of this interpolation can be seen on figure 3.2.

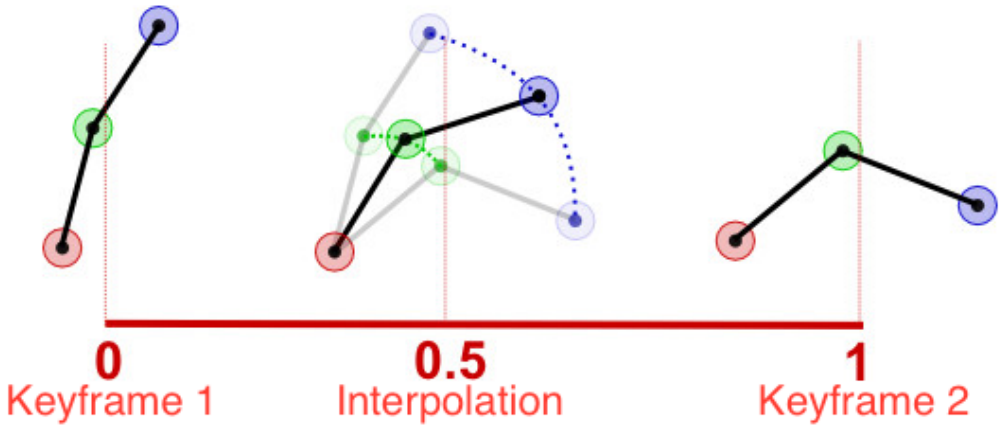


Figure 3.2: Interpolation between keyframes. Example of the interpolation of a spline shape between two keyframes over time. Picture from Real 3D Tutorials[2].

As the second core component, keyframes are, first and foremost, data structures – but not limited to the context of annotations. They hold two fundamental variables: a point in a value space and a point in time. Now by defining a keyframe as the position of a 3D object for a given discrete time, one can use an arbitrary number of keyframes to define the path for the motion and timing of said object in 3D space. The use of keyframes – a process called keyframing – is not limited to the placement of 3D objects. It is, rather, associated with the context of animation – it can be used to "animate" properties through time. For instance, one can use keyframes to animate the blurriness of an object during a transition, by defining "key points" containing the value of the parameter and their timing. An example of this is presented on figure 3.3. The main facility of the keyframing process is however the ability to use interpolation between keyframes to avoid having to describe every single value of a parameter. The set of keyframes of an annotation becomes thus a set of turning points – they need only mark where motion starts, changes or ends, and every position in between is interpolated automatically. One can see then why the keyframe object is independent from the annotation object.

[2]Real 3D Tutorials: Tutorial 17 - Keyframe interpolation – http://www.real3dtutorials.com/tut00017.php. Last accessed: 2017-06-18

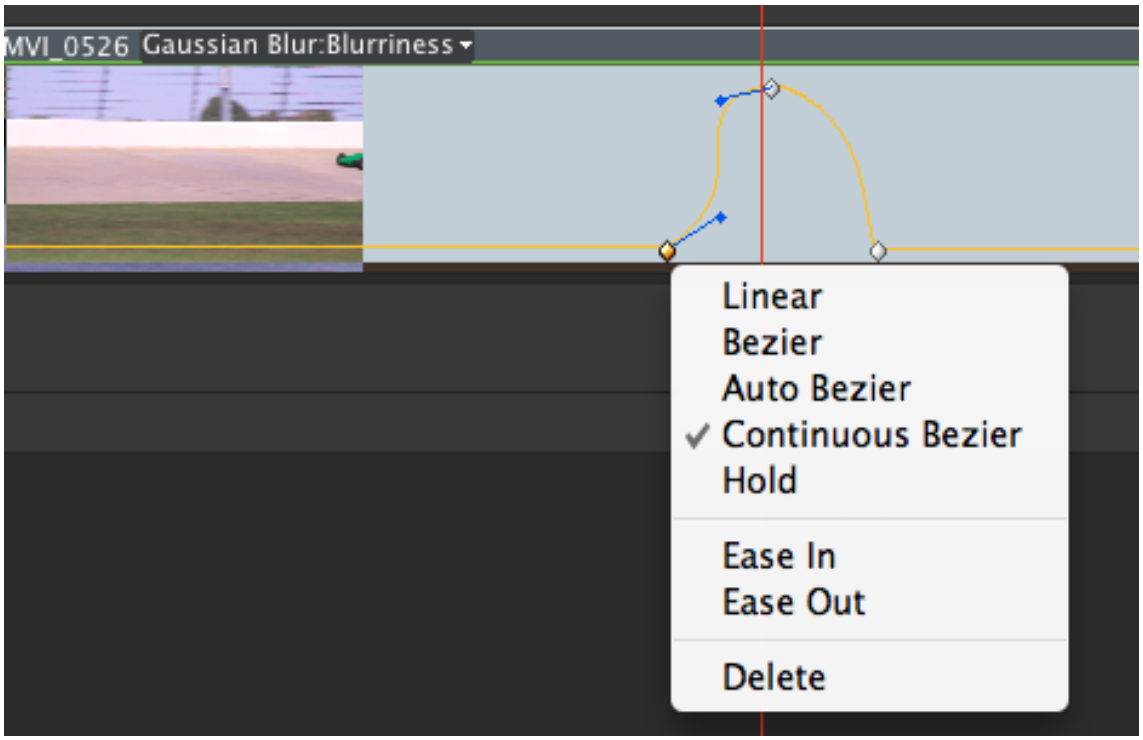


Figure 3.3: Keyframing applied to animate a "blur" property within Adobe Premiere Pro. Keyframes are indicated by a diamond shape in the timeline. Trademark of Adobe Systems, Inc. Picture from ProVideo Coalition[3].

To conclude, the supported hardware remains important to at least warrant a mention here as it may shape the requirement analysis process. Each set of input and output peripherals – defined in section 1.2 as an "environment" is recalled below:

- Desktop screen with keyboard and mouse
- HTC Vive head-mounted display and HTC Vive wand-controllers
- Oculus Rift head-mounted display and Leap Motion controller

3.1.2 Overall description

Focusing on the research questions refined in 1.3 it is possible to define a group of user needs that must be met, even if the absolute minimum requirement is to just establish a testing platform. Presented below is a systematic overview of the system requirements. These are elicited in a list format, with a brief description of each requisite statement.

Requirement 1 – **Correctly play back digital 360° video with adequate field-of-view restraints, both in desktop and VR environments**, since users will be enveloped in a digital

[3]ProVideo Coalition: The Adobe Premiere Pro timeline for Final Cut Pro users by Scott Simmons – https://www.provideocoalition.com/the-adobe-premiere-pro-timeline-for-final-cut-pro-users/. Last accessed: 2017-06-18

rendering of 360° content at all times. Naturally only a sub-section of the video will be visible at any time due to the field-of-view constraints of the human eye. While it is arguable that one can display the full 360° imagery compressed to fit a 120° field-of-view – making it completely visible at all times – this is not natural and offers no advantage over traditional 2D editing of 360° footage.

Requirement 2 – **Provide navigation facilities within the 360° environment, both in the desktop and VR versions, using keys and head movement respectively.** Considering that the intent is to enable users to edit content overlaid on a 360° video it follows that the application must be able to respond to navigation within this type of media. The user should then be placed at the centre of the action and be able to pan the 360° environment – or simply their point of view.

Requirement 3 – **Import metadata for each annotation from an external source, effectively converting all necessary elements into objects within the virtual environment.** A mechanism for placing objects in 3D space according to an *a priori* description is needed. This can be achieved by importing external metadata.

Requirement 4 – **Render the created 3D objects in their appropriate positions and timings, and displace them according to established keyframed properties over time.** The system should provide a way to render the aforementioned overlaid content – in the form of annotations – in real-time, so that users can both see the aspect of the edited content and perform the edits themselves within the same environment.

Requirement 5 – **Allow users to pause and resume both the 360° video content and the annotation overlays.** Pausing the video explicitly has consequences over the annotation overlays, as these 3D objects should be synchronised with the 360° video at all times, and therefore must also be paused and resumed in tandem with the media.

Requirement 6 – **Allow users to stop both the 360° video content and the annotation overlays, instantly rewinding them to the beginning.** With the ability to stop and rewind to the start instantly, users can travel back to any point by playing the video again.

Requirement 7 – **Allow users to travel to a 360° video time of their choosing, with the annotation overlay synchronisation following suit.** This operation is called seeking and enables users to instantly travel to any point in the video with sub-second precision.

Requirement 8 – **Allow users to add new keyframes to existing annotations.** 3D objects containing the annotations will have to alter their path – sometimes even their current position – to comply with changes to the keyframes that control them, all in real-time. Adding keyframes is a powerful way to extend an annotation's path.

Requirement 9 – **Allow users to remove specific keyframes from existing annotations.** Likewise, removing a keyframe enables annotation paths to be trimmed or truncated at will.

Requirement 10 – **Allow users to alter the position in space of existing keyframes.** If a keyframe is at the wrong position, so will the annotation at that time. Altering the position of a keyframe independently is crucial in micro-managing annotation behaviour.

Requirement 11 – **Allow users to alter the timing of existing keyframes.** The timing of a keyframe can cause annotations to enter or leave the scene at incorrect times, or move at inadequate speeds. Adjustment of timings grants finer control over annotation motions.

Requirement 12 – **Allow users to add new annotations to the scene, with no regard for content.** Without annotations, keyframes alone do not have significance. The application structure must support not only the loading and creation of annotation objects from external sources as described in Requirement 3, but also provide users with the facilities to create annotations from within the application.

Requirement 13 – **Allow users to remove specific annotations from the scene.** Since keyframes micro-manage the path of an annotation, the removal ability stands as a powerful macro-management tool.

Requirement 14 – **Allow users to change perspectives and look at different parts of the video, on desktop and VR environments.** The most fundamental control aspect within a 360° virtual environment is the ability to pan one's perspective, enabling users to look at any part of the video using continuous motion. Lacking other inputs in VR environments, this can be achieved using head motion.

Requirement 15 – **Allow users to manipulate objects in the scene, on desktop and VR environments.** Part of the immersion resides in the direct manipulation of 3D objects, such as annotations or keyframes. This should reflect traditional interaction paradigms such as pointing, selecting and dragging. Such interaction should be as natural as possible, using any available input devices.

Requirement 16 – **Allow users to invoke and interact with global functions, on desktop and VR environments.** Operations transversal to the entire 360° environment, such as controlling playback, should have symbolic commands or interfaces that can be accessed from anywhere, at any time. The resulting implementation of Requirement 15 can be used to further extend this interaction.

Requirement 17 – **Allow users to save authored edits to an external file without destroying the original source.** Since maintaining a non-destructive method is essential

to the application's philosophy, the environment should be able to export the authored work to separate files such that it can be reproduced for viewing or the edited at a later time.

Requirement 18 – **Gather data pertaining to user movements and actions, especially regarding selection and dragging interactions.** Data gathered during each session should pertain primarily to user movements and actions, especially those directly related to the input methods and/or directly indicative of user intention such as the interactions resulting from Requirement 15.

Requirement 19 – **Export the gathered data as a formatted text file – such as *.csv* – such that it can be analysed easily.** This directly concerns the testing framework, and should be treated as a plug-in in the sense that it should not impact the core functioning of the solution.

3.1.3 Constraints and Specific Requirements

One of the most important constraints, especially due to the nature of VR rendering, is performance. While traditional video content is typically distributed with frame-rates between 23 and 30 frames-per-second – such is the case with Hollywood film DVD releases – most traditional platforms support playback rates of up to 60 frames-per-second. However, to prevent motion sickness in VR, HMD technology is capable of refresh rates of over 90Hz. This is not to say that content must conform to this. Content can still be played back at 30 frames-per-second as long as the device is capable of rendering a new image at 90Hz, so that user sight accompanies their head movement. The first constraint – Constraint 1 – states therefore that the virtual environment's update and tracking cycles should target this rate aggressively.

When one of the main purposes of the proposed solution is to comparatively evaluate the usability of different devices though, compatibility becomes a major concern. Even when only one single operating system is supported and deployment facilities are a not an issue, one must guarantee that the application is essentially the same for all input devices. There is a duality to this – on the one hand, established common elements must not hinder the development of device-specific features; and on the other, these specific components should not require major changes to the architecture to work. Compatibility with the multiple target devices should be prioritarily maximised. This further defines a set of constraints, culminating in the following list:

Constraint 1 – Enforce accurate rendering of content as well as a target frame-rate superior to 90 frames-per-second.

Constraint 2 – Enforce a comparable experience across multiple devices, especially – but not limited to – desktop screen and VR HMD for outputs, and keyboard, mouse, HTC wand-controllers and Leap Motion for inputs.

Constraint 3 – Establish interface uniformity and device-independence, possibly excepting punctual interaction metaphors.

These should delineate the constraints for this particular software fairly thoroughly. Other plausible constraints are already defined by the requirements themselves, and their inclusion would be redundant.

3.2 Early Prototype

Having established the aforementioned requirements it is important to recall the lessons learned from previous implementations and prototypes. As described by Meira et al. (2016), this early prototype deals exclusively with playback, although it is complemented by an external program to facilitate editing. Playback and annotation rendering are both achieved using the Unity 3D engine, much like the proposed solution. This early prototype can open a video file and a modified *.srt* file containing annotation metadata. This prototype however only supports plain text-based annotations, with no support for rich text.

3.2.1 General Description

While this prototype focuses on achieving content playback with acceptable fidelity, it became necessary to also develop authoring tools that allow for fast iteration. Supposing that the editor wants to annotate a 360° video piece, the set of desired annotation points must first be identified. As described, the annotations in this prototype are purely textual. To identify each item during the authoring process, however, a colour was assigned to each annotation. Figure 3.4 shows a still image – a frame – of such a 360° video piece, with the coloured spots overlaid on it to mark desired annotation points. Note that the content of the annotations is still undefined at this point.

In this case, only static points like buildings are shown, but it is desirable that annotations are capable of accompanying moving subjects during the course of the video as well. These coloured spots would then also move, essentially meaning this overlay is a video as well. This technique is called masking, and the overlaid composition is a "mask" – in this case it is called a video mask because it is, in itself, a video. This mask can be created and rendered within traditional video editing software independently of the video – editors would produce two files, the unadulterated 360° video file and the mask file containing the coloured spots.

The conceived editing system uses this technique integrated in a pipeline capable of generating human-readable annotation files using traditional video editing software. This is a one-time process and is done in two steps:

1. A video mask is rendered by the user, composed of coloured spots on a black background, each of which corresponds to an annotation and its position and timing on the video (an example of the mask by itself can be seen on figure 3.5). This type of video can be easily created using video editing software. Using the "screen" blending mode (featured in most



Figure 3.4: Coloured spots marking annotation points in a frame of a 360° video. Represented in this image is a single frame of a 360° video in 2D format, with the coloured spots marking the desired annotation points.

mainstream editors) allows a transparent layer overlap of content and mask, thus allowing to paint actual objects on top of the video.

2. This video mask and an optional regular subtitle file are parsed in a computer vision algorithm that generates an augmented *.srt* subtitle file compatible with the prototype. The latter contains annotation and keyframe metadata generated from automatic video image analysis of the video mask. This does not modify the original files in any way.

Parsing in this case is done off-line and not within the application. This avoids starving the processor of resources as computer vision and real-time image analysis are very taxing operations, which would indubitably impact performance. An example of the video and annotations running in-engine can be seen on figure 3.6.

This method proved to be lacking in terms of optimisation, as it would generate an unnecessary amount of keyframes due to flawed path optimisation strategies. Each frame in the video mask would be seen as independent, with which a colour-based analysis would be performed. The exaggerated number of keyframes had a marginal toll on performance, but so diminutive that it could not be discerned to the naked eye without a performance profiler.

3.2.2 Key Takeaways

This early prototype validated the use of annotations as useful conductors of viewer attention during 360° video playback. Much like regular subtitles, the worth of dynamic annotations is dependent on the producer's ingenuity – if too tame the purpose of informing is defeated, and if too

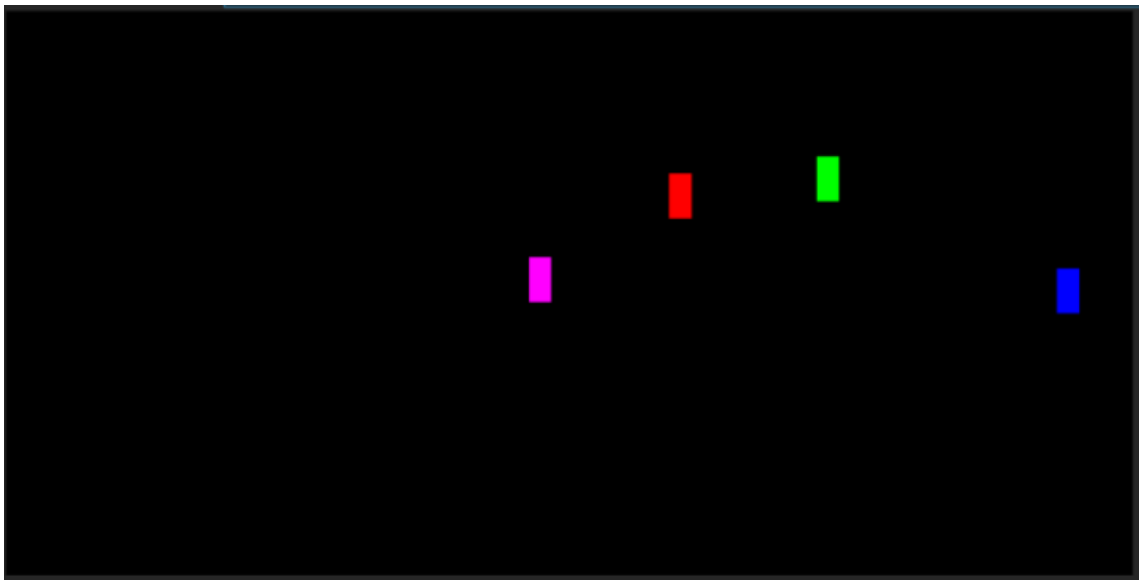


Figure 3.5: Annotation video mask. This is a single frame of the video depicting four annotated locations. This mask can be created using a layer in any common video editor software.

intrusive viewers will perceive the information as cumbersome or condescending. The conducted tests with this early prototype show that these paradigms are valuable assets nonetheless.

Moreover, the importance of smart annotation path-making strategies became apparent, as interpolation becomes a dead weight if there is a keyframe on each frame of the video and further throttles performance. A better approach than restricting the paths that users can create is to guide and optimise this process so that it is both more application friendly and in line with the producer's views. By letting editors preview and tweak the path of annotations using discrete control points the surplus of keyframes is minimised and paths are overall smoother.

Another important remark is that of the aspect of the authored content. As mentioned, this early prototype only supports text-based annotations. Additionally, they can exhibit a coloured circle next to the text to easily identify the annotation. This colour is assigned to the annotation based on the spots used in the video mask, but can be changed at a later time by altering metadata on the augmented *.srt* file. An example of this type of annotations can be seen on figure 3.7. While it serves a basic informational purpose, it is incapable of following a style guide.

One important aspect related to this is the organisation of metadata within the files. In this early design, data is described in the YAML language and injected in the beginning of a regular *.srt* subtitle file, augmenting it. This metadata is ignored by traditional 2D video players, therefore maintaining the retro-compatibility of the subtitle file. This "described metadata" approach was maintained in the proposed solution, albeit with different schemas.

In a journalistic environment, such as that of a news 360° video piece, there are elements that serve both branding and informational purposes, with a focus on visual appeal. These follow a per-programme style guide and vary wildly, so it is natural that producers and designers alike would want additional elements like annotations and subtitles to conform to the same style. Since

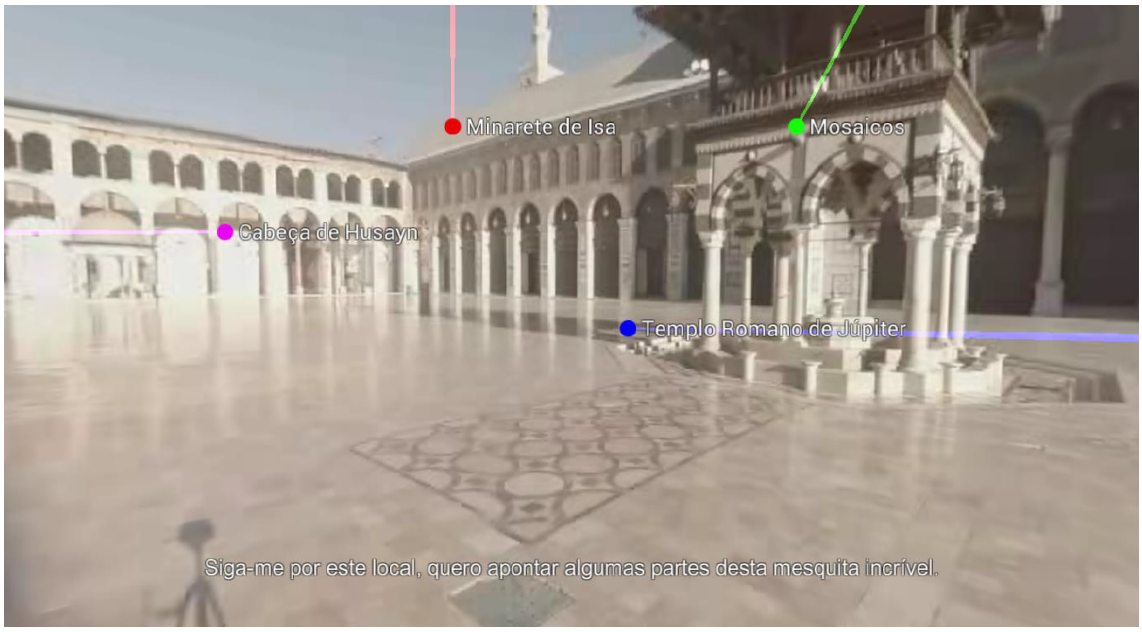


Figure 3.6: Example of the early prototype operation. Each annotation is assigned the colour used in the video mask. Furthermore, coloured lines indicate the direction of each annotation to guide the user.

the prototype is incapable of such customisation, it follows that it is one of the key aspects to improve. It is unrealistic to predict multimedia arrangements to suit every producer's needs, but it is possible to provide a scheme with which to construct them, such as the combination of HTML and CSS as employed in webpages.



Figure 3.7: Example of a textual annotation with a red coloured marker.

3.3 Proposed Solution Overview

Since the requirements referred above were all aimed at providing a framework with which to answer the research questions raised in section 1.3 then, in principle, an implementation contemplating the fulfilment of such requirements would constitute a fair solution. Taking into account the primary conclusions of the aforementioned prototype, this section means to highlight a proposal to match exactly this. Firstly, a brief introduction of the established systems will be exposed in section 3.3.1, and after it the interaction with the proposed system will be detailed. Section 3.3.2 will illustrate the top-level architecture, identifying the relevant processes and systematically define their communication.

3.3.1 System Context

This document's objectives stand by assessing each device on equal grounds and with simple graphical user interfaces. This is because more complex interactions would have to be device-dependent, eventually biasing the response. If one is indeed going to implement such interfaces, one should study users' device preference.

This serves to illustrate that it is not uncommon for solutions like the one proposed in this document be become integral parts of an already existing system. Given that the posited software has as primary input a set of files, it is relevant to understand how these are created, and how the solution fits into the existing workflows. Figure 3.8 shows a diagram illustrating this scenario.

In the case of 360° video content there are several steps between the acquisition of imagery by a 360° camera and the obtention of a playable video file. After the capture – typically through multiple-camera set-ups – one must condense the footage from each individual camera into a single

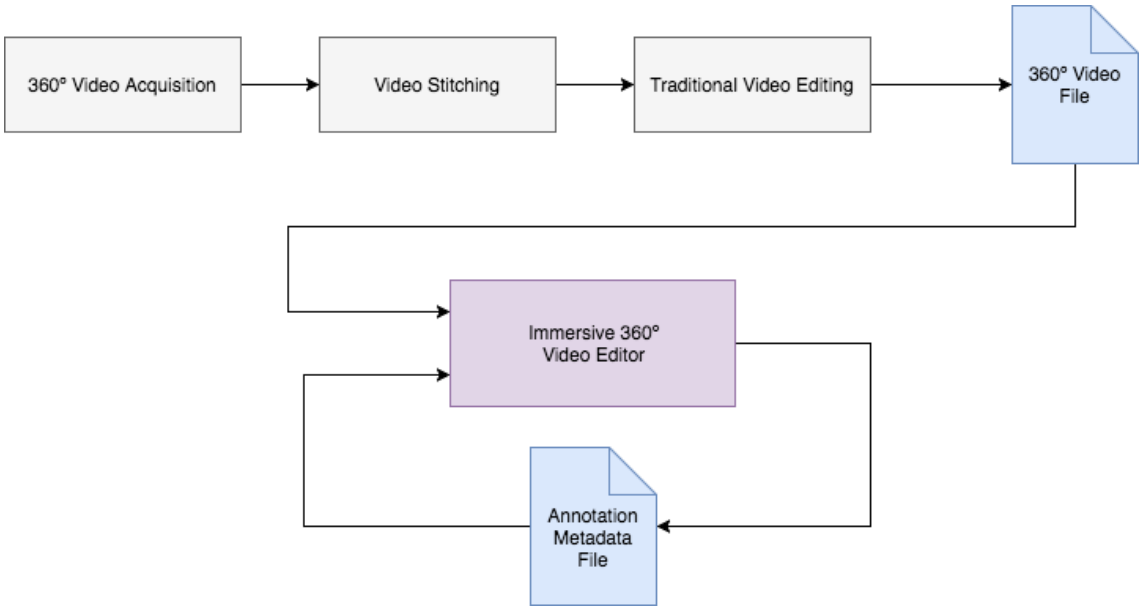


Figure 3.8: System integration diagram . In purple, the proposed system; and in blue, its input files. Gray elements are external components used to create and maintain such files.

video. This presents additional challenges as different dynamic parameters, such as exposure, will create noticeable seams at points where the images from different cameras overlap. Hence the name – stitching – for the process of eliminating this side-effect. During this step distortion is also applied to prepare the video – in reality a 2D image – for the adequate projection type on 360° environments. Having concluded this, the video is ready to be edited using traditional means – audio, dubbing and transitions can all be added at this point. Finally, a video file is produced containing a 360º video mapped to a 2D canvas, with distortion. This file is responsible for the video content, and is consulted by the application at start-up, along with the annotation metadata.

Regarding the metadata file – responsible for describing annotations and their content – the process is similar, albeit more flexible. Taking advantage of the fact that traditional subtitle files are typically stored as text files, one can conceive an extension to this format that contemplates this annotation metadata. Any sort of text editor can be used to edit traditional subtitle files, so if metadata are also text they can be used to augment the traditional subtitle files. Another way of creating a metadata file is to use the proposed solution – by adding, removing, or modifying existing annotations, one can generate a metadata file for those parameters.

Since there are no facilities within the developed system that allow for actual video content editing, the source 360° video file is left unaltered, serving only as an input. Despite also serving as an input, the metadata file is the only output of the platform. However, the original file is not replaced. Instead, a new file is created that is in itself a metadata file. Naturally, this file can also be edited using any text editor.

The proposed solution then becomes a non-intrusive part of the video production pipeline, by placing itself at the end of the video content production life-cycle.

3.3.2 System Architecture

After understanding where in the already-established workflow the proposed solution stands, one must then plan which type of connections will be made to external sources, as well as how that data will be used internally.

Figure 3.9 depicts a very high-level architectural diagram with flow information. More specifically, the different types of relationships, but also interaction with the user.

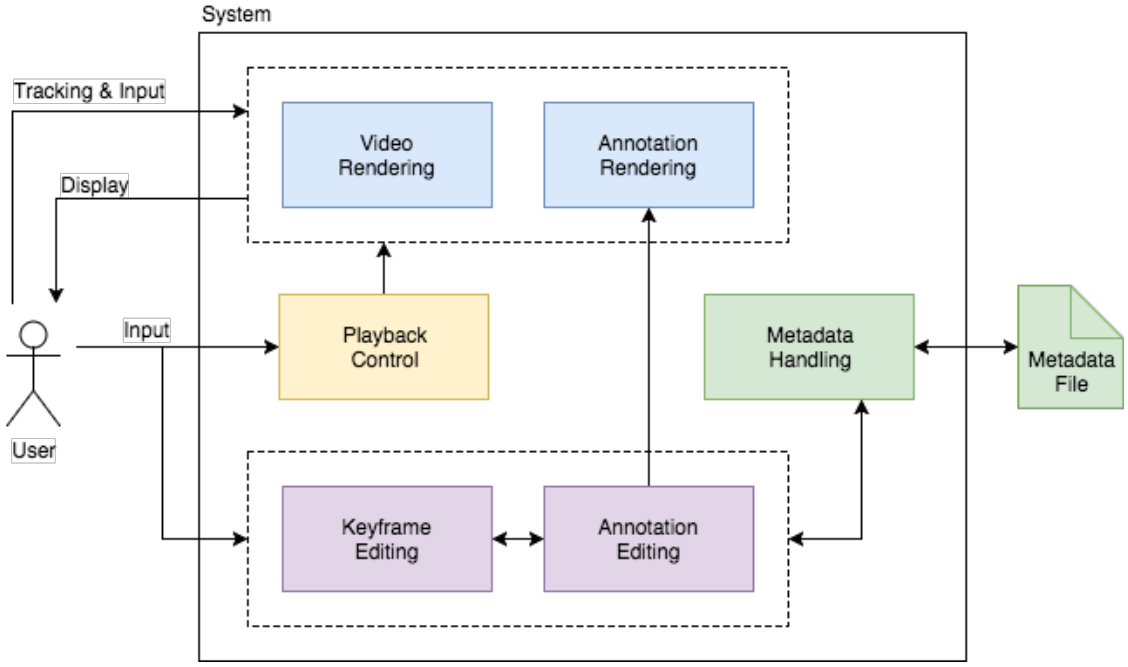


Figure 3.9: High-level architecture relationship diagram. Arrows indicate the direction of each relationship. Unidirectional lines indicate explicit control while bidirectional lines illustrate autonomous interaction.

As established, there is a facility with which users can watch 360° videos in an immersive environment. The foremost feature is content rendering – both the 360° video and the annotations overlaid on it (in blue) are authored media that come in direct contact with the user through the display. They are therefore responsible for generating the virtual environment. However, users need to be able to control such elements. This is done via the input devices. Firstly, the flow of the video can be manipulated using playback controls (in yellow). These naturally affect the annotations as well, as timing is a crucial factor controlled by the video flow. Secondly the annotations themselves can be manipulated using the provided editing tools. These tools come in the form of high-level annotation controls, and keyframe micro-management on a per-annotation basis (in purple). To conclude, a mechanism is in place to handle the life-cycle of metadata (in green). This system is responsible for exporting and importing the state of the annotation editing to and from an external file.

3.4 Interaction Techniques and Design

Before delving into the design of the interfaces *per se*, it is important to understand how to inform users about exactly what type of data is being modified by their actions. This section approaches this assuming that due to the immersive nature of VR environments, conveying information via classical interfaces can have reduced efficacy as it overwhelms users.

Interaction techniques are techniques that condition the ways in which users operate the application, and whose behaviour is disjointed from the implementation.

In this section the main interaction techniques will be discussed in a manner that is independent of the interfaces they are applied to, with a focus on their translations to each different device.

A basic description of pointing is given in section 3.4.1. As the most basic, it will introduce the structure for the following sections: starting with the mouse version of the technique one is able to identify key points that are then deduced for both the HTC Vive's wand-controllers and the Leap Motion's hands interface. Additionally challenges that arose during development will also be addressed, as they shaped the overall feeling of the experience.

Selecting is presented in section 3.4.2 as the primary means of user choice. While pointing is a continuous operation, selecting is spontaneous and occurs during pointing – yet is separate from it. The most interesting aspect of this section is how a binary operation – as, for instance, selection – translates to a button-less interface like the human hand.

In section 3.4.3 dragging is introduced as the combination of pointing and selecting. It retains the continuous nature of pointing while giving the user control over when it starts and ends. While natural interfaces like hands should excel at this type of interaction, years of mouse usage have conditioned most users to identify external tools as natural extensions.

A first-approach design of the user interface is presented in section 3.4.4. Further analysis and the elaboration of higher fidelity models allow this study to reflect the final desired menu interface more closely, using the established interaction techniques.

Finally, menu interaction is addressed in section 3.4.5. While in theory menu operation is achieved by a combination of the three already-discussed paradigms, several questions remain unaddressed. For instance, one must consider how users can invoke the menu, among other practical considerations of maintaining a 3D interface.

3.4.1 Pointing at Objects

Pointing is the foundation of any modern user-centric operating system. Since a computer screen can be abstracted as a 2D canvas, one can get from any arbitrary position to another by means of a sequence of 2D transformations. For example, in modern operating systems navigation within the graphical environment is done through a pointer or cursor. Users typically control this pointer with the use of a mouse – the movements of this device on the 2D surface that is the table can be translated almost directly to the 2D surface that is the screen, making it efficient and intuitive.

Moreover, the act of pointing can be used as a trigger for interactions by itself. By pointing at an item, additional non-intrusive information can be displayed, such as style changes or tooltips. The act of pointing at a target is called hovering, as the pointer appears to hover over it.

It should be noted that this type of interaction is not applicable to touch-screens. Each tap in a touch-screen instantly makes the cursor travel to that position. Often times, cursors don't even exist in touch-screen-based environments. This was taken into consideration when developing the interaction for hand movement, as users tend to maintain their preconceptions regarding interaction.

3.4.1.1 Pointing with Keyboard and Mouse

Since users are already using a computer to interact with the application, it is fair to assume they have access to a keyboard and a mouse. The virtual environment, however, does not exhibit the same properties as a 2D screen. Instead, users have to navigate the internal surface of a sphere. In the context of the application, since the user is always at the centre of the sphere, only a section of its surface is visible at any time. The exact dimensions of this surface are given by the intersection of the camera's viewing frustum – which can be modelled as a rectangular pyramid whose base is defined by screen size and field-of-view – and the spherical surface. The result of this intersection is a warped rectangle. In Euclidean geometry theory, when the sphere's radius approaches infinity, this warped rectangle approaches a plane rectangle as long as the arc generated by the diagonal remains the same. In practice, this means that for the sphere-centric user, this warped rectangle will appear to be a plane rectangle, onto which mouse movements can be translated. To avoid mapping problems at the corners and edges where the warping is more noticeable, a 3D cursor placed in world space was used.

Moving the mouse then moves this cursor – mouse movements in screen space are projected onto the sphere to affect the world space position of the cursor. The presence of a 3D object to act as a cursor also allows for more direct collision checking, as one can simply check colliders against one another without having the need to explicitly process raycasts.

It is possible to detect if a user is pointing at an object and trigger certain behaviours, like colour changes. This opens up new alternatives for interface design, as hovering becomes a non-intrusive option to relay feedback to the user.

3.4.1.2 Pointing with the HTC Vive Controllers

The fundamental difference when working with the HTC Vive wand-controllers is that they move in 3D space around the user. Furthermore, they have a natural orientation that can change, meaning that rotating the controllers is also possible. Operating this device is then a combination of arm and wrist gestures that control 3D models of the controllers in the virtual world.

Their shape is called wand-like because it is approximately cylindrical with height being significantly larger than the radius. A more direct comparison would be that of a television remote control. A peculiar aspect of these types of controllers is that years of television operation have

conditioned users to use these controllers like laser pointers – point them at the device and press a button. While in television remote controls this aspect is essential to functioning, mainly because they employ optical communications, in a virtual environment this is unnecessary as the controllers' position and orientation are being tracked by the system.

To further encourage this kind of operation, a line resembling a laser is drawn at the tip of the right hand controller as seen is figure 3.10. Users can point at objects by moving and rotating the controller as they would a laser pointer. Much like the mouse version, a cursor object is drawn in world space to indicate where exactly the user is pointing. This also makes it easier to point at distant objects. However, this cursor is only displayed when in the vicinity of an object. This is done because otherwise with both a controller and virtual laser on screen the visuals just for pointing would be overwhelming.



Figure 3.10: Example of the virtual laser serving as a pointing aid with the HTC Vive controllers.

Since the cursor object is still in world space at all times – just not visible – it still retains all its physical properties, such as the collider that triggers hovering behaviours on objects.

3.4.1.3 Pointing with the Leap Motion Controller

Pointing with hands is not very widespread in computer interaction, as the most common touchscreen interfaces use discrete touches. And yet, children use their hands to point before even

learning to talk. At the root of the act of pointing is human perception – fine pointing is always done relative to a person's gaze. The most straightforward pointing method is to use one's index finger. But with the Leap Motion controller being attached to the VR headset, fingers are occluded by the back of the hand when pointing forward. This would mean that pointing would be imprecise (or impossible in some cases), unless one induces a deviation in pointing that forces the user to keep their hands at an angle which would make the index finger visible, but this is not comfortable as hands would occupy a larger part of screen real-estate.

A sensible approach is to use the palm of the hand or wrist to point. One must not fall into the trap of naïve execution, though. For instance, if one uses the facing of the palm of the hand directly, one forces users into awkward and fatiguing positions to point at anything above eye-level. Other example is using the wrist's orientation to point, forcing users to extend their arm completely to point forward.

A more thoughtful approach must account for both comfortable arm and hand posture, as well as user perception of pointing. For instance, when arms are retracted and hands close to the user's head, one can draw a line passing through the user's shoulder and wrist, and this line will reflect user's intention with acceptable accuracy. This holds even with arms extended. When pointing at distant objects, the orientation of the palm of the hand is not relevant. Additionally, when the arm is extended, the palm will be facing down, further reinforcing that direct mapping of the palm's orientation would be counter-intuitive. It is also possible to fine-tune pointing direction with movements of the hand, as these are typically accompanied by movements of the wrist, effectively correcting the pointing direction.

Like previous cases, a laser and a world space cursor object exist, as seen on figure 3.11. Like the previous case, it is also omitted unless near an object.

3.4.2 Selecting Desired Elements

Selecting is the act of explicitly triggering an interaction with the object one is pointing to. This is the primary method of interaction when navigating an operating system's environment. This is because when travelling to a desired object, users sometimes have no choice but to hover over multiple elements on the way. If hovering (or pointing) was the primary method of interaction, these elements would be unintentionally triggered along the way. Selecting is introduced as a binary action – users are either attempting to select an object, or they aren't. As such, this is typically done resorting to physical buttons, which are also binary. These can be the click of the mouse, the tap of a touch-screen or the pressing of a button on a controller.

Selecting occurs at the behest of the user, after pointing to an object. This event is propagated to that object so that proper behaviour is triggered. Moreover there is no analogue of hovering for this interaction. This is because pointing is a continuous action, while selecting is spontaneous and instantaneous. However there is an interesting consequence of this binary operation – since the state of the selection changes between on and off, one can analyse state transitions, and trigger events based on those. It works almost as a derivative – when the user presses a button, state goes from off to on in what can be referred to as "button down"; when the user lets go of the button,



Figure 3.11: Example of the virtual laser serving as a pointing aid during Leap Motion usage.

this state goes from on to off, resulting in a "button up". This essentially means that selecting can provide two distinct types of interaction.

3.4.2.1 Selecting with Keyboard and Mouse

Selecting for the mouse is the exact same as within the operating system environment – after pointing to the desired object, pressing the left mouse button will trigger a selection event in that object.

Additionally, state transitions are accounted for, although this does not require any additional hardware – it is simply a state machine within the application.

3.4.2.2 Selecting with the HTC Vive Controllers

The HTC Vive wand-controllers offer a multitude of buttons. With each controller featuring an ambidextrous design, all six buttons are equally accessible. Perhaps the easiest reachable button is the trigger – shaped akin to a firearm trigger, it provides both a continuous axis as well as a binary operation mode. A set of two symmetrical buttons are featured on each side of the controller, where the palm and fingers rest, called the grip buttons. To trigger a grip event, only one of this has to be pressed. Due to their positioning, a grip can be triggered by tightening the grasp on the controller and either the palm or the fingers will press one of the buttons. The last of the main buttons is in actuality a trackpad, which can detect touching events (without actually pressing the button) on its surface, but also allows physically depressing the button. The remaining two buttons

are the Menu and System buttons. They serve to navigate the HTC Vive's interface, although applications can override their usage. These last two buttons are not so comfortably accessible due to their distance from the thumb's resting position.

The chosen button to accommodate the selection interaction paradigm was the trigger on the back. This button can be comfortably pressed when the user's arm is both extended and retracted, no matter the rotation of the wrist as seen on figure 3.12. The selection is triggered by pressing this button until a click sound is heard, indicative of the binary operation mode. Like on mouse and keyboard, this device registers state transitions. To trigger a "button up" event one must simply let go of this button, letting it default to resting position. To prevent multiple "button down" events occurring without one "button up" event between them, the button must always return to resting position before it is pressed again, or the repeated "button down" events won't be accounted for.



Figure 3.12: HTC Vive user selecting a distant object.

3.4.2.3 Selecting with the Leap Motion Controller

There is no binary operation mode for fingers as human hands are not capable of discrete motion. The most common interaction tasks involving human fingers typically have some type of feedback, either through a tool like a mouse or through a physical surface like a screen. Interaction with recourse to tools allows for discrete "on" and "off" states. When translating such interaction to a completely natural medium, such as the Leap Motion controller's hand detection, one must find a way to discretise hand motion while giving users feedback.

While it may be hard to detect extended fingers when they are occluded by the back of the hand, thumbs are almost always visible. By adopting a pinching gesture – in which the user needs only press his index finger and thumb together – one can attain sufficient gesture precision, even if the index finger's position is being estimated with occlusion. While arguably fingers touching is a binary operation, such is not the case for detection, as often times the 3D model of the hand will exhibit measurement errors such that even when fingers are pressed together in the physical world, they do not appear so in the virtual environment.

To solve this issue, thresholding is applied. In the specific case of this application, when the distance between the tip of the index finger and the tip of the thumb is less than 30 millimetres a selection is triggered. Naturally, this is only valid if the two fingers belong to the same hand. This threshold allows users to use pinching gestures while accounting for detection error, as seen on figure 3.13. Furthermore, the fact that fingers press against each other provides natural feedback, as users know when a selection event has been registered both sensorily and visually.

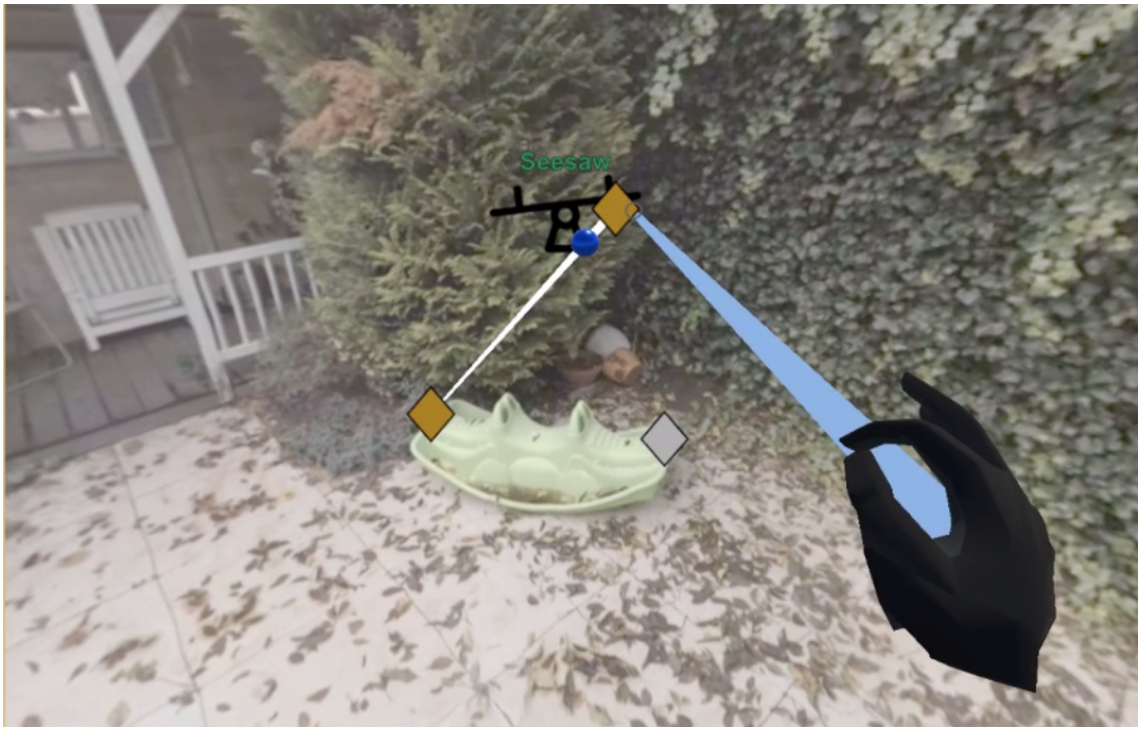


Figure 3.13: Leap Motion user selecting an object by pinching.

Since distance is being continuously monitored as the user moves their hands, a real-time differentiation would yield the user's intention with regards to selection before it is triggered. This provides no additional usability for this case in specific, but is an interesting side-effect of continuous detection. Regardless, state transitions like the ones described for the previous two devices are still accounted for, so "button down" and "button up" are still two distinguishable events. This type of detection is not liable to incur into the multiple "button down" before a "button up" issue, as is the HTC Vive. This is because being a one-dimensional variable, distance can be thresholded unambiguously into one of two mutually exclusive states. There is no physical possibility of the

user triggering multiple "button down" events, because unless the distance between the tip of the index and the tip of the thumb goes over 30 millimetres, the gesture keeps being interpreted as one single "button down" event.

3.4.3 Dragging Distant Objects

Dragging can be seen as a linear combination of the two previous gestures. After a "button down" is registered for selecting, the target object accompanies the cursor in all pointing operations until a "button up" is registered. This means that the selecting button is held during the whole interaction. After the button is released, the dragging operation ends. This type of interaction is prominent in modern operating systems, as it allows for greater abstraction. For instance, to copy a file from one disk to another one simply needs to drag that file from one window to another. Some operating systems expand this to allow file deletion or application installation with dragging operations. Multiple applications employ this paradigm as well, typically called "drag and drop" when independent objects are involved, or "click and drag" when it expresses a continuously tracked operation, such as painting with a brush.

Only keyframes support this operation as they are the only objects that require repositioning. This is done by interacting with a button on the interface as seen on figure 3.14. In reality this button is the one implementing this action. All devices implement dragging the same way as it relies on the already implemented paradigms.

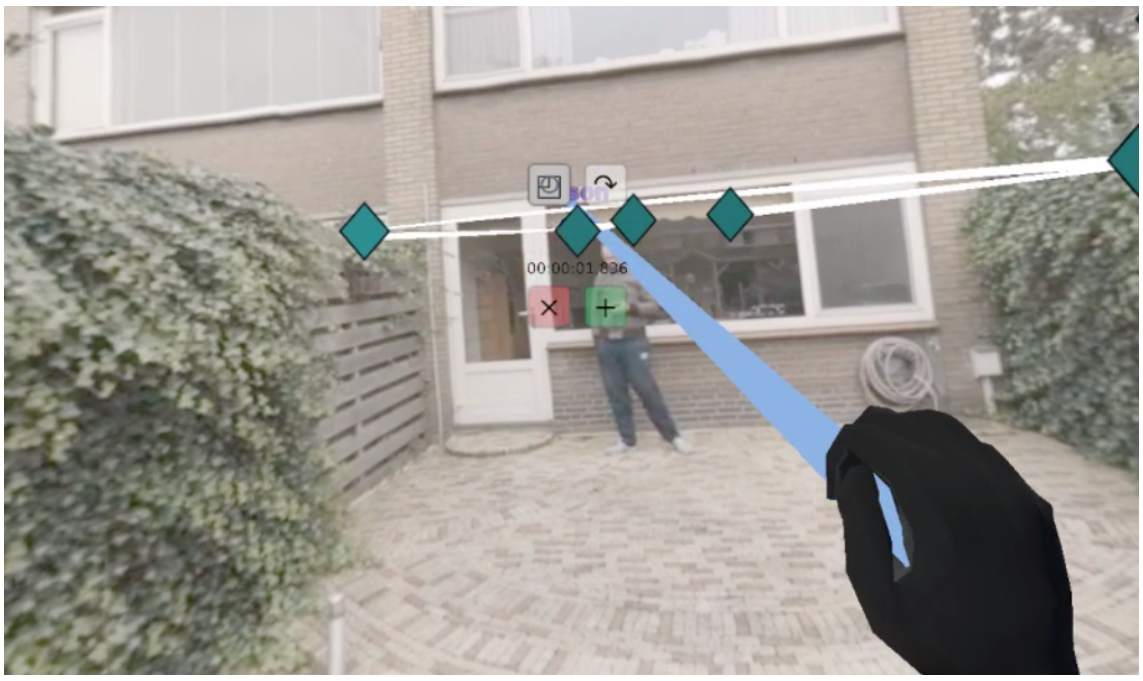


Figure 3.14: User dragging an object using the Leap Motion controller.

The only requirement is that users point at this specified button and select it, and then move the cursor while holding this selection.

3.4.4 Interface Design

Having established basic interaction, one is equipped to design symbolic elements that can be manipulated using these operations. When designing interfaces, however, especially graphical user-oriented ones, it is advisable to start first with minimalist designs that allow for fast iteration. Wire-frames are simplistic representations of an interface's skeleton which serve as a general visual guide for layouts. Focusing on expressing the domain of available functions, wire-frames enable a schematic overview of information priority within each scenario, as well as the transitions between different states. At this level low-fidelity wire-frames are used to aid in communication. They are abstract in nature, allowing the use of basic shapes and filler text to represent content. When this process is concluded high-fidelity wire-frames can be created to further convey the style and production values, as they will match the finished product more closely.

In the context of this thesis, low fidelity wire-frames were constructed using an on-line wire-framing tool[4]. This permitted an arrangement of the layout such that controls could be placed in adequate proximity with compliance to the requirements elicited in section 3.1. After this, high-fidelity wire-frames were structured inside Unity itself – the disjointness between the interface elements and their actual function allowed these objects to be built quickly without needing to consider the implementation behind them. By making use of Unity's drag-and-drop feature one can quickly create user interface mock-ups that closely reflect the final graphical aspect of the application.

3.4.4.1 Low fidelity mock-ups

Of the requisites explicit in section 3.1, Requirement 5 through Requirement 13 specifically target interface concerns. They can be split into three groups – video controls, keyframe controls and annotation controls. Rather than establishing menus for each, it is perhaps advantageous to consider a different division: while keyframes must be controlled independently, video and annotation controls affect the whole scene at once. Moreover these only support macro-operations that do not need to be contextualised to the specific object. For instance, unlike keyframes, annotations are single objects, thus in order to remove an annotation one needs only know its name or other unique identifying property – there should be no ambiguity. This means that one can easily remove an annotation by picking it from a list, while the same is not true of keyframes. Essentially, most of the functionality can be provided by a single global menu rather than a contextual approach for each annotation object. The same obviously goes for video controls – since the video is a constant in the scene, it is logical to provide global controls to influence it.

The chosen approach follows this in that it provides first a global menu with video controls and then supplements it with annotation controls via a drop-down extension. Figure 3.15 shows a low-fidelity wire-frame for the main menu.

This menu features a simple header displaying the "Video Controls" title. below it, a slider with which user can control the flow of the video. To change the currently displayed frame (and if

[4] Wireframe.cc - minimal wireframing tool – https://wireframe.cc. Last accessed: 2017-03-13

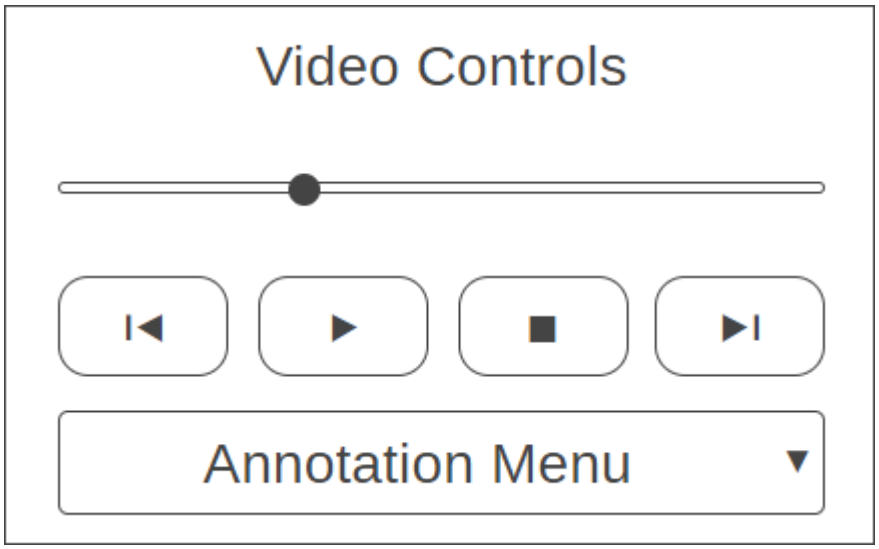


Figure 3.15: Low-fidelity wire-frame for the video controls menu.

the video is playing, resume from that frame) simply drag the slider handle to the desired position or select a point anywhere in the bar to instantly travel to that video time.

Below that, media buttons enable finer control over the flow. The first and last buttons are seek – or travel – buttons, that advance the video by 200 milliseconds with each selection. The second button has two mutually exclusive procedures: if the video is playing, this button pauses the video; if the video is paused, the button resumes playback. The third button is the stop button. When pressed, it halts playback and automatically rewinds the video to its initial frame.

The last button – which works as a drop-down – simply allows the expanding and collapsing of the annotation menu. Figure 3.16 shows this expansion.

This expanded menu provides a feature set that is annotation specific. Firstly, it features a list that works like layer menus in common video- and image-editing applications, with the exception of inter-layer dependency. The order of annotations in this list does not have any bearing on the actual application, and as such permuting layers is not permitted. Each entry in this list will also feature a button to remove an annotation in its entirety.

This list should also feature a "New Annotation" button at the end that allows users to add placeholder annotations to the scene, presumably at a point at which the user is looking at.

With the menu categorically defined above, one must then conceive the contextual menus to be employed in keyframe interaction. The four methods established to be exposed by the keyframe implementation as derived from the requirements are:

- The ability to drag a keyframe object to change its position in space
- The ability to match video time to keyframe time, also known as travelling or fast-forwarding.
- The ability to add new keyframes.
- The ability to remove existing keyframes.

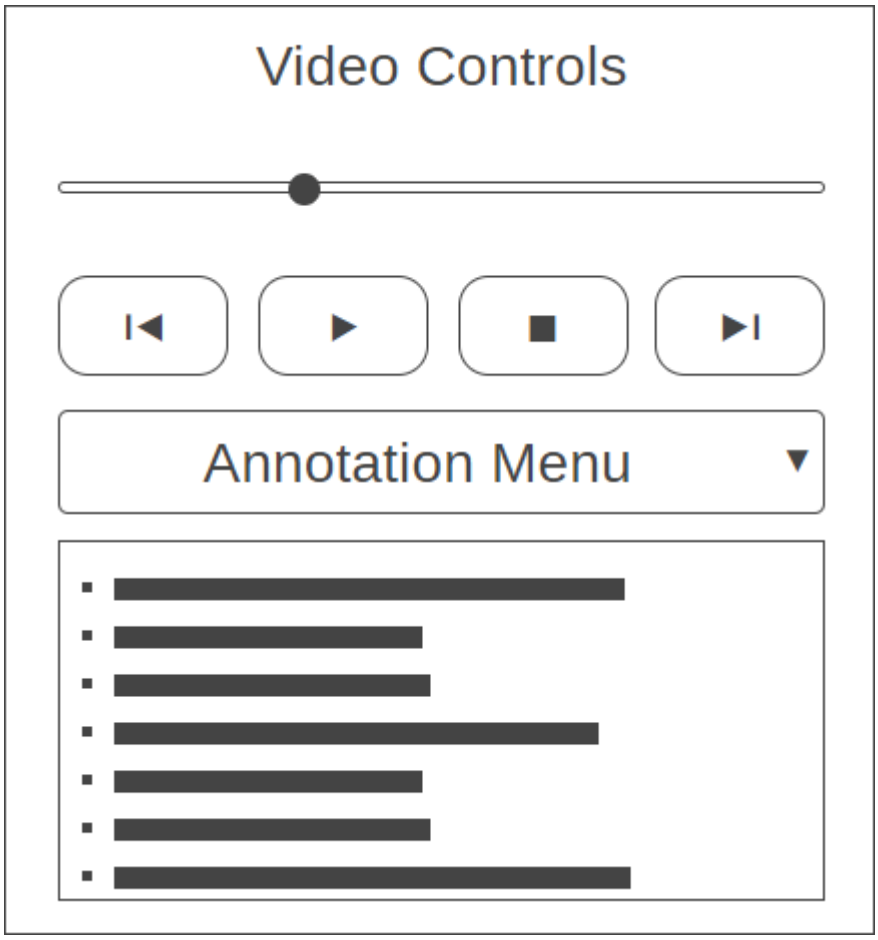


Figure 3.16: Low-fidelity wire-frame for the annotation menu extension within the video controls menu.

Naturally, a first approach is to assign each of these methods to its own button, as seen on figure 3.17. This simplistic approach is one of the most efficient given the planned requirements.

Clicking on a keyframe object summons this contextual menu, after which users can use the provided buttons to trigger each method. Users can move the keyframe by dragging the "Move" button, effectively displacing the keyframe object and the menu with it. Each other button performs their advertised action with just one click. Special emphasis to the "Add" button, which adds a keyframe at the current video time. This prevents additional interfaces for tweaking keyframe time, as users can simply add a new keyframe at the desired time and delete the old one. To dismiss the menu, users need only click on the keyframe object.

3.4.4.2 High fidelity mock-ups

As mentioned in the introductory paragraphs of this section, high-fidelity menus were wire-framed inside the Unity engine. The modular nature of user interface elements as well as their disjointedness from implementation allows for a quick, iterative structuring of the wire-frames defined in the previous section.

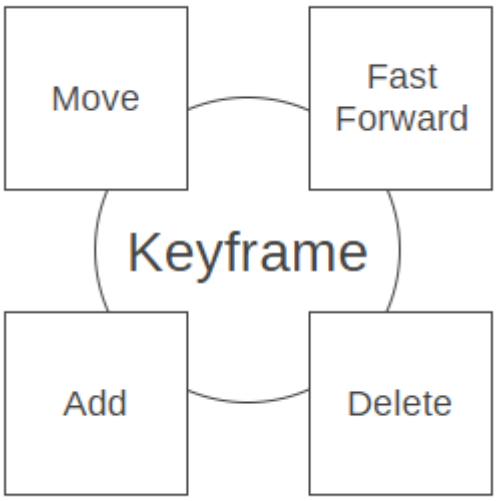


Figure 3.17: Low-fidelity wire-frame for the keyframes' contextual radial menu.

While the translation to an in-Unity interface more akin to the final implementation is relatively straightforward, one must consider usability concerns that are not answered by interface alone. For instance, when adding new annotations, even after the user has made their intention clear by pressing the button, the fact remains that the placement of this new annotation is undefined.

A solution would be to prompt the user with additional interfaces in order to determine an approximate position. However since keyframes are the sole constituents that fully define an annotation, and considering that users can freely move these keyframes, the exact placement of new annotations is not so relevant.

A simpler approach was taken, where the new annotation is placed directly ahead of the user's gaze. This way users can retain some control while avoiding the need for additional, complicated interfaces.

Figure 3.18 shows the high-fidelity wire-frame for the main menu depicted in figures 3.15 and 3.16. All the elements mentioned in section 3.4.4.1 are translated one-to-one, as can be seen with the media controls. However, additional interface elements were added. One such is a label over the slider handle that allows users to read the current video time. This facilitates a more intuitive understanding of the result of each motion applied to the slider handle.

Furthermore, the items in the annotation list were refined at this stage. Each annotation file's name is displayed over a background of its specified colour. Each item also provides three buttons, two of which are toggles: "hide", which renders the annotation invisible when triggered; and "lock" which prevents interaction with the annotation (whether through the menu or through the keyframes). The last button, to the right, allows users to delete the whole annotation, as per Requirement 13.

Regarding the contextual keyframe menus, translation to the Unity engine is much more straightforward as there are no practical usability considerations that are not contemplated by

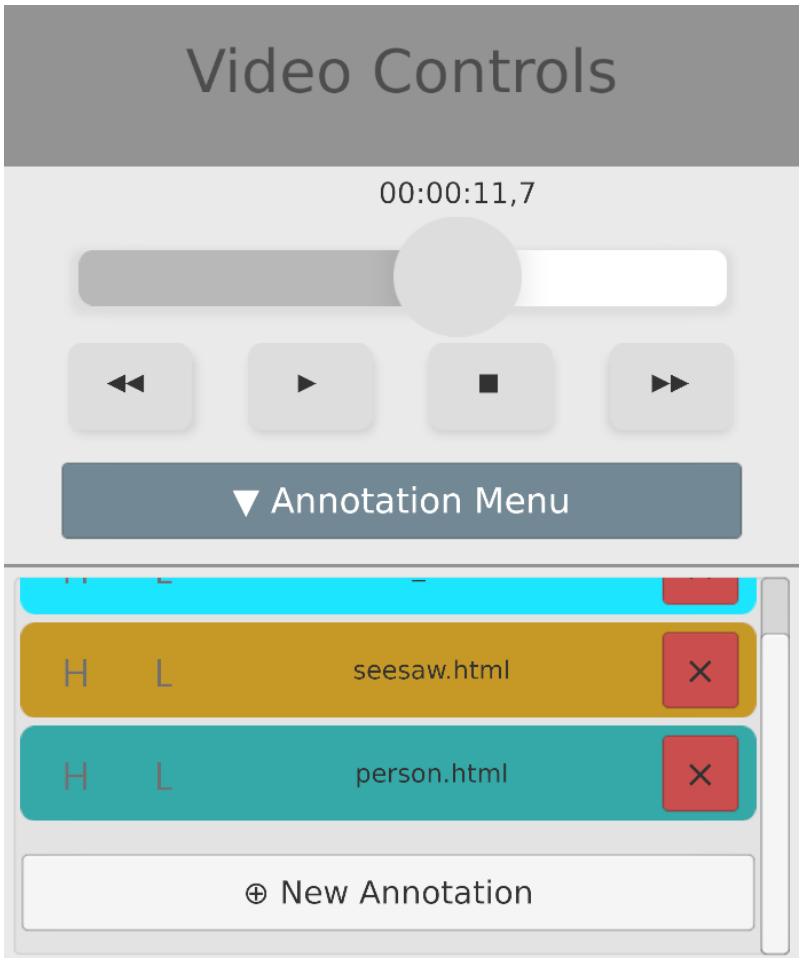


Figure 3.18: High-fidelity wire-frame for the expanded video controls menu.

design or requirements. Figure 3.19 shows the Unity preview of the keyframe menu.

This menu exhibits all the properties elicited at concept and requirement analysis phases. To keep a more condensed interface – and considering some people can not use reading glasses with an HMD – the text within the buttons was replaced with icons. To allow the prototype to retain its flexibility, these icons are displayed in Unicode. While arguably it affects the intuitive understanding of buttons during these initial iterations of the software, it is an intentional compromise to allow for more detailed, scientific analysis of future implementations.

An additional element added at this phase not present in figure 3.17 is the timestamp, placed between the keyframe icon and the bottom-most two buttons. This timestamp is simply a textual element displaying the time at which the keyframe was placed. This allows for a more complete situational overview upon selection of the keyframe, and further helps mitigate any ambiguity that might exist with regards to the relative timing between keyframes.



Figure 3.19: High-fidelity wire-frame for the keyframes' contextual radial menu.

3.4.5 Menu Operation

While arguably menu operation is dependent on the interface elements that constitute it, substantial analysis can be performed on each element separately. The menu is ready to support only three basic interface elements: buttons, sliders and scroll-views. Buttons can work in two modes. Firstly and most basically with regular presses where each press triggers the same interaction. The second mode is toggle, where each selection changes the button's state. The latter works like a checkbox. Sliders are horizontal bars that feature an indicator that discloses a certain value. This value is typically a value between 0 and 1 that can be changed by the user by dragging the indicator or selecting a point anywhere on the slider bar. Lastly, scroll-views are containers that can be larger than the display and allow users to scroll them vertically, much like long web pages work.

Within the application, multiple buttons are used to control video flow, as well as create and manage annotations. A slider is used to both indicate the current time of the video and perform any seeking operations. This can be done by dragging the slider indicator or selecting an arbitrary point of the slider bar. A collapsible scroll-view is used to display a list of existing annotations.

Having established these building blocks, one must then design interaction paradigms for each input device. The navigation within the application is non-linear, as elements can be accessed instantly by pointing at and selecting them. The linear alternative requires the use of additional commands – such as the arrow keys – to move the selection from one element to the other, meaning that elements further away from the initial position take longer to reach. Since pointing and selecting are already implemented, non-linear navigation is both the most user friendly and the most convenient.

Only one second-order pattern remains, that is the method by which the menu is invoked. Unlike contextual menus that display information and options based on the object they apply to, the main menu holds common operations that should be available at any time. To prevent cluttering, this menu must accommodate visibility facilities such as collapsing groups of elements, and be

able to be completely hidden and shown at the will of the user.

3.4.5.1 Menu Operation with Keyboard and Mouse

Again, modern operating systems make extensive use of menus. It is no wonder that users can navigate such interfaces effortlessly with a mouse. As previously mentioned, though, interfaces in the proposed system are rendered as 3D objects in world space. This goes against usual operation, in which menus are 2D canvases rendered on top of the working area in a computer screen, often with a drop-shadow for emphasis. The operating system analogue for invoking the menu is the secondary button – the right-click on the mouse invokes a menu which stays in the same place until an action is taken or the menu is dismissed. Actions are typically triggered by selecting a menu item. Dismissing a menu is done by selecting outside the menu area. In this context, right-clicking anywhere on the working area invokes the menu at that place, dismissing the previous one.

While it is arguable that in modern operating systems this menu is contextual, its interaction paradigms retain their usefulness even for common or shared menus. The main menu within the application is exactly this. Right-clicking anywhere in the virtual environment summons the menu at that location, typically attached to the surface of the sphere. Clicking anywhere on the sphere dismisses this menu. Moreover it is not possible to drag the menu – to move it, simply right-click on a new location. To trigger buttons one needs simply select them, so a left-click while pointing at a button is the correct choice. Buttons that work as toggles can be triggered in the same way, although their colour or text will change to reflect their state. The usage of scroll-views is where menu usage in the virtual environment differs from regular operating system usage. While scroll-views are very uncommon on contextual menus, users are used to scrolling with the mouse wheel. However, since it may not be available, scrolling is done by either dragging the scrollbar on the right vertically, or by dragging the contents of the scroll-view up and down. This type of dragging is common on touch-screen interfaces where mouse wheels are unavailable.

3.4.5.2 Menu Operation with the HTC Vive Controllers

When translating these interactions for VR, one must consider that users typically have two control devices instead of one. More than focusing on optimising the balance in usage, it is important to design meaningful interactions between the two. For instance, instead of designing a completely ambidextrous operation mode that ends up being redundant, one should instead split functionality between the two and have them interact.

Due to the robustness of the HTC Vive's controller detection, one can apply the same pointing and selection paradigms as before for menu interaction with acceptable accuracy. Since these operations are performed exclusively on the right hand controller, it makes sense for menu invoking and positioning to be done with the left hand controller, so they can complement each other further. For this, the menu can be summoned with a press of the trackpad button on the left hand controller. Moreover, once summoned the menu is attached to the right side of this controller, and the user can effectively position it as they desire by shifting and rotating the controller in space.

With the menu open in their left hand, users can interact with it using the pointing and selecting techniques imbued in their right hand as seen on figure 3.20. By pointing the right wand-controller at elements – and with the assistance of the laser that is rendered from its tip – users can select menu items using the back trigger. This approach essentially recycles the operations that users are already familiar with. Interaction is implemented this way for all the described building blocks – buttons can be pointed at and selected, and the same goes for the slider indicator and bar. Scroll-views retain their mouse operation – scrolling can be done using the scrollbar on the right or dragging the contents.

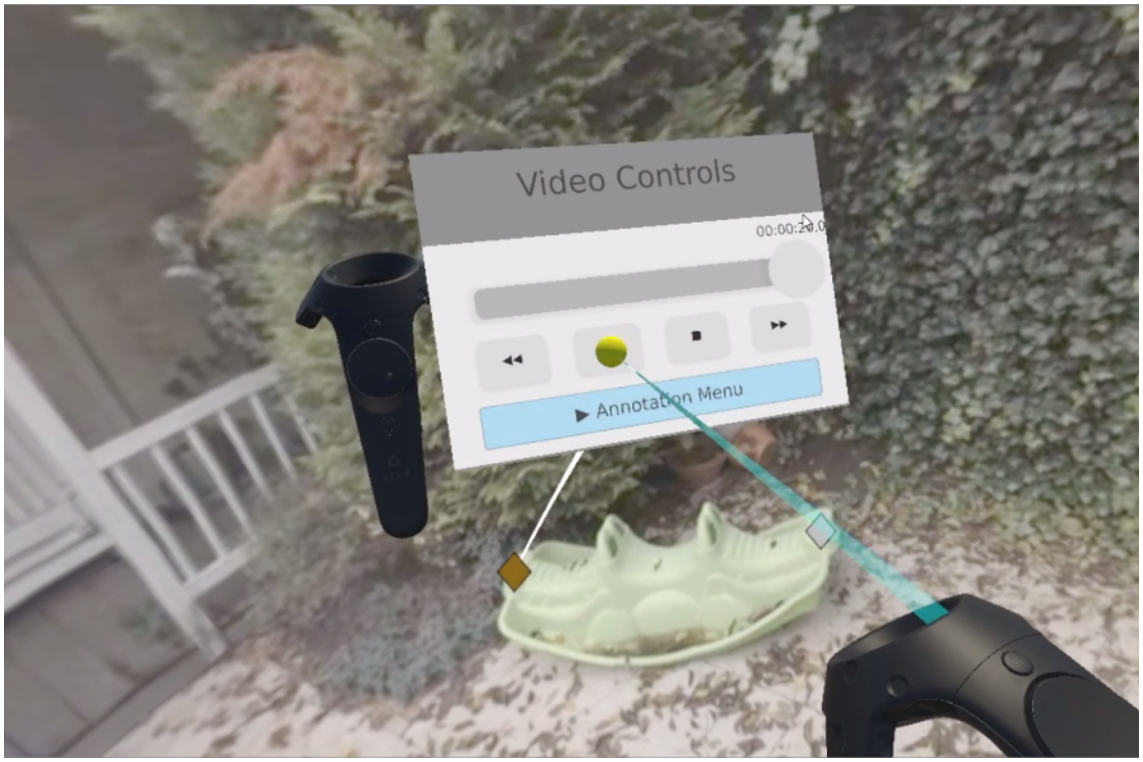


Figure 3.20: Example of Menu operation using the HTC Vive controllers.

Finally, once one wishes to dismiss the menu, one simple needs to click the trackpad button on the left hand controller again. It becomes easy then to show and hide the menu at will, and furthermore its position can be adjusted to facilitate reading and pointing at elements.

3.4.5.3 Menu Operation with the Leap Motion Controller

Once again, the user has two hands in the scene, but this time there are no buttons to conjugate. To allow for easier transition between devices – and because having a movable menu is a major convenience in VR – the same paradigm was implemented. One must now conceive a gesture that allows for the invoking of this menu. Since analogously to the HTC Vive, the Leap Motion implementation uses the right hand for pointing, it is logical that the left hand should also be used for summoning the menu. This is done by turning the left hand so that the palm faces the camera, as seen on figure 3.21. When the angle between the palm's normal vector and the user's

head orientation is less than 30° this interaction is triggered. This allows for a direct translation of the HTC Vive scheme, while retaining the naturalness of hand gestures. Likewise, the menu is attached to the right side of the user's hand, and its position and rotation match that of the palm.



Figure 3.21: Example of Menu operation using the Leap Motion controller.

Despite this, due to the limited range at which the Leap Motion sensor can perform, having both arms extended with hands pointing at each other is not reliable enough for menu operation. Instead, making use of the users' preconceptions regarding the operation of touch-screens, interaction with the menu is made with taps from the right hand's index finger. To press a menu button, set the slider position or drag the scroll view, the user simply takes their right index finger to the desired element, and a selection or drag will be registered.

The problem with this approach is the lack of feedback. Arguably, one could place the menu directly on top of the left hand instead of at the side. This would mean that the user's index finger would touch the palm of their open left hand, effectively providing the necessary feedback. However, this implementation forces one hand to be directly behind the other, creating an occlusion that would make the tracking unacceptably unreliable or even unusable. As such, the attachment to the right side is maintained, but interactable elements such as buttons are made salient like keys on a keyboard. As soon as pressure is detected on one of these elements, they start being depressed until they reach the plane of the menu canvas. Only then is the interaction registered. While this is no substitute for haptic feedback, it helps maintaining the sense of depth, and assists users in understanding the relative depth of their two hands.

To conclude, when users want to dismiss the menu they simply need to turn their left palm away from the camera. Alternatively they can remove their left hand from the scene altogether,

which is possible due to the Leap Motion's limited detection range. This assures that menu operation is most similar to the one achieved with the HTC Vive controllers, but employs instead touch-screen techniques on the right hand side of the interaction.

3.5 Summary

This chapter began with an overview of the demands of the proposed solution, with respect to the research questions, in the form of a requirements analysis. A general eliciting was performed, but constraints regarding the performance were also gathered, despite not being strictly requirements.

The findings and lessons learned from the development of an early prototype in the past were then described. The main focus was to identify key points that should be incorporated, and others who could be improved. Furthermore its study allowed the early prototype to validate the importance of annotations.

Based on these, a contextualisation of the proposed solution was elaborated, within the scope of the requirements. The boundaries of the system were delineated, integrating it into already existing work-flows. Furthermore, a general architectural structure was established, with information flow within the application in mind.

The final section of this chapter is dedicated to interaction and interface design. A model for user interaction within the system is elaborated based on the software structure. Using it, a description of researched techniques in the context of transforming traditional interaction metaphors into VR-compatible paradigms was presented. An implementation-independent analysis of these techniques was performed for each set of input devices, and the considerations on their VR applicability were explained. Further interfaces are developed, firstly with low-fidelity diagrams, then with in-engine specifications.

Chapter 4

System Implementation

The previous chapter detailed the high-level architecture, and in this chapter the implementation of the proposed solution is discussed. The work described in previous chapters provides the basis for the implementation work and researched techniques. The most important objective of this implementation is to provide a framework with which to answer the research questions posed in section 1.3.

The techniques presented below follow a procedural, rather than behavioural approach. This is, they condition the application's response – how it relays the appropriate and necessary information to the user. An example of user engagement is illustrated in figure 4.1. It is easy to see how important the rendering techniques are, as well as the robustness required of the interaction.



Figure 4.1: Usage of the implemented solution.

This chapter is divided into four categories. The first is dedicated to exposing first a set of challenges regarding the playback of 360° video within the Unity engine. The techniques behind

the rendering of informational three-dimensional objects, such as annotations, within an immersive environment are all illustrated in section 4.1.

Subsequently, the fundamentals of annotation and keyframe manipulation and the exposed controls by each of of those objects are explicit summarily in section 4.2. These feature a deduction of chosen interface paradigms.

Section 4.3 describes a technique for accurately calculating and interpolating Catmull-Rom curves on the surface of a sphere. This is the result of mathematical deduction and results in an algorithm that can be used to connect keyframes of the same annotation, and thereby calculate the position of an annotation at any time.

Lastly, section 4.4 presents a set of methods to fulfil data management and storage. The mechanisms employed to process annotation content are described. Additionally, the metadata file protocol is categorised according to existing standards. Lastly, the challenges posed by the tracking and measurement of the system are illustrated, and the solutions used to circumvent them are exposed.

Additionally, a detailed architectural diagram is presented in appendix A, followed by a description of each component. A summary of the interactions between elements is also presented in this appendix.

4.1 Rendering Techniques

In order to fully contemplate the requirements elicited in section 3.1.2, a set of rendering techniques were employed either by modifying existing code, such as shaders, or by researching and implementing new algorithms. Firstly, the facilities for rendering 360° video are explicit in section 4.1.1, including external tools that essentially dictate parts of the low-level architecture in order to provide an abstraction of such mechanisms. Section 4.1.2 discriminates the difference between annotations and regular subtitles and presents the issues that arise during the transformation of such content to 360° space. It presents solutions for these issues and explains them with Unity-specific logic.

Then, section 4.1.3 presents a set of considerations deduced from the necessity of maintaining annotation content on external files, and the subsequent challenges in rendering such content within the application. From these, a mechanism for generating imagery from HTML files is presented.

The rendering of annotations and their keyframes is described in sections 4.1.4 and 4.1.5. Each of these sections presents the unique challenges each object faces, and the steps taken to address them. Furthermore, considerations that must be taken into account when such objects are present in a scene are also explained.

4.1.1 Rendering 360° video

Native video playback is lacking in the Unity engine: firstly, it lacks the flexibility to load the most common high quality video formats – such as *mp4* or *mkv* – at run-time, requiring video files to be

part of the compilation assets; secondly, and perhaps more importantly, it lacks a seeking functionality. This means that the user cannot instantly travel to specific parts of the video, which causes usability to be greatly impacted when annotating multiple points that exist in time simultaneously.

As such, a video decoder plug-in was employed to provide such features. This plug-in is called MediaDecoder[1] and is available at the Unity Asset Store. Furthermore, this plug-in's source code has been recently made available on GitHub[2]. The software makes uses FFMPEG[3] – a framework for manipulating media such as video which provides encoding and decoding solutions through a command-line interface – to stream video from disk, avoiding large load times at start-up. The developers advertise frame rates of above 90 frames-per-second (FPS) and support for resolutions up to 8K. During development videos of standard 1080p resolution were playable at over 200FPS, while videos of 4K resolution and above were met with buffering issues. It is also suggested that this plug-in was designed with VR in mind, and it even provides a specialised spherical mesh with efficient UV mapping of the inside surface, ready for immersive video. This very mesh was used to provide quality 360° video playback, and additionally implement control mechanisms like rewinding and seeking, in tandem with the standard play/pause behaviour.

Recalling figure 2.3, reproduced below in figure 4.2, the user, taking the role of a 3D camera, is placed at the centre of this sphere. The video is rendered onto its internal surface, completely enveloping the viewer. This sphere has a radius which would be translated to between 2 and 3 meters in the real world: rendering the imagery far enough away helps retain the sense of depth, and prevents user height and position from influencing the viewing experience.

One may infer that this integral part of the prototype can double as a 360° video player. While this is true – and one can indeed decouple all interaction from the current prototype with simple tweaks – the deep interaction with the editing components may require more than a simple refactoring if one wishes to keep only the media controls.

4.1.2 Positioning Content in 3D Space

Contrary to classic subtitles which are two-dimensional text boxes within the user's field of view at all times, annotations exist in three-dimensional space in the context of 360° videos – although attached to the inner surface of the sphere at all times. As such, even if one conceives a purely two-dimensional annotation (with the simplest example being a plain image), it is easy to deduce that the additional degrees of freedom in position and rotation will pose a problem if not properly harnessed. An annotation, much like a physical photograph, has a "side". If one turns a photograph on its back, the image is not visible. An annotation is thus completely visible only when it faces the user directly, as seen on figure 4.3.

It is possible to visualise a scenario in which an annotation's position and rotation align it such that readability of the content is severely reduced. A simple and yet effective solution to this

[1] MediaDecoder plug-in page in the Unity Asset Store – https://www.assetstore.unity3d.com/en/?_ga=1.144717936.579493789.1466955373#!/content/63938. Last accessed: 2017-05-30

[2] GitHub page for the MediaDecoder plug-in source code – https://github.com/ViveSoftware/MediaDecoder. Last accessed: 2017-05-30

[3] FFMPEG information page – https://ffmpeg.org/about.html. Last accessed: 2017-05-30

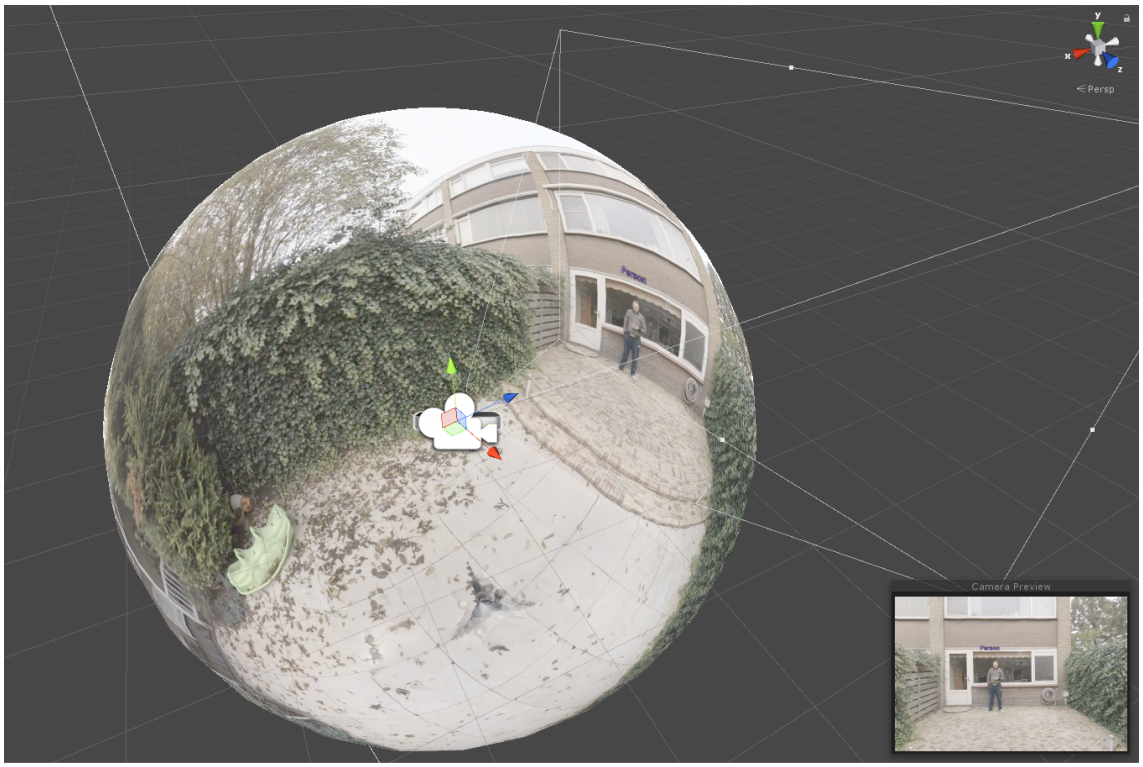


Figure 4.2: 360° video mapped to the inner surface of a sphere in the context of the proposed solution. In it, the viewer is placed at the centre, taking the role of the 3D camera.

problem is to continuously adjust the annotation's rotation so it is always facing the viewer. This technique is known as billboarding, and can be formalised as follows: for an arbitrary object in 3D space there exists a plane that defines such object as "completely visible" only when strictly orthogonal to a viewer's line of sight. Additionally, if one defines the object's *facing* as being the normal vector to this plane, then the object is always perfectly readable if it this vector points towards the viewer. In other words, said object will be completely visible as long as it is facing the user.

One can see then how this condition can be forced so that an annotation object faces the viewer at all times: simply subtract the annotation's position from the viewer's to obtain a vector pointing from the former to the latter. That vector can then be used as the aforementioned plane's normal vector. The translation applied to that plane in order for its normal to be correct is the exact same translation that needs to be applied to the annotation object so that it faces the user correctly, as seen on figure 4.3. In the Unity engine, all objects in the scene contain an instance of a component called a Transform. This component holds a set of vectors and quaternions that define and object's position both within a global frame and a local one, relative to its parent. This means that objects in a Unity scene can rotate locally, which enables the usage of definitions like "forward", "back" or "up", since the direction in which local position axes point also varies with local rotation. In practice, once one places an object in an Unity scene and defines its facing – for example, rotating

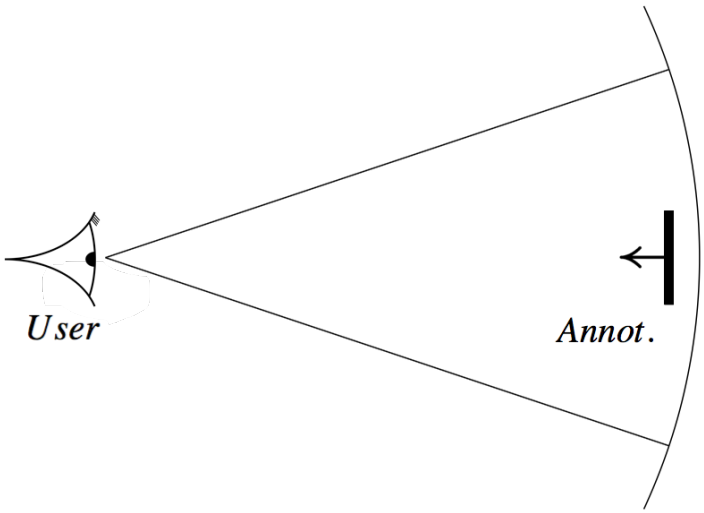


Figure 4.3: Annotation allowing proper viewing.

a cube so that one of its faces is perpendicular to a local position axis – one can then apply local rotations considering that same axis to be "forward". If one were to force this axis to point to the user at all times, then that object would also be facing the user at all times.

4.1.3 Content Authoring

As mentioned, an annotation deviates from a simple subtitle due to the capacity of displaying multimedia content. The most immediate applications are imagery and rich text. While arguably a subtitle could also display a simplistic rendering of rich text, a direct implementation of this paradigm would still not contemplate images. Supposing one had already implemented subtitles, an *ad hoc* approach would be to create a separate component or subclass for images. However, by analysing the coupled programming effort required to achieve true multimedia content one can arrive at the following conclusion: it is ambitious – and cumbersome – to try and predict native text and image component arrangements that the editor would like to employ while retaining enough customisation flexibility. That is to say the author should be able to change between different setups – for instance two lines of text and one image versus one line of text and two images – instantly and with little effort, something that would not be possible if they had to stack image and text components and synchronise them. A more friendly approach is one that unifies the components into a single structure implementation-wise, and gives free rein to the author in terms of content.

One does not have to look far – web pages are exactly this. By separating structure and style one can easily organise different documents into coherent content. As such, if instead of specifying different components for each annotation the editor could simply develop an HTML snippet and have it serve as an annotation, the problems explicit above would be mitigated. Furthermore, with the use of CSS a style guide can be compiled and used across all annotations, and even rich text ceases to be an issue.

This is exactly the followed approach. In terms of authoring, editors are able to create HTML documents in whichever form they like, as long as they are valid. Implementation-wise, one simply has to instantiate a plane onto which a texture is rendered. The two come together thanks to a mechanism that generates static images based on the HTML documents.

This mechanism, called *wkhtmltopdf*[4] or *wkhtmltoimage* is a set of open-source command line tools to render HTML into PDF and/or various image formats using the Qt WebKit rendering engine. For this specific project, only the image generation variant was adequate. A tool-chain of this set called *wkhtmltopng* was used to render HTML documents into PNG images, which the Unity Engine could then import as textures at run-time. This established a workflow that separates the annotation's content from its actual motion in space, making it possible for the authors to edit just the content, or just the path if they so choose.

To invoke these tools, Unity launches a headless command-line interface and issues the conversion of the HTML documents to images silently. This is all transparent to the user as the generated imagery is immediately loaded as a set of textures ready to be used by the application. Any temporary files that may have been generated during this process are also immediately deleted.

Annotations are self-contained objects, capable of instantiating their components from serialised data, and conversely serialising themselves. In the editor environment, they are simply textures as previously stated, but also exhibit a spherical marker on their centre of mass, so as to make it easier to pinpoint where exactly the object is in 3D space. This marker is hidden from the user when the video is playing in order to mimic the experience that a viewer would have.

4.1.4 Rendering Annotations

New challenges arise when considering the possibility space of an annotation's position in a virtual environment. In the context of a 360° video, the viewer is placed at the centre of a sphere, whose radius is virtually between 2 and 3 metres, and inside which the video is projected onto the inner surface. This means that the narrative is taking place at a fixed distance on an opaque spherical surface. Consider then a 3D object whose centre is placed on the surface of this sphere – such an object would be partially occluded as the sphere would clip through it. Similarly, an object completely on the outside of the sphere would be completely occluded, and therefore invisible. The only scenario in which an object would be completely visible is if it were wholly on the inside of the sphere, as seen on figure 4.4.

This poses a problem for annotation readability, especially considering that annotations will rotate locally to match the user's field of view, as previously discussed. This would have the adverse effect of occluding different parts of the annotation depending on the user's head position, which takes control away from the editor. A naïve approach would be to keep the annotation objects inside the sphere at all times. This solution does not account for very large annotations unless the distance from the annotation to the centre of the sphere was dynamically adjusted.

[4] wkhtmltopdf Homepage – https://wkhtmltopdf.org/. Last accessed: 2017-04-14

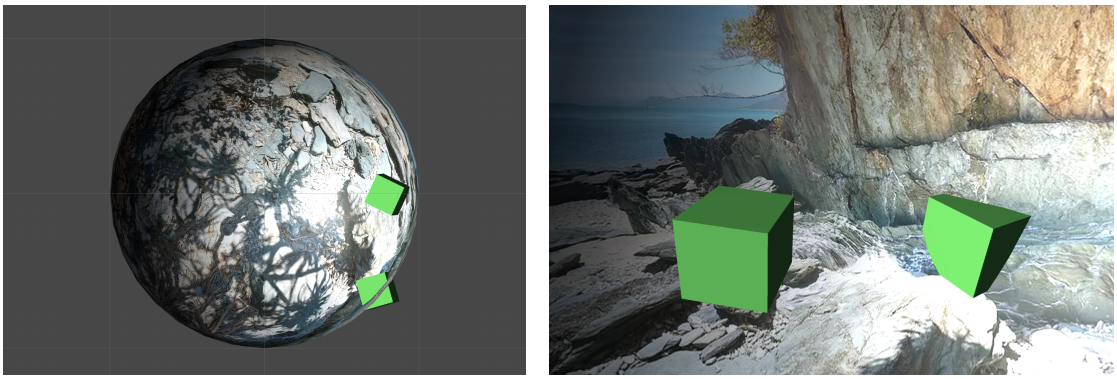


Figure 4.4: Examples of visible and partially occluded objects. On the left, the position of two cubes on a projection sphere of a 360° video. On the right, the same cubes during playback. The rightmost cube is partially occluded for intersecting the sphere.

This in turn would pose two additional problems. Firstly, since the distance to the centre of the sphere is varying, the apparent size of the annotation would also vary, which would make scaling annotations troublesome, as each specific piece would need a custom size or style, rather than an uniform one. The additional problem occurs under circumstances where leaning is supported by the VR device. When the user leans, the virtual camera in the scene is displaced from the sphere's centre, which means that when the viewer looks at an annotation it may not be in the same place as before. Not only would this cause even static annotations to shift, it would potentially cause a misalignment such that an arbitrary annotation would apparently no longer be in the correct place.

A more robust solution is then to remove occlusion culling altogether from the annotation objects. By writing a custom shader which renders even occluded objects, one can make it so that all objects in figure 4.4 are visible. This way it is possible to keep the annotation clamped to the spherical surface, while keeping it completely visible at all times. An example of this can be seen on figure 4.5. This also eliminates the potential problem of leaning, as any angle that the user might try will result in the object apparently being on the same place on the spherical surface.

4.1.5 Rendering Keyframes

In the editor environment, keyframes are identified by a rhombus shape, which takes on the specified colour of the annotation. This colour is never visible to regular viewers, only the producers, as it serves only to identify annotations during editing. To ensure visibility, a shader like the one described in section 4.1.4 was developed and applied. The difference between the two however, is that the keyframe shader allows a customisation of the render queue on a per-object basis. This is to ensure that keyframe objects that are close to one another overlap correctly without tearing through each other, by placing them on different layers of the render queue.

Additionally, the stand-alone nature of these keyframe structures means that it does not make sense for Unity game objects that hold them to be in any way hierarchically tied to the annotation game objects that control them. In practical terms, this means that annotation objects are not



Figure 4.5: Examples of annotation rendering. On the left is an annotation clamped to the spherical surface that, to continue facing the user, is rotated beyond the spherical surface. On the right is the depiction of this annotation as seen by users, completely visible thanks to the shader.

parents of the keyframe objects. Instead they occupy independent levels in the hierarchy, and all instances of keyframes are siblings. This makes navigation and inter-keyframe interactions simpler, as a keyframe can be cloned and still be completely correct, and immediately be used by other annotation without any parenting issues.

4.2 Object Manipulation Techniques

As discussed in the previous section, there are two fundamental synthetic objects that call for user interaction: annotations and keyframes are both the main targets of user focus. This section is dedicated to establishing which control methods each component exposes, especially in the context of VR, when applicable. These are not necessarily mechanisms to control spatial parameters, as they include both convenience methods that manipulate the display and management methods that control the behaviour of the objects more strictly. Implementation details regarding the manipulation of symbolic and 3D objects is also described in Appendix A. For specific menu details, consult sections A.2 and A.4 of this Appendix.

4.2.1 Manipulating Annotations

Considering that an annotation is simply an informational object, direct manipulation of its parameters would not be very efficient since this is already done by keyframes. Instead, only the features that cannot be arranged by manipulating keyframes are isolated and exposed in separate interfaces.

Examples of such interaction are the Hide and Lock methods described in section 3.4.4.2 – they allow the user to render the annotation invisible without removing its keyframes and to prevent all

interaction with it, respectively. The last exposed feature is the Delete method, which removes an annotation object from the scene, erasing all its keyframes. Every other parameter directly related to annotation properties does not need to be exposed, as it is either derived or calculated from keyframe data. One such example is position, that takes advantage of an annotation's keyframe data to interpolate position with respect to time. Further details about the interactions between this component and its properties can be found on section A.1 of Appendix A. All of this is transparent to the users, as they only interact with the keyframes themselves.

4.2.2 Manipulating Keyframes

Within regular video editing applications, keyframes and their properties are presented in separate windows or tables known as dope sheets. Such complicated interfaces do not translate well into VR, since an immersive environment leaves no room for additional windows. This forces all types of interface to be spatial – that is, they exist as objects in the virtual world rather than a typical overlay. Furthermore, since the keyboard may not be available, shortcuts are not a viable means of interaction, terribly reducing the efficiency of such tables.

As such, VR interfaces tend to converge on contextual menus. Rather than cluttering that menu with one button for every possible function, the chosen approach presents a reduced set of features that can be combined to achieve the desired outcome. For instance, given that one can already add and remove keyframes, a "change time" feature becomes redundant as one can simply add a keyframe at the desired time and remove the previous one.

Each keyframe exposes four methods, or manipulation features. Firstly, the Drag method allows one to change a keyframe's position. No additional operation is needed after this, as changing the keyframe's position changes its state, and the annotation it belongs to uses this state constantly in its positional calculations. So any change acted on a keyframe is immediately reflected on the annotation it controls, instantly. Second, a Travel method was implemented. This allows a keyframe to manipulate video time and bring it to its timestamp. It's essential to achieve precise seeking in long keyframe sequences, without having to memorise timestamps. The last two features are the Remove and Add methods. The former simply removes the selected keyframe, leaving the rest. The latter clones the selected keyframe but alters its timestamp to the video's current time, effectively creating a new valid keyframe. Further details on the chain of interactions and propagation of commands can be found in section A.1 of Appendix A. This allows the user to create a keyframe using any other of the existing ones, provided that the video was sought to the desired time.

4.3 Interpolation in Spherical Space

Interpolation is a major asset in the annotation process, since one does not need to explicitly set an object's position at every frame as discussed in the previous section. The type of interpolation one uses, however, will define the user's perception of how objects are supposed to move.

The main use for interpolation is the construction of new data points within the range of a closed set of known points. For instance, given two co-planar points A and B, one can interpolate the position of a third point C that would be coincident with the line segment $\overline{AB}$. For linear interpolation, one could parametrise the position of this point C as:

$$C = A + w(B - A), \quad 0 \leq w \leq 1 \tag{4.1}$$

As such, when $w = 0$, $C \equiv A$; and when $w = 1$, $C \equiv B$.

This type of interpolation is known as linear interpolation, as it produces constant velocity travel in a straight line. However, from equation 4.1 comes that it is possible to interpolate – and therefore animate – a multitude of parameters.

A supplement to this section is provided a description of various interpolation techniques as well as mathematical deductions to support them. This supplement can be found in Appendix B.

In the context of this dissertation, interpolation is used to compute annotation positions based on keyframes. Each keyframe holds a position and timing, and therefore a position between any two keyframes can be computed based on the current video time, t, which is essential since in a video editing environment users will likely scrub the video and seek its time back and forth. An example can be seen on figure 4.6 featuring this computation as perceived by the user.

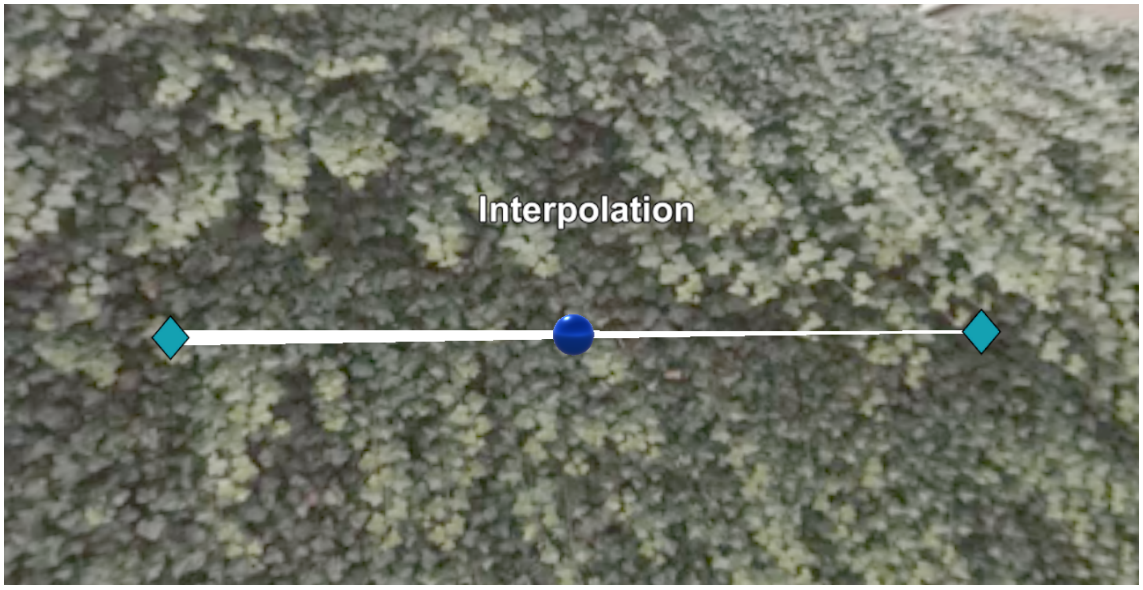


Figure 4.6: Example of annotation interpolation. Annotation position is calculated based on its keyframes, coloured light blue.

4.3.1 Linear Interpolation

Considering figure 4.6, there is no way for the user to know if the white line – indicative of the annotation's path – is indeed an arc on the spherical surface or a straight chord of the sphere. For linear interpolation in its purest form, it would be the latter. While for keyframes in close

proximity this would be a minor problem, if one considers two keyframes on opposite sides of the sphere one can see that linear interpolation would produce a diameter chord. This would mean that objects would travel through the sphere and not exclusively on its surface, which defeats the purpose of annotating.

The alternative is then to transpose this paradigm to spherical space. The process by which this is done is discussed in section B.3 of the Appendix. Important to retain, however, is that assuming there exists a function for linear interpolation called *lerp(A,B,t)*, then there exists an equivalent function for linear interpolation called *slerp(A,B,t)*, which will produce adequate constant-velocity interpolation on a spherical surface.

Using *slerp*, the technique in figure 4.6 is correctly achieved even when keyframes are on opposite sides of the sphere, as the annotation will be travelling across the spherical surface. To illustrate this, figure 4.7 shows a top view of the annotation on figure 4.6 being interpolated on a spherical surface.

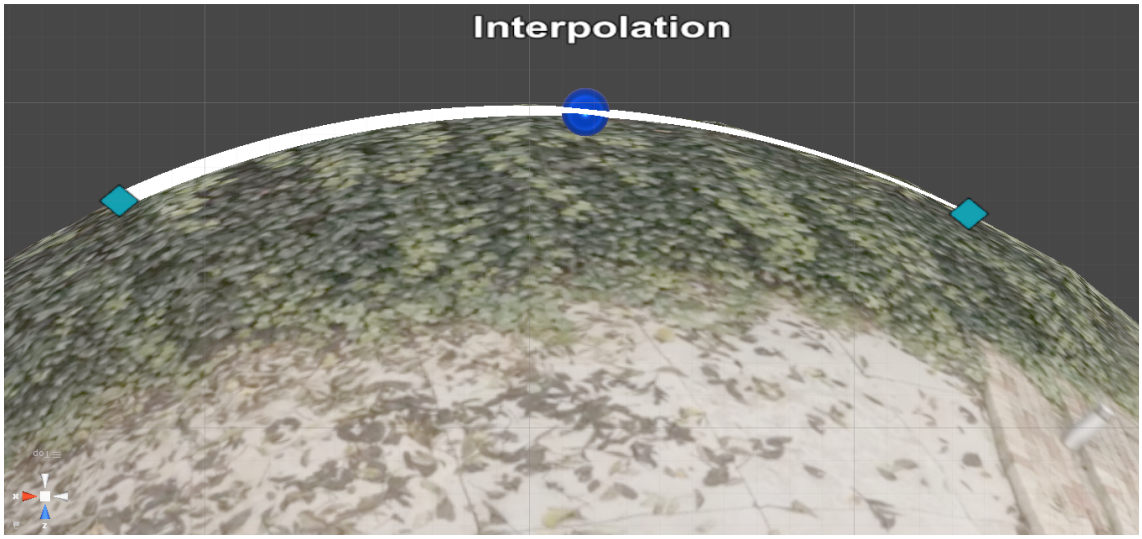


Figure 4.7: Interpolation on a spherical surface. While users at the centre of the sphere see what appears to be a straight line, it is in reality a curve projected on the spherical surface.

To connect multiple keyframes, however, this interpolation must be done in a piece-wise fashion. That is, every two keyframes are considered independently. The desired point can only be between two consecutive keyframes, and as such only those are considered in computations. This piece-wise nature is characteristic of spine interpolation, of which linear interpolation is a particular case.

4.3.2 Spline Interpolation

The linear type of interpolation described above works with pairs of points. As such it creates sharp, ragged paths, and generating curves is unnecessarily cumbersome as one needs to group multiple keyframes in close spatial and temporal proximity.

When one desires smoother curves, or splines, one must drop the linearity between points and replace it with higher order polynomials. Several different types of spline interpolation exist. For some of them, the concept of control points exists. A control point is a point in space whose function is to further define the shape of the curve. The most advanced and flexible spline curves take advantage of this, as is the case of the popular Bézier curve. The disadvantage is that the curve is not guaranteed to pass on these control points, which means that the should be separate from keyframes. In the context of immersive 360° video editing though, interfaces are a limited commodity, and as such attempting to enhance keyframed path-making with control point manipulation tools can be overwhelming for the users.

Ideally, one would use keyframes as control points themselves. The simplest type of spline capable of this is called a natural cubic spline, and is discussed in section B.2 of the aforementioned Appendix. This allows a smooth curve to pass through all keyframes. However, the disadvantage of this spline is that it produces an oscillation when interpolating mostly periodic sets of points that contain outliers. This means that naturally periodic movements like walking would be poorly annotated if the subjects stopped or changed pace, as the path of the annotation would oscillate.

To prevent this oscillation, a type of spline called a Catmull-Rom spline was used. This spline and its interpolation are also discussed in section 4.3.2. This spline is capable of interpolating a point P existing between P_i and P_{i+1} provided that P_i and P_{i+1} are consecutive keyframes and are known, and that their respective neighbours P_{i-1} and P_{i+2} are also known. Of P, only time t_P needs to be known for interpolating the position. This is further detailed in section B.2 of the Appendix.

This type of spline is capable of providing smooth, curved paths that contemplate all the keyframes of an annotation. Furthermore, it allows the annotation to follow the keyframed path with constant velocity. Figure 4.8 shows an example of this type on interpolation, from an user's perspective.

Furthermore, it is important that the employed interpolation maintains a linear behaviour when only two keyframes are involved. An example of this can be seen on figure 4.9, where in the two-keyframe scenario this type of spline behaves as if it was linearly interpolated.

In section B.2 of the Appendix a recursive algorithm is presented to evaluate points on a Catmull-Rom spline based on time. The most prominent feature of this algorithm is that it consists of a set of nested $lerp$ operations, called the de Casteljau's algorithm.

This is, of course for the Euclidean space where points can move freely. When one desires to apply constraints to this space – as will be needed – one can assure proper Catmull-Rom interpolation as long as the target space features a $lerp$ analogue.

It is not enough to simply project the Euclidean space interpolation into the target space. Consider for instance a spherical surface, onto which all interpolation will occur. Consider now a curve that crosses the sphere's centre, and then travels to the outside of the sphere and wraps back in again. This curve is not impossible to generate (by placing two knots at diameter's length), and yet is impossible to project, as the centre point has no analogue on the surface. Furthermore, on paths that are orthogonal to the spherical surface the projection is lossy, as distances travelled on the surface will be much smaller than distances travelled outwards.

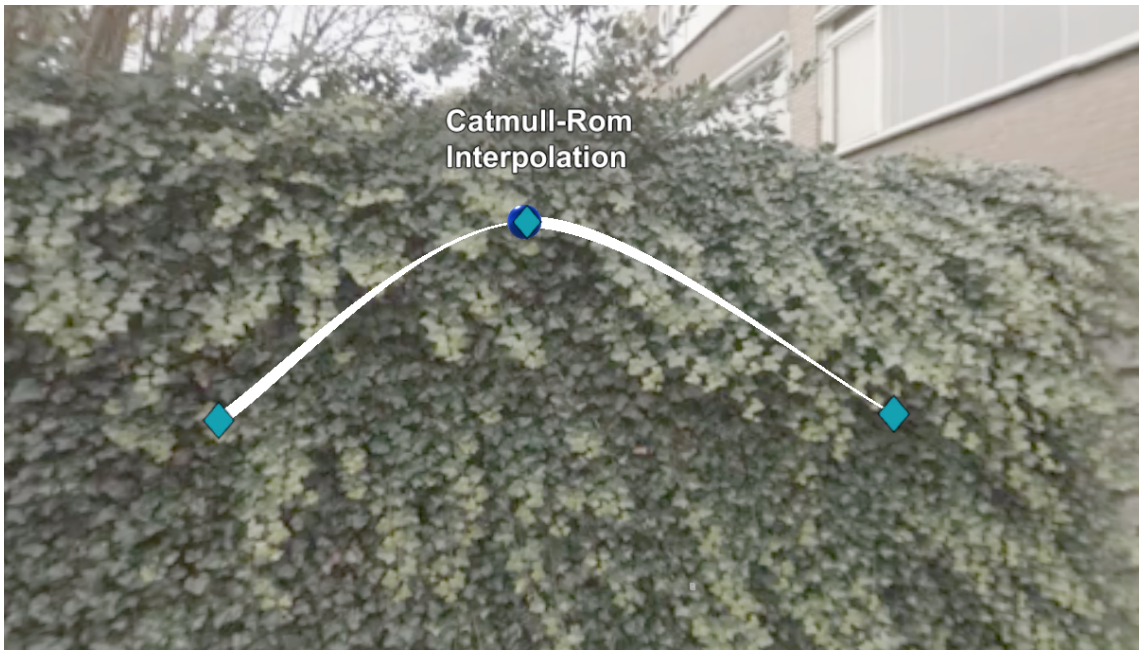


Figure 4.8: Example of Catmull-Rom interpolation with three keyframes. Annotation position is calculated based on its keyframes following a smooth, continuous curved path.

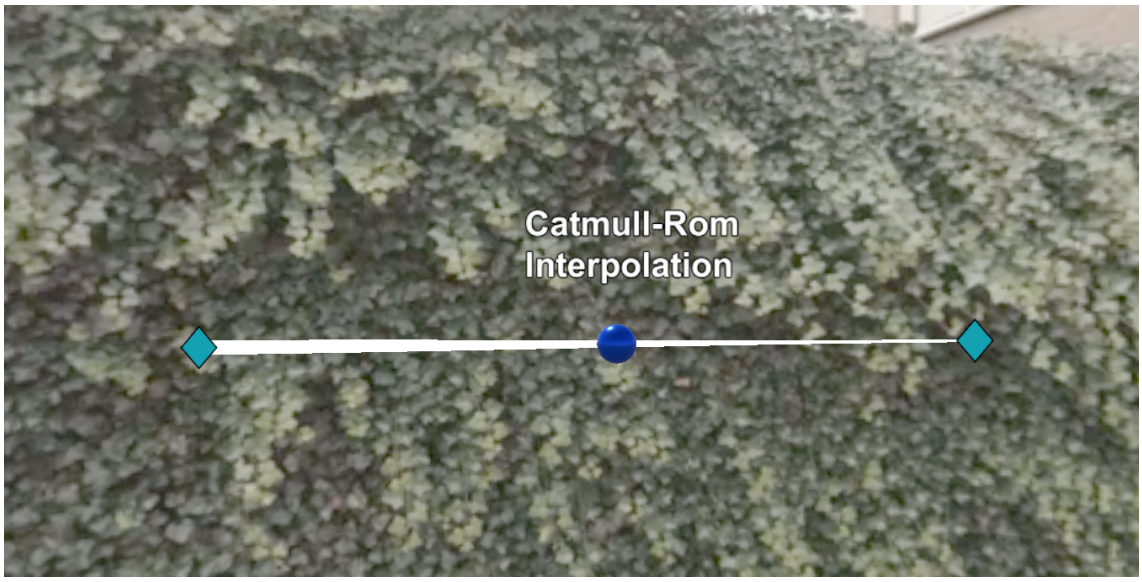


Figure 4.9: Example of Catmull-Rom interpolation with two keyframes. In this case, the annotation exhibits the same behaviour as linear interpolation.

4.3.3 Spherical Spline Interpolation

The proposed solution applies the Catmull-Rom interpolation described above. However, figures 4.8 and 4.9 only show the users' perspective. For these splines to be correct, interpolation must

occur exclusively on the spherical surface. The logical deductions that lead to the following conclusions are presented in section B.3 of the Appendix.

As introduced in section 4.3.2, de Casteljau's algorithm can be translated to a target space provided it has an analogue of *lerp*. As seen on the introductory paragraphs of this section, the spherical surface analogue of *lerp* is *slerp*. The sources and deduction of this assertion are also presented in section B.3 of the Appendix.

Since *slerp* is the analogue for *lerp* on spherical surfaces, one can immediately employ the algorithm described in equation B.13 to obtain Catmull-Rom interpolated splines on a spherical surface. As mentioned, provided one has pinpointed the neighbouring points P_i and P_{i+1} and their respective neighbours P_{i-1} and P_{i+2}, for a point one wishes to find based on t, one can directly apply the following:

$$\begin{aligned} Cr(P_1,P_2,P_3,P_4,t) = slerp(&slerp(slerp(P_1,P_2,t+1),\\ &\quad slerp(P_2,P_3,t),\\ &\quad \frac{t+1}{2}),\\ &slerp(slerp(P_2,P_3,t),\\ &\quad slerp(P_3,P_4,t-1),\\ &\quad \frac{t}{2}),\\ &t) \end{aligned} \tag{4.2}$$

This technique ensures constant velocity sweeping of spherical arcs, while generating smooth curves constrained to a spherical surface. These curves can be further customised by adding keyframes – or knots – and moving them in space and time, allowing for flexible and intuitive path-making.

While completely transparent to the users, this technique can generate accurate interpolation on spherical surfaces. For instance, when one inspects the Unity objects of figure 4.8, the result of that of figure 4.10. One can see that while the users sees an apparently planar curved path, the actual objects are placed on a spherical surface, and the path itself is warped. Nevertheless, it behaves as Euclidean-space Catmull-Rom interpolation would.

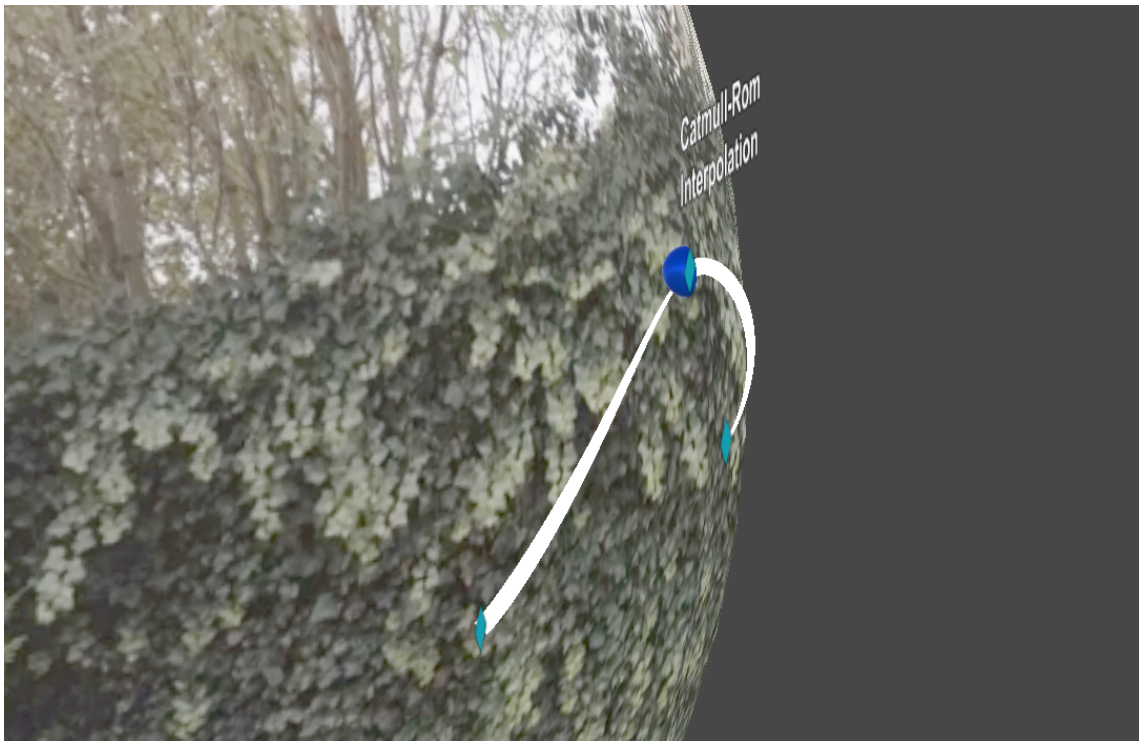


Figure 4.10: Example of Catmull-Rom interpolation seen from outside the sphere.

4.4 Metadata Handling and Persistence

The nature of content editing is rooted in the transformation of data. These data are disseminated during content delivery, thus logically require the tangible support of a file system. Whether it is video content or subtitle and annotation data, the need for persistent storage outside of the system is clear – the proposed solution is not a portable playback system, but the content authored through it should be easily distributed.

After the integrity of the content is assured, an additional file is required – the augmented subtitle file that describes the timing and placement of all annotations in a piece. As an integral part of the solution's life-cycle, one of the system's output is precisely this type of file. Section 4.4.1 presents the selected approach for maintaining a non-destructive modality, and further details file structure and protocol.

Finally, section 4.4.2 presents the techniques used for telemetry gathering and the challenges of ensuring consistent sampling rate and robust file output. Moreover the mechanisms employed to respond to this challenges are deduced and detailed.

4.4.1 Data Storage

Non-destructive editing is a modality of video editing in which the original assets are preserved, and can additionally be accessed in arbitrary order. Like subtitles, annotations are authored on a per-video basis due to their time-dependent nature. Moreover, while different standard video

players may render subtitles differently, the end-result must be similar to ensure a consistent experience. To achieve this, subtitles must be *described* in a way that is agnostic of the player. The potential to customise textual messages naturally gave birth to multiple different formats. Some, such as the Timed Text Markup Language[5] (TTML) allow for extensive manipulation of styles, such as colours, font styles and even the timings of independent words within the same subtitle. Others, such as the SubRip format (also known for its typical file name *.srt*)[6] are simply formatted text files containing only the essential information, such as timings and text.

It is logical that simpler formats find more support among independently developed players. It is substantially easier to parse formatted text on a line-per-line basis than to resolve complex XML nests. Annotations and subtitles differ in the sense that, being static, subtitles can reuse the same container for display, while each annotation requires its own container moving in space. Subtitle description can therefore be made peculiarly simple. Below is a sample of an *.srt* file:

```
682
01:11:17,816 --> 01:11:21,694
- I am the senate.
- Not yet.

683
01:11:26,992 --> 01:11:29,368
It's treason then.
```

Listing 4.1: Example *.srt* file

Each subtitle block is separated by an empty line, and features its start and end timings. While a system this simple is not possible for annotations, there is one thing to take away: a subtitle block for which the timing is `00:00:00,000 --> 00:00:00,00` would never be shown. This can be used to inject data into an *.srt* file that classic players will ignore and that will be transparent to the viewers. In the proposed system, this is the method for extending the *.srt* file format to contain annotation data in a backwards-compatible fashion.

Regarding the data itself, there are multiple ways to describe keyframes. In previous prototypes YAML was used due to its human-readability. However this proved to be inflexible as the C# language supported by Unity must rely on external, community-made plug-ins for it. The plug-in for YAML in specific does not fully support Unity's distribution of C# and required additional maintenance, namely local modifications and compilation. XML was considered at one point for its native implementation, but mark-up languages like it are less human-readable and produce large amounts of overhead text.

While still requiring a plug-in, JSON offers the flexibility needed to be able to modify the architecture. Furthermore, the plug-in is actively maintained and is compatible with Unity. By

[5] Timed Text Markup Language 1 (TTML1) W3C Recommendation – https://www.w3.org/TR/ttaf1-dfxp/. Last accessed: 2017-06-06

[6] SRT Subtitles specification – https://www.matroska.org/technical/specs/subtitles/srt.html. Last accessed: 2017-06-06

injecting JSON messages in the `00:00:00,000 --> 00:00:00,00` block, one can safely extend the *.srt* format to accommodate annotation metadata. Below is an example of this extended *.srt* file.

```
0
00:00:00,000 --> 00:00:00,000
{
    'annotations': [
    {   'content': 'person.html',
        'colour': '#34A8A8',
        'sizeX': 512,
        'sizeY': 512,
        'vertices': [
        {   'time': '00:00:00,000',
            'posX': 0.44753,
            'posY': 0.53417
        },{ 'time': '00:00:06,177',
            'posX': 0.37410,
            'posY': 0.53480
        },{ 'time': '00:00:20,033',
            'posX': 0.47422,
            'posY': 0.53326
        }]
    },{ 'content': 'seesaw.html',
        'colour': '#C69927',
        'sizeX': 512,
        'sizeY': 512,
        'vertices': [
        {   'time': '00:00:00,000',
            'posX': 0.02823,
            'posY': 0.33288
        },{ 'time': '00:00:20,033',
            'posX': 0.08899,
            'posY': 0.43555
        }]
    }

    1
    00:00:02,930 --> 00:00:06,937
    Hello there.

    2
    00:00:07,100 --> 00:00:10,952
    General Kenobi.
```

Listing 4.2: Extended *.srt* file

The schema presented above is the one used in the context of this thesis. Metadata relative to each annotation is kept, where `content` refers to the name of the file containing the annotation's HTML document, `colour` refers to the colour that identifies the annotation both in the menu and by its keyframes, and `sizeX` and `sizeY` define the size of the texture to be generated. The `vertices` array holds an anonymous object for each keyframe, which in turn contains time and normalised positional data – also known as UV coordinates, each is a number between 0 and 1 that represents a proportion of the width or height, for X or Y respectively.

In the current prototype, this file is loaded at start-up and its data parsed. At shut-down a new file is generated for that session, further reinforcing the non-destructive paradigm. The original file is never modified, but future implementations might disregard this requirement with explicit permission of the user.

4.4.2 Tracking and Measurements

There are two basic paradigms when gathering data from a continuous system. Event-driven sampling is the means by which snapshots of the system are taken as a reaction to state changes. An example of this is the gathering of click events, since they are punctual and occur at arbitrary times. Continuous sampling works by discretising the system's evolution state through the gathering of snapshots at fixed intervals. An example of this is the continuous measurement of user head orientation.

Additionally, one can further divide this gathering of data into two modalities: tracking or acquisition is the means by which an event's properties or the internal state's values are acquired; measurement or processing, on the other hand, is a mechanism to obtain further information that is not directly reflected by tracking, but can be obtained for examining other components of the system.

To illustrate this dichotomy, consider the following example: the user misses the click on an object, triggering a click event which should be sampled. The tracking component is responsible for intercepting this event, and extracting from it the cursor position and the target of said click, as well as the system time at which it happened. However this is little information, since the user missed the click, the selected object is incorrect, as it will likely be part of the scene and not a clickable object. The measurement component is then activated to complement this data – using the reported cursor position, the positions of nearby objects are found and their distances to the cursor are computed. The position and distance of the closest clickable element can then be attached to this sample, effectively providing more comprehensive additional data.

The synergy of tracking and measurement is exalted during discrete event gathering, but it can provide useful preliminary data treatment during continuous sampling as well. A particularly useful application of this is the conversion between Unity objects' quaternion-notation rotations and their more trivially interpreted Euler angle translation. With careful implementation, this adds virtually no complexity to the sampling algorithms.

From an implementation perspective, these "logger" or "tracker" classes serve as a stand-alone component for the gathering of data. Their attachment to a game object in the hierarchy allows them access into the game loop, allowing sampling as fast as the game state updates. Furthermore, through navigation of the hierarchy these classes can access information all throughout the system.

An additional consideration when continuously sampling a system is the necessity of maintaining a healthy pipeline. For instance, in a system that is sampled at 200Hz, repeatedly attempting to write data onto a file at this rate would either require a very powerful machine or slow down the system. Alternatively, one can parallelise write-out methods so that data is buffered and written to disk simultaneously with the system's execution. While there is no conceptual drawback to this approach, it would not work in the specific case of the Unity engine due to the way the C# language manages textual data. Being managed code, C# features a garbage collection system for dynamically allocating and deallocating memory automatically. In this language, however, strings are immutable – in order to append data to a string the system creates a new one, de-referencing the former. These unneeded strings are then gathered by the garbage collector for disposal and consequent freeing of memory. This means that dynamically constructing textual data eventually fills the garbage collector's capacity – which, accounting for the volume of data gathered, can easily reach 1GB of memory in one minute – and unscheduled emergency collections will have to be performed. This is an expensive process that greatly slows down the system.

The solution is then to postpone the generation of textual data and instead perform all the sampling on numerical data alone. Special care should be taken to avoid the garbage collection pitfall – even numerical data should be kept in class-wide variables that work as a cache, avoiding unnecessary allocation and de-referencing of temporary variables. The exporting to an external file can then be done at shutdown. To avoid data loss, this process should be wrapped in an event handler that can be assigned to multiple shutdown events – even if the application crashes.

4.5 Summary

This chapter details the followed implementation, which started by stating the challenges of rendering content in immersive 360° VR environments. The solutions for each of the mentioned issues were also detailed. After it, a brief section approached the problem of establishing which interface methods would be exposed to the user.

The following segment of this section described a mathematical technique used to achieve parametric curve interpolation on a spherical surface, and its resulting algorithm.

The last section illustrates the steps taken towards obtaining a self-contained data storage protocol that maintains external files independent from the implementation. The tracking and measurements files are naturally an output of the proposed solution but both the annotation content and description files are independent, and constitute a system that can be replicated and interpreted using other types of software.

Chapter 5

Evaluation

In order to assess the adequacy of the implemented solutions the prototypes were subjected to user studies. These tests consisted of an experiment during which users were invited to perform a series of tasks on all three environments. The gathering of data was divided into two components – data was collected by the system during usage and users were asked to fill out post-experiment surveys. This approach enabled the analysis of mechanical data, such as head and hand movements as well as task performance, coupled with the gathering of perceptual data, such as understanding usability concerns and user preference. While substantially more subjective, the latter allowed the use of a System Usability Scale as established by Brooke et al. (1996), which enables a standardised scoring for each environment.

The first part of this chapter describes the test protocol and collected data, while the second section presents the analysis of these results.

When attempting to answer the research questions presented in section 1.3 using a prototype, one will notice two distinct components: an interaction-oriented solution that users interact with directly, and a perceptual component gathered through questionnaires.

Like the duality present in this definition, so too this chapter is divided. After a brief description of the test protocol and its objectives, the gathered data will be presented and the questionnaires detailed.

User tests were conducted during the course of two weeks using the software described in chapters 3 and 4. A total of 31 people participated in the test, of which 10 were female and 21 male, all voluntary. Most participants were in their twenties, while 7 were under between 13 and 20 years old, and 3 were over 30, one of which worked professionally on journalistic user interfaces. Further details are presented in section 5.1.1.

Spontaneous feedback was also collected both in written and verbal forms, to be discussed in section 5.4.

5.1 Experiment and Results

The experiment was designed around interaction with the platform such that it could facilitate the gathering of two types of data – mechanical telemetry regarding device usage and perceptual user considerations regarding usability and overall satisfaction.

In section 5.1.1 a general outlining behind the motivation for the chosen approach will be described, and its primary objectives will be stated in section 5.1.2. Having exposed these concerns, a protocol was elaborated to provide adequate support for the gathering of desired data. It also describes the environment in terms of hardware and device organisation.

5.1.1 Overview

As stated by the research questions exposed in section 1.3, the implementation of the proposed solution is instrumental in providing a framework for data gathering. While the previous chapters detailed a solution, this one means to support it with data. The experiments conducted and subsequently presented here aim to enable the deduction of answers to these questions, and at the same time incite potential new ones.

Demographic data was gathered in order to understand the general composition of the tester group. It is presented summarily in tables 5.1 through 5.6.

Table 5.1: Gender distribution.

	Participants
Female	10
Male	21

Table 5.2: Age distribution.

	Participants
17 or younger	1
18-20	6
21-29	21
30-39	2
40-49	1

Table 5.3: Previous video editing experience.

	Participants
Yes	22
No	9

Table 5.4: Previous virtual reality experience.

	Participants
Yes	16
No	15

Table 5.5: Education levels.

	Participants
Less than high school	1
High school or equivalent	5
University but no degree	7
Bachelor degree	6
Master degree	12

Table 5.6: Participant handedness.

	Participants
Right-handed	27
Left-handed	3
Ambidextrous	1

Participants were invited to take part spontaneously either via individual contact or through an e-mail message that was sent to all Faculty students with an invitation to the experiment. The response was rather categorical and testing began immediately.

5.1.2 Objectives

The most fundamental goal of this experiment was to answer research questions regarding usability – are exclusively gesture-based virtual environments capable of sustaining video editing applications? Is the current input device technology adequate for such operations? If so, do users manifestly prefer any specific devices?

Despite these, one should not be closed to further questions that may arise during the analysis. User feedback – whether written or verbal – should not incite confirmation bias. That said, user preference is sometimes more important than the actual mechanical specifications of a product.

Firstly, however, comprehensive visualisations of gathered telemetry data will be elaborated and presented. In this section, only information and summary objective explanations will be presented. All the conclusions and any potential extrapolations will be elicited in section 5.4. The objective of this segment is to compare mechanical data by visual inspection, although some statistical analysis was also performed and is presented posteriorly.

The second half of the gathered data was acquired using questionnaires. The principal contribution of this segment is to illustrate user impressions and preference – provided the data was properly treated statistically. Again, any extrapolation one might gather from these data is left to the discussion in section 5.4.

5.1.3 Protocol and Specification

The experiment was carried out on a per-user basis, individually.

Firstly, the context of this thesis was explained, and users underwent a brief exploratory session within the virtual environment using a keyboard and mouse. This was done to explain unfamiliar concepts such as annotations and keyframes in a way that users could interact and experiment with the controls.

After a brief demographic data gathering questionnaire, users who were not acquainted with VR were given the opportunity to experiment the devices in a neutral environment, unrelated to the experiment. This was suggested in order to familiarise users with the devices and head movement.

For the evaluation, each user was exposed to all three environments, one at the time, for a maximum of ten minutes, in an event referred to as a “session”. Each session featured first a training segment within the actual editing virtual environment, so that users could experiment the controls they had explored before. These training sessions lasted no more than five minutes. After this practice segment the measurement session started, in which a set of four tasks — identical across sessions — was presented, to be carried out by the users. The list of tasks was the following:

First Task – T1 To drag a misplaced keyframe of a specific annotation (called "Seesaw" and coloured yellow) to a point in space indicated by a grey unsaturated keyframe icon, as seen on figure 5.1 in a mostly vertical motion.



Figure 5.1: Initial scenario for T1.. The yellow keyframe on the right should be dragged to the grey coloured icon with a vertical motion.

Second Task – T2 To remove a specific annotation (via either the main menu or removal of all individual keyframes) entirely. This annotation is called "Delete Me" and is coloured in light blue, as seen on figure 5.2.

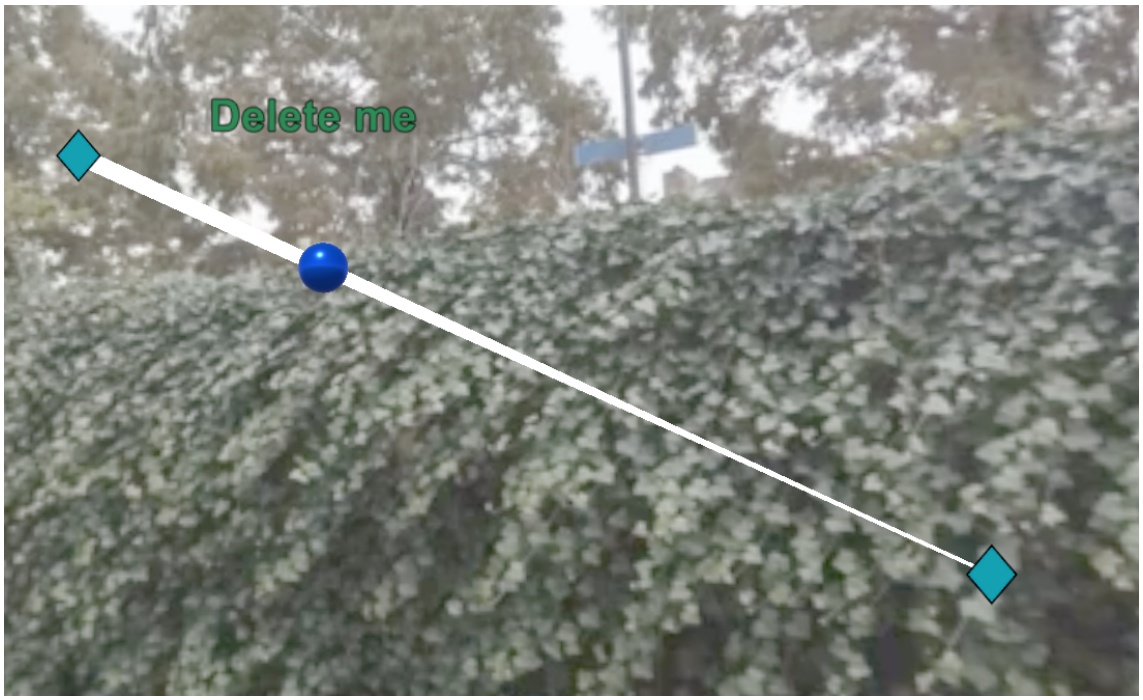


Figure 5.2: Annotation to delete during T2.

Third Task – T3 To create a new annotation and place it anywhere on the scene. While the place is not relevant, this task is completely transversal in that it requires usage of all menus,

generating substantial measurement data. An example of very basic completion is presented in figure 5.3.



Figure 5.3: Example of a new annotation placed during T3.

Final Task – T4 To annotate the path of the only person in the video. This person describes a motion that starts at the centre of a terrace, walks to the left of the viewer, then to the right, and finally back to the centre, as seen on figure 5.4. Three keyframes are already in place, and users need to add at least one more with precise timing to complete this task. They can, however, be more rigorous.



Figure 5.4: Expected scenario upon rigorous completion of T4.

The order by which users were exposed to each environment was deliberately permuted each environment, to mitigate any type of user learning that might have impacted the results. Table 5.7 enumerates the four possible sequences of environments that a user might have been subjected to.

Table 5.7: Permutation of environment order.

First Environment	Second Environment	Third Environment
HTC Vive + Wand-controllers	Oculus Rift + Leap Motion	Desktop + Mouse and Keyboard
HTC Vive + Wand-controllers	Desktop + Mouse and Keyboard	Oculus Rift + Leap Motion
Oculus Rift + Leap Motion	HTC Vive + Wand-controllers	Desktop + Mouse and Keyboard
Oculus Rift + Leap Motion	Desktop + Mouse and Keyboard	HTC Vive + Wand-controllers

This type of permutation is very similar to a Latin Square if one considers both the rows and the columns of the Latin Square matrix. However, table 5.7 as-is is missing a row and a column from this matrix, the ones which would start with the *Desktop + Mouse and Keyboard* as a first environment. These were deliberately left out since users were given a tour of the application on this environment. This tour serves as common ground with which to compare both the first VR experiences, and the lessons learned upon revisiting the desktop environment in later stages of the experiment.

Between each section, users answered a System Usability Scale (SUS as presented by Brooke et al. (1996)) questionnaire regarding the environment they had just experienced. The purpose was to obtain SUS scores for each environment session. At the end a final survey was answered. It presented a set of questions for each device set to which users answered on a Likert scale, with 1 representing complete disagreement and 7 being complete agreement. While the SUS is a well-established survey that allows comparative evaluations – even with existing consumer products – a custom-made Likert scale questionnaire based on works by Farhadi-Niaki et al. (2012, 2013) provides specific information that can put all tested environments in perspective and enable statistically valid conclusions regarding user satisfaction.

As for hardware, the solution and the input/output environments naturally need a machine to run on. The solution was implemented with Unity version 5.4.2, and requires a Windows operating system to run. A very summary description of hardware specifications of each utilised computer is described in table 5.8

Table 5.8: Computer specifications.

Environment	Machine
HTC Vive + Wand-controllers	Intel Core i7 CPU @3.30GHz 32GB RAM 3×NVIDIA GeForce GTX 980 GPU (SLI)
Oculus Rift + Leap Motion	Intel Core i7 CPU @3.60GHz 16GB RAM NVIDIA GeForce GTX 750 Ti GPU
Desktop + Mouse and Keyboard	Intel Core i7 CPU @2.60GHz 8GB RAM NVIDIA GeForce GTX 960M GPU

5.2 Data Gathering and Assessment

During the experiment telemetry was gathered to try and document user behaviour. Most types of data were common to all versions of the software. However, due to the nature of VR devices, the versions of the software which dealt with these types of interfaces allowed for further gathering of data, namely regarding head and hand motion information not present in the Desktop environment. The types of gathered data is presented in table 5.9.

Table 5.9: Globally gathered data.

Data Type		Description
Cursor data		Split into three categories: position of on-screen cursor, position of world-space cursor and normalised texture co-ordinates of the projection of said world-space cursor. This data was sampled continuously.
Continuous timestamps		A value in seconds for elapsed time, recorded with each sample of continuous data gathered.
Discrete data	Selection accuracy	With each triggered selection a boolean value was stored indicating if the user had {de facto} clicked on an element or if they missed.
	Closest target	With each event the type of the closest object was stored, as well as the distance to it. This allows further inspection of user intention.
	Object position	In successful clicks, the initial and final positions of the selected object were stored, providing an indicator for drag operations.
	Selection type	A mechanism to tell "button-down" events from "button-up" events, allowing a temporal delimitation of each interaction.
	Discrete timestamps	A value in seconds for the elapsed time at wchich the event happens. Unlike continuous data timestamps, these are not consistently spaced apart.
VR-only data	Head tracking	The position and rotation of the users' head was continuously sampled throughout the experiment.
	Hand tracking	The position and rotation of each hand was continuously sampled throughout the experiment. The different methods of input require careful interpretation of collected data.

The approach focused on gathering as much data as possible, even if it ended up not being used in its totality. The multi-sensory approach further allowed the cross-referencing of data. For instance, using the target distance and position data from the selection events one can identify instances of T1 and compare user behaviour.

5.2.1 Directing User Gaze

As mentioned, the order of the environments within each session was permuted in an attempt to unbias the results of aspects like task and content learning – since the tasks are uniform across

environments, users naturally complete their last session faster. This meant that different users experienced the video content for the first time in different devices. While the first experience was exclusively in VR devices, users also re-discovered the video when they got to the desktop version. Although it is possible to assess this experience as well, it is not so meaningful to compare "user gaze" when it is done using arrows on a keyboard. Figure 5.5 shows the first frame of the video, projected to a 2D plane. On it, each of the initial annotation paths are marked.Next to each annotation is the name of the task it corresponds to.

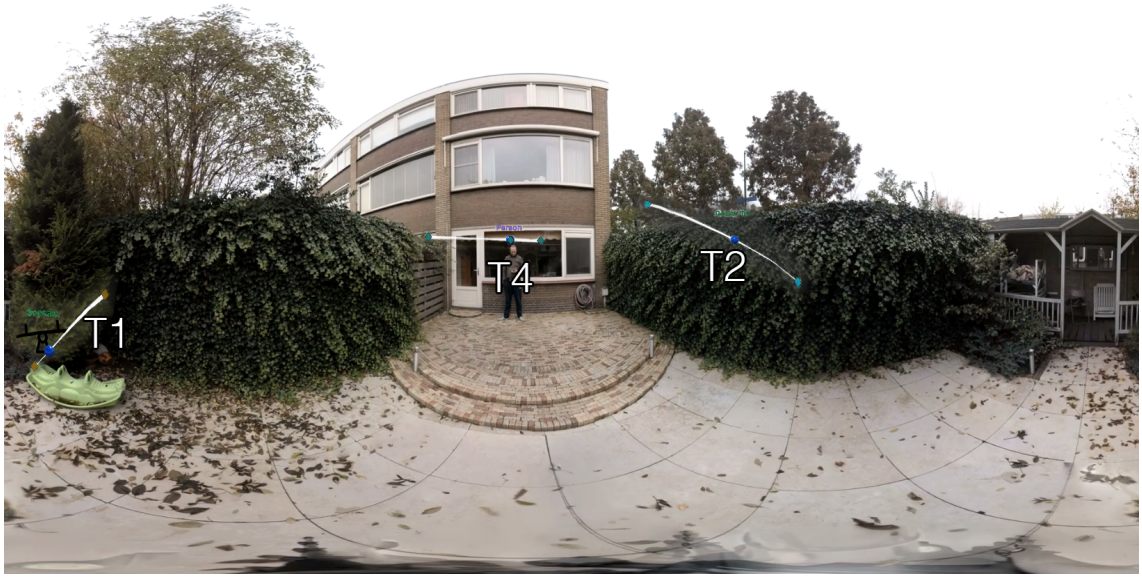


Figure 5.5: Distribution of annotations in the scene. The initial paths of annotations are overlaid on the 2D projection of a video frame. T1, T2 and T4 are also marked on the scene next to the respective annotations.

Figure 5.6 shows a collection of points sampled from arrow key usage during the first twenty seconds of the experience, overlaid on top of the initial frame of the source video. Colours represent density starting with blue for the lowest, going through green and then red for the highest. As before, annotations and task identifier are marked on the scene.

Due to the discrete nature of keyboard usage, data is overly dense around points where users could frame their focus elements. For instance, one does not need to place the camera directly pointing at the person in the video to see it, as long as it is within the screen. This also means that the spots at transitions are more spaced, since users can typically get to one point of the video form another in two or three keystrokes (accounting for adjustments to the direction).

This shows that the main focuses of user gaze are the seesaw, the person and the shed. The first two are annotated points, while the third is where most users chose to place the new annotation in T3. While the person – as a moving target – is naturally more eye-catching, the relatively unremarkable seesaw should not attract as much attention. This further reinforces the purpose of annotations.

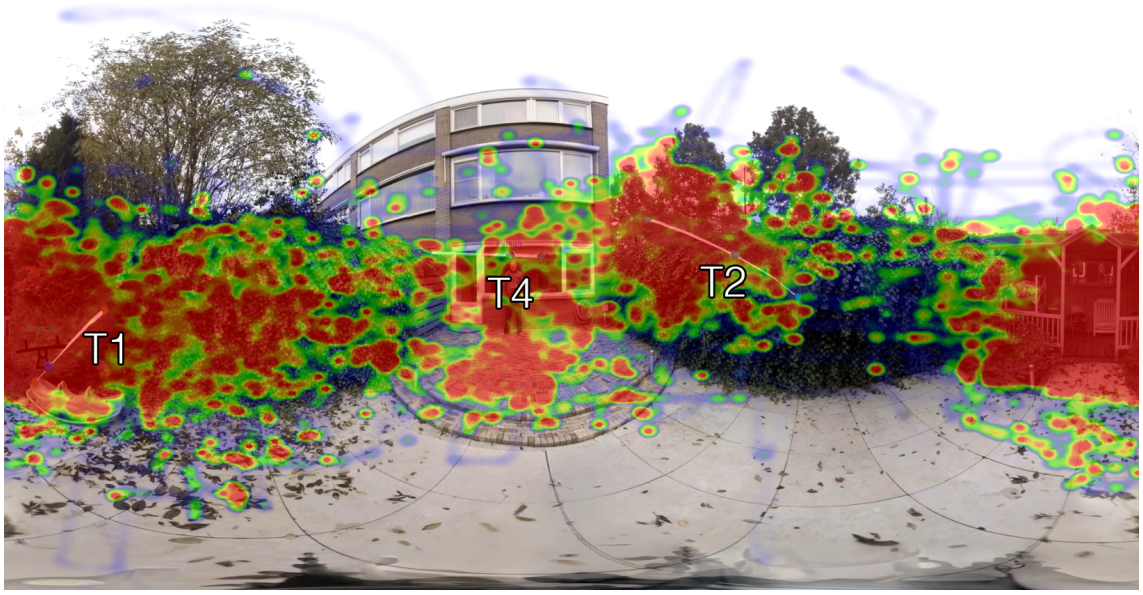


Figure 5.6: Keyboard and mouse user gaze heatmap. Density goes from blue (very sparse) to red (very dense).

Figure 5.7 shows gathered data for Oculus Rift usage which, due to the high computational demands for the actual power of the machine, suffered from lower sampling rates. While the general idea is still readable, transitions are displayed as discrete points.

Like the mouse version, the worth of annotation procedures is demonstrated. During the first twenty seconds of the video, users immediately identify the seesaw as well as the person, being the largest coloured spots. Unlike the mouse, which was featured exclusively as second or third session, the Oculus Rift was half the users' first experience, hence why coloured spot density is more in line with the placement of annotations.

Finally, and because it ran on a higher-end machine, the HTC Vive allowed for more granular sampling. As demonstrated in figure 5.8, density remains focused on the two annotated spots during the first twenty seconds of each session. It is also possible to see that users explored multiple other parts of the video in the vicinity of these annotations.

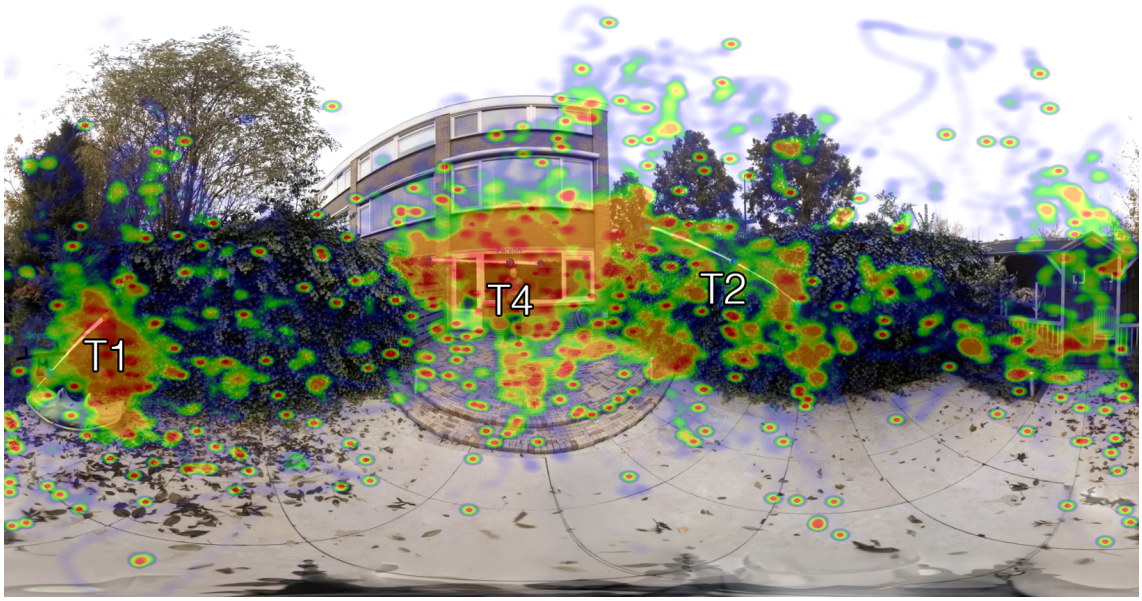


Figure 5.7: Oculus Rift user gaze heatmap. Density goes from blue (very sparse) to red (very dense).

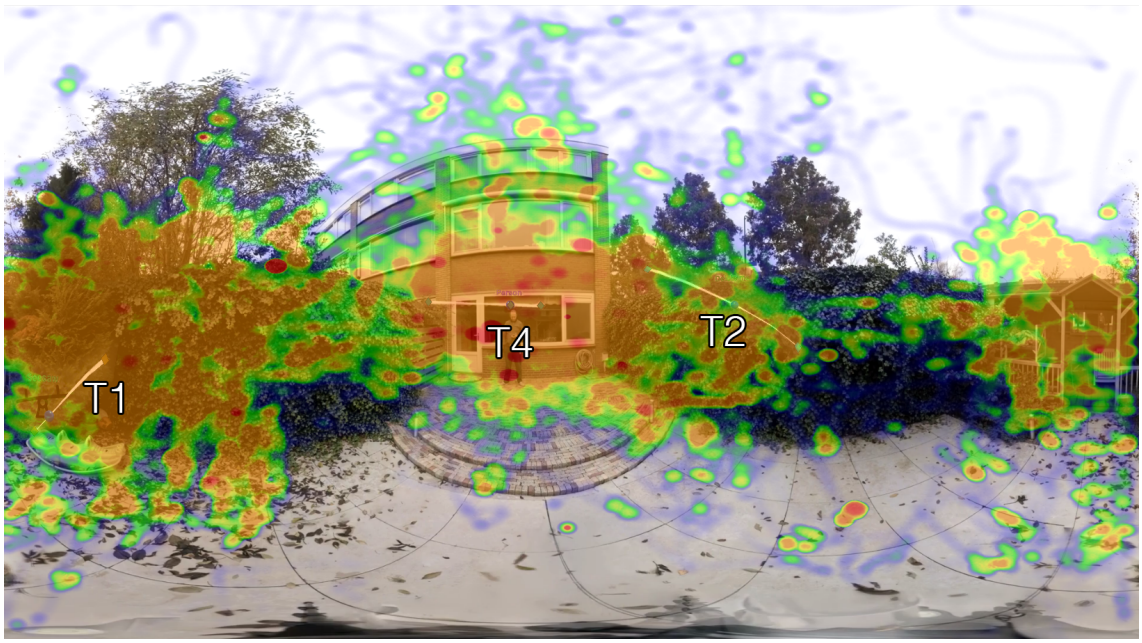


Figure 5.8: HTC Vive user gaze heatmap. Density goes from blue (very sparse) to red (very dense).

In addition to a higher sampling rate, HTC Vive usage shows substantially larger spots, which suggest greater vertical head movements.

5.2.2 Measuring Dragging Precision

Firstly, T1 – in which users had to drag a keyframe from its initial position onto a specific target – was designed to assess precision, both mechanically in terms of device operation and perceptually in terms of user experience. While the latter is reflected only in section 5.3, the former can be expressed in the data.

In this part of the experiment, users had to drag a keyframe object from a known initial position $P_i(x,y,z) = (2.19, -0.60, -1.56)$ to a target position $P_t(x,y,z) = (1.97, -1.57, -1.49)$, an operation that is mostly vertical. As expected, the process of dragging an object from one position of the virtual environment to the other is not linear. When subjecting users to such an operation several significant conclusions can be drawn. Since y axis – which is vertical in Unity engine coordinates – is the one exhibiting greatest variation, it becomes a useful means of visualising convergence behaviour when users are targeting a specific point.

Consider figure 5.9 which plots the position of the y coordinate of the dragged object in T1 for the HTC Vive and wand-controllers. Represented in a solid line in the average of the path followed by all users, and in dashed lines an average of the 10% most outlying measurements for both extrema. This not only shows general trends for all users but it also enables visual representation of common corner cases.

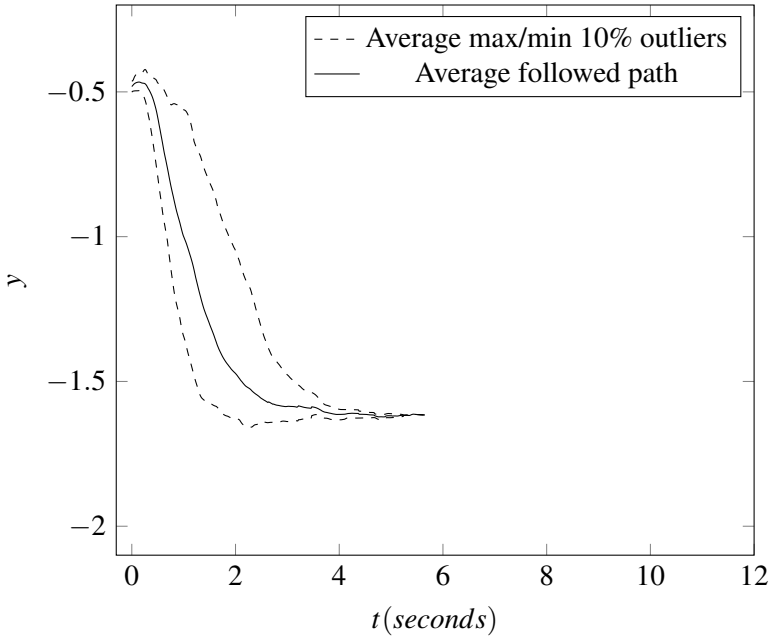


Figure 5.9: HTC Vive and wand-controllers y axis dragging during T1.

It is possible to see the solid line is acceptably smooth, and furthermore that most users are able to place the dragged object within acceptable range of the target point in under four seconds. The relative proximity of the dashed lines indicates a consistent experience, as even these outlying points possess similar trends to the average. The longest time this task took to complete, by any user, was 5.648 seconds.

This information loses part of its purpose if not compared with data obtained for the other input devices. Figure 5.10 shows the data collected for mouse usage, for this same task. Like the previous case, the solid line does not exhibit wild variations and indicates that users reach acceptable proximity to the target point in four to six seconds. In the screen recordings it is also possible to see that people try to be more perfectionist with mouse controls. Additionally, some users reported having trouble with mouse sensitivity.

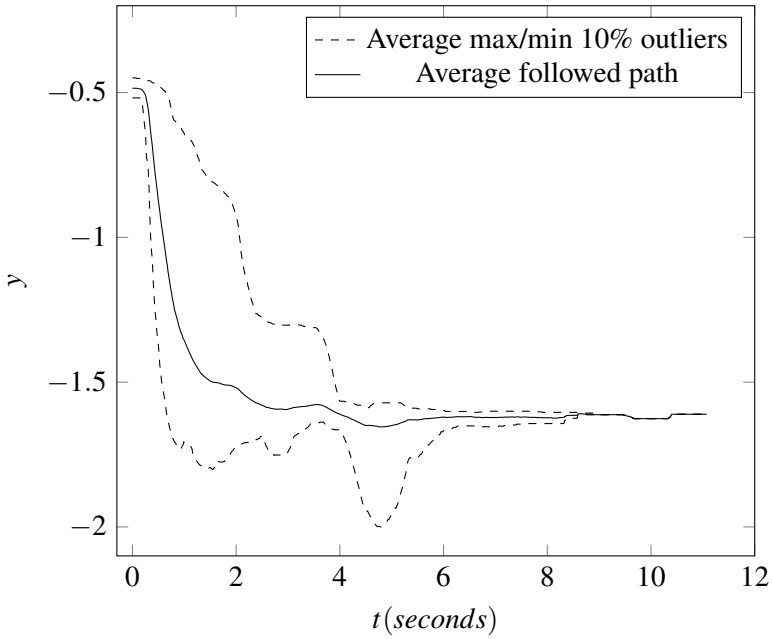


Figure 5.10: Desktop screen plus keyboard and mouse y axis dragging during T1.

The outliers, however, show that this medium is more inconsistent judging by the varying proximity of the dashed lines. Task time was moderately increased as users typically overshot the target position while moving the camera and had to re-adjust their pointing. The longest time this task took to complete on mouse and keyboard was 11.072 seconds.

Lastly, figure 5.11 displays the same information but with Oculus Rift and Leap Motion controller data. Inspection of the solid line shows that users can achieve adequate proximity to the target faster than other devices, in under two seconds. Despite this, users remain in this position for long periods of time due to the difficulty of releasing the object's grasp without moving the hand. This is further evidenced by the protuberance visible after roughly four seconds of exposition, indicating that the release gesture can be detrimental to task speed.

Moreover, the dashed lines indicate that outlying cases are more severely overshot than in the HTC Vive case. While even the bottom dashed line is smooth and consistent with the solid line on the HTC Vive case, with the Oculus Rift and Leap Motion controller it presents a pronounced error, and the same thing happens for the top line.

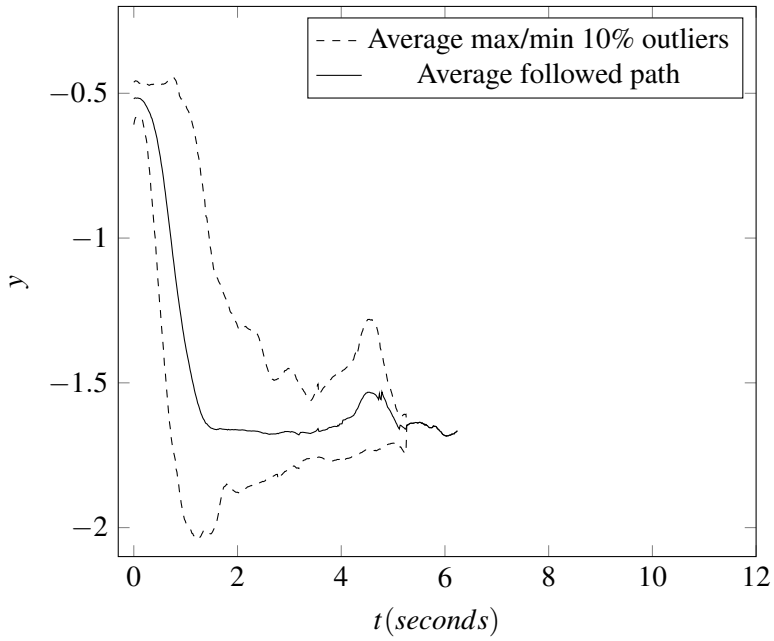


Figure 5.11: Oculus Rift and Leap Motion *y* axis dragging during T1.

5.2.3 Measuring Selection Precision

Another interesting aspect that can be extracted from data across multiple tasks is precision data in terms of pointing, rather than dragging. While during dragging operations one also employs pointing, one also already has the object selected – the dragging operation will not start otherwise. It might be interesting to also measure how often selection is successful, which in turn means that pointing is accurate.

When a user misses a selection it is possible to infer on the intended target based on distance to objects in the vicinity. While not every missed selection is unintentional (e.g. users can click outside any element to reset their current selection), one can safely assume that any selection more than one meter away from any element is intentional and served as a reset.

Table 5.10 shows relative hit and miss ratios for each environment. As mentioned, these were gathered during the course of the experiment, and intentionally missed clicks are not accounted for.

The environment that exhibited superior performance was the HTC Vive, with a miss ratio of under 10%.

Table 5.10: Comparative precision scores for selection actions

Environment	Hit	Miss
HTC Vive + Wand-controllers	90.88%	9.12%
Desktop + Mouse and Keyboard	80.97%	19.03%
Oculus Rift + Leap Motion	48.94%	51.06%

Closely behind, the mouse and keyboard variants achieved a precision ration of over 80%,

something users attributed to high mouse sensitivity but also to a difficulty in selecting 3D objects with a 2D cursor.

The device that showed by far the worst performance precision-wise was the Oculus Rift and Leap Motion controller combination with over 50% of selections being misses. While the disparity is severe, it is possible to demonstrate that it is not consistent throughout the experience. Knowing a user's sequence of clicks – and therefore a sequence of hits and misses – it is possible to compute the average number of errors within a 10-click sliding window. This will provide an overview of user hit-and-miss performance over the course for the experiment.

Figure 5.12 shows exactly this. By computing the 10-point sliding window average for each user it is possible to see that trends are similar – the line shows this measure averaged for all users.

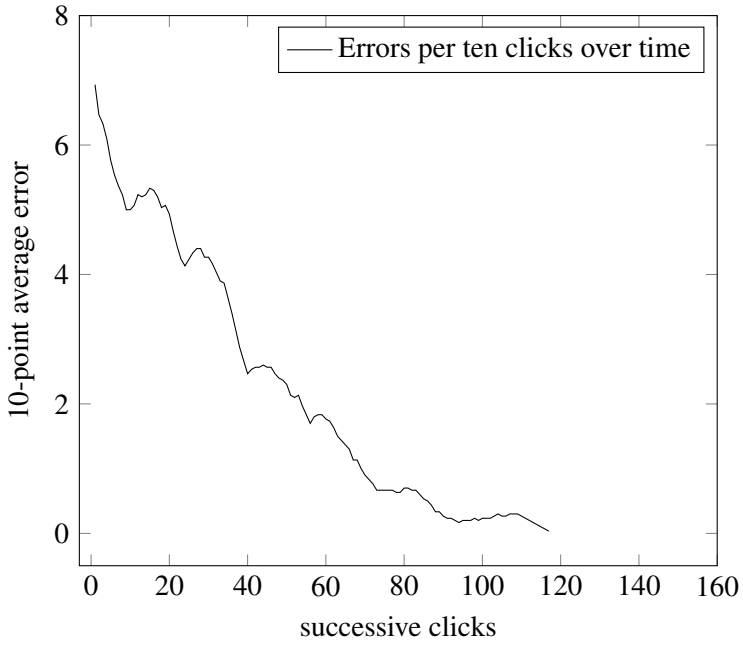


Figure 5.12: Oculus Rift average error per ten clicks over time.

Considering the downwards trend of this average one can infer that users can adapt to the medium quickly. The tendency towards zero in the final stages of the experiment show that however systematic this error may be, it can be corrected by users. Moreover, while some users manifested concerns regarding the precision of the device, a majority of test participants claimed that most errors were due to unfamiliarity, and that they considered that this device would be their favourite if they had more time to adapt.

While this feedback may seem generic it is not without cause when compared to, say, mouse measurements. Figure 5.13 shows the same trend but for mouse and keyboard controls. It is visible that not only is the range of this function significantly smaller, it also exhibits much faster convergence. Interestingly, while the drop to below one error per ten clicks is quick, there are still visible spikes after it, and the convergence slows down until the line barely reaches zero towards the end. The spikes are evidence of task changes, as users tended to click outside elements to reset

the selection before moving on.

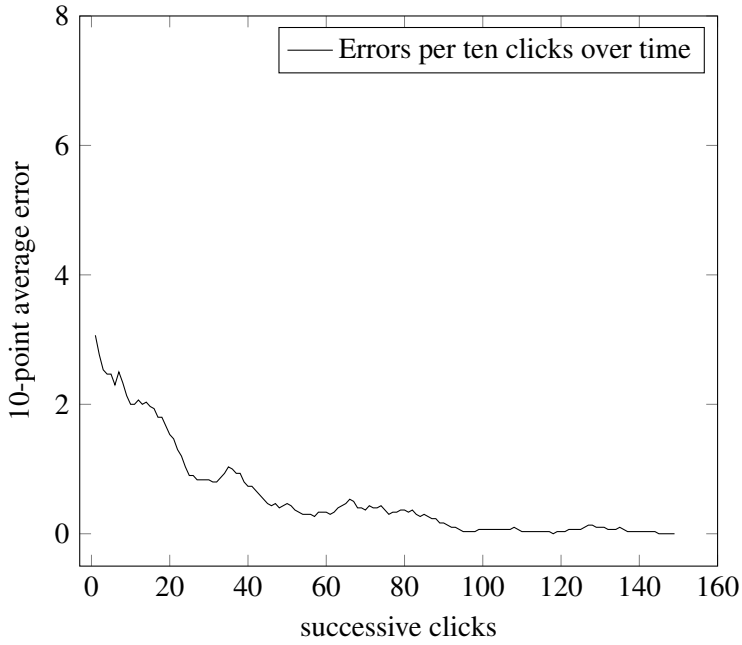


Figure 5.13: Mouse average error per ten clicks over time.

The reluctance of the curve in converging to zero can indicate a more systematic problem that users could not adapt to. Most users spontaneously mentioned mouse sensitivity as a potential issue. However, video evidence shows users clicking objects that feature no interaction, or attempting to drag objects that are not draggable.

The HTC Vive + wand-controllers performance is presented in figure 5.14.

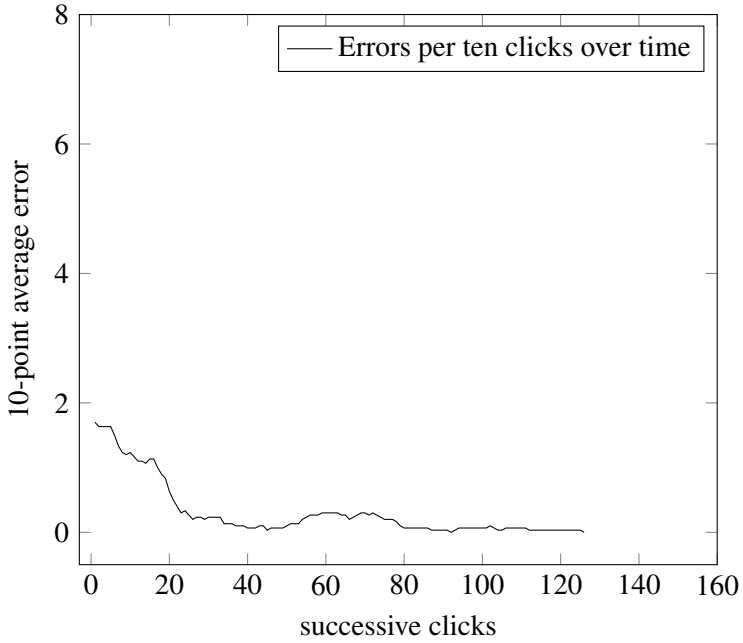


Figure 5.14: HTC Vive average error per ten clicks over time.

In accordance to table 5.10, the HTC Vive + wand-controllers are once again the highest performant devices. Not only does figure 5.14 exhibit a more compact function range, it also presents the fastest convergence. Granted, task changing is still visible in local maxima, but error ratio remains under 1:20.

During the final stages of the experiment, this error smoothly converges to zero, save for a sporadic error. This visualisation is extremely useful because not only does it yield information about user learning on a per-environment basis, it also describes error density over the course of each session.

5.2.4 Measuring Hand Motion

When attempting to compare hand motion across all devices, one much reach the compromise that mouse movement by itself has unfair advantages. Firstly, a resting position on a surface; secondly, operation exclusively on a 2D plane; and third, years of usage and familiarity. It makes sense then, that gesture based, 3D-operating novel interfaces cannot be directly compared to mouse usage. They can, however, be compared to one another. Since both the HTC Vive wand-controllers and the Leap Motion tracked hands are models in 3D space, they can be measured in terms of absolute coordinates. Data was isolated for each task relative to its starting point. The measurements were then averaged and presented below. All units are in metres.

A useful analysis is to compare measurements using a box plot. By aligning the averages one can visualise the range of motion and any tendencies that might arise. Consider, for instance, figure 5.15[1]. The top two boxes describe the vertical (topmost, in green) and horizontal (second

[1] As a brief note, an introduction to whisker-box plots like that of figure 5.15 is presented in appendix C.

from the top, in red) motions of the right HTC Vive wand-controller, while the bottom-most two boxes refer to the same thing, but for the detected Leap Motion hand.

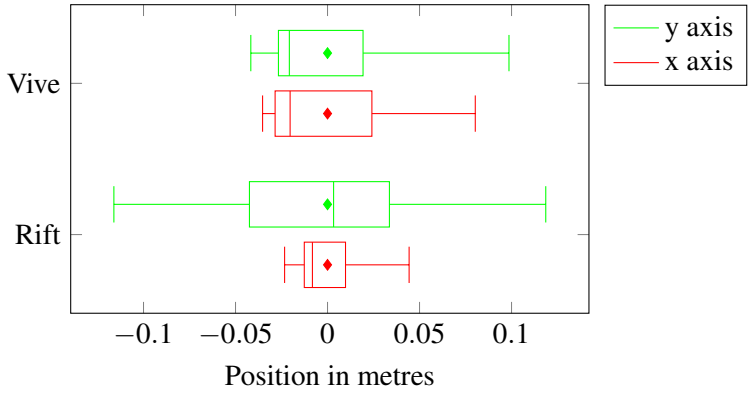


Figure 5.15: Centred right hand horizontal and vertical positions for T4. Vertical deviation from initial position is represented in green, and horizontal deviation in red. Units are in metres.

Is it possible to see that for the HTC Vive, the bulk of measurements is on the negative side, which means that users held the controllers close to their chest most of the time and used minor adjustments to complement pointing with wrist motions.

The Oculus Rift on the other hand exhibits a wider vertical motion, indicating users had to raise their hands above eye level numerous times. Furthermore, while horizontal motion appears to be reduced, it was matched with a complementing head motion, to prevent detection from losing track of the limb.

Figure 5.16 shows the same plot but for the left hand. It is expected that the left hand has reduced mobility in the Oculus Rift scenario, as it only enters the scene to summon the menu and then is removed. However this does not mean it is necessarily resting. On the HTC Vive's left wand-controller, motion is much more widespread.

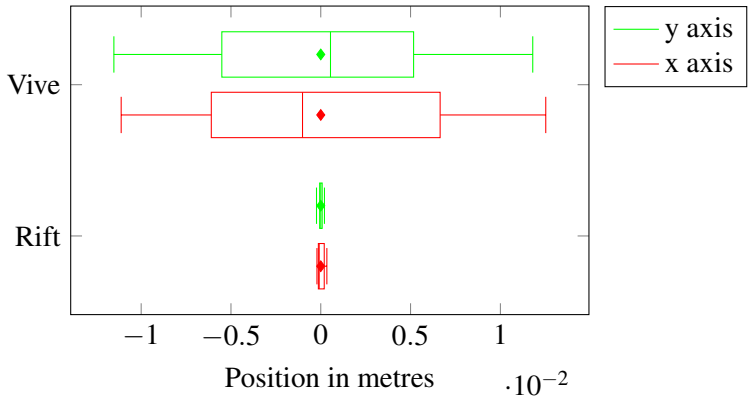


Figure 5.16: Centred left hand horizontal and vertical positions for T4. Vertical deviation from initial position is represented in green, and horizontal deviation in red. Units are in metres.

While vertically the HTC Vive wand-controller remains mostly at chest height it is also often laid to rest on users' legs when not in use. Horizontally it alternates between stretched in front of the user and fully retracted to the side or on the users' lap. The latter can be confirmed with video footage, as controllers can be seen explicitly resting at consistent positions when users are looking down.

5.2.5 Measuring Time Efficiency

Subsequently, to try and assess proper efficiency with regards to time, a one factor analysis of variance (ANOVA) was performed on T1 and T4 task times – the simplest and most complex, respectively. The goal was to test the null hypothesis that there are no differences between all three environments when considering completion time. Due to the arbitrary degree of completion users can achieve – they were the ones determining when a task was done – it would be unexceptional to consider task time by itself. Furthermore, longer completion times feature longer individual task times. As such, to properly compare user performance, a normalised timing was used, where user scores were assigned as a proportion of their individual session length.

When considering T1 users using the HTC Vive achieved marginally faster completion times, with this task occupying 2.41% of total session time on average. The Oculus Rift and mouse versions of this task average at 3.06% and 3.02%, respectively. This is congruent with the mechanical drag precision advantage the HTC Vive exhibits as described in section 5.2.2.

To assess the significance of these differences, a one-way ANOVA was performed on this normalised timing data in order to discriminate if (and not which) environments have any bearing on user performance.

This analysis is based on the null hypothesis that the average normalised completion times for a task are not statistically different. The ANOVA test aims to provide a level of confidence for accepting this null hypothesis, called a p value. Another interpretation of this p value is the likelihood of a false rejection of this null hypothesis. As such, if one is expecting the influence of the environment on task time to be significant, one must reject this null hypothesis. To do so with a confidence level of over 95%, the ANOVA test must yield a p value such that $p < 0.05$. Likewise, for a confidence level of 99%, $p < 0.01$ must be true. In order to obtain this p value, the analysis first extracts an $F_{critical}$ from the ANOVA's F distribution based on the data's degrees of freedom and targeted significance level (α level). For this case, $F_{critical}(2,90) = 3.098$ with $\alpha = 0.05$. Subsequently, an F value is calculated for the test in the context of the specific data. If this value is superior to the $F_{critical}$ value, evidence is strong against the null hypothesis and the aforementioned p value is obtained.

The one-way ANOVA analysis performed on the normalised times for T1 with an α level of 0.05 exhibits an $F(2,90) = 1.063$ and $p > 0.05$ (the actual p value is 0.3496). This indicates weak statistical significance of the results, as the data does not constitute strong enough evidence to reject the null hypothesis. One must then conclude that there is a possibility that performance is no different among environments for simple tasks.

These results are corroborated by the analysis of variance performed on the completion times for T4. Mouse operation proved to be faster, with an average 40.90% of session time, followed by the Oculus Rift with 49.90% and the HTC Vive after it with 65.62%. It should be known however that this task is very flexible with completion criteria, as users can chose to terminate it at any time. The results exhibit varied levels of perfectionism, as some users legitimately completed the task in under 10 seconds, while others took more than 4 minutes.

The F value obtained with this data is $F(2,90) = 1.592$ for an α value of 0.05 as before, and results in $p > 0.05$. There is therefore no evidence to reject the fact that environments do not differ significantly. As these analyses show, performance difference might be negligible among environments but, as described during the course of this chapter, there are mechanical differences to these devices that impact user preference, whether in terms of comfort, fatigue or frustration.

5.3 Usability Questionnaires

The purpose of surveying users is to gather subjective data that might not be directly reflected by mechanical tracking. That is, user impressions and opinions about each of the studied environments. Since questionnaires were an integral part of test sessions, every participant in the experiment also participated in the survey.

Two types of survey were presented to the users. Firstly, one System Usability Scale (SUS) test consisting of ten questions was answered for each environment. This survey achieves a single numeric score for each environment per user, which can then be averaged to find a grade that fits into a grading curve. This essentially provides a normalised rating that can be used to compare the proposed solution to other versions of itself, or to alternative software that underwent the same type of questionnaire.

After all three sessions were complete, a comparative Likert-type scale survey was conducted with hopes of providing a more granular comparative basis, as it allows direct comparison of specific metrics, such as perceived comfort and ease of use. This type of survey is more prone to statistical treatment, enabling conclusions to be drawn on the relative standing of each environment.

5.3.1 System Usability Scale (SUS) Results

Figure 5.17 shows the distribution of SUS scores on a per-environment basis. Despite appearing as values between 0 and 100, these scores are not a percentile rating. It should be noted that the numbers yielded as a score are not absolute ratings either, they are instead meant to be converted to a "grade" based on the distribution of all scores.

With 31 scores per environment one ends up with a data-set of 93 points. To find the appropriate grades, one must arbitrarily select a scale. In this case, a percentile five-rank scale was used: rank A corresponds to the 20% higher scores, rank B to the next 20%, and so forth until rank E, which corresponds to the lowest possible 20% scores. The actual value ranges calculated for this data set are displayed in table 5.11

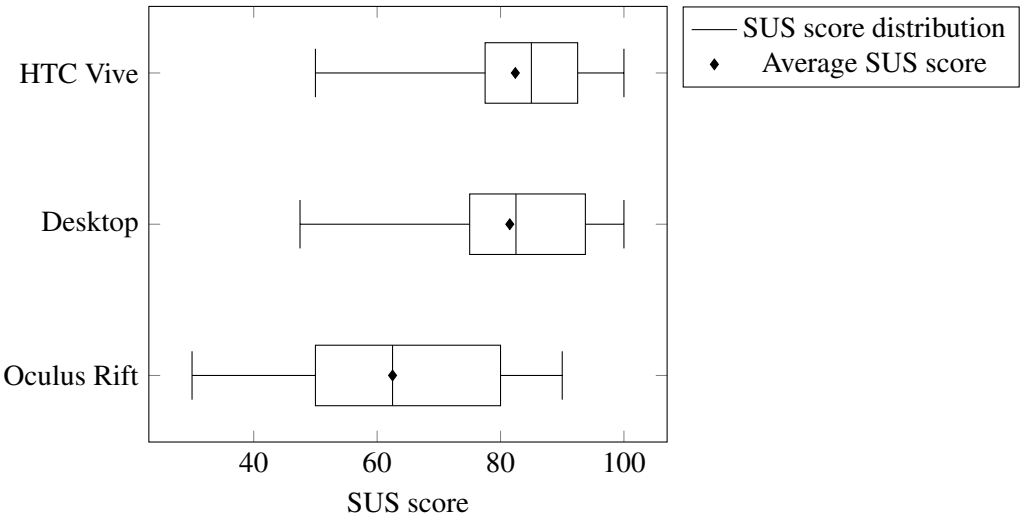


Figure 5.17: Distribution of SUS scores per environment. Despite being a value between 0 and 100, this scale does not reflect a percentage-based rating.

With C being the middle point, this means that any score superior 73.75 is considered above average. Of course, these limits will change if one tries to compare the studied environments to other applications, as the distribution of scores of other contestants will doubtlessly alter this scale.

To grade the obtained data, one can analyse on a superficial level the relative rank of the average scores for each environment. The average SUS scores are summarised in table 5.12.

This ranking shows that the Oculus Rift + Leap Motion environment is the least popular in terms of usability. Without analysing the full breadth of the data, however, it is difficult to know how well these measurements reflect the actual scoring of each environment.

Perhaps the most interesting takeaway is when one attempts to attribute grades to the medians rather than the averages. While the Oculus Rift and Desktop environments suffer no change, the HTC Vive's median score of 85 would grant it the rank of B.

This however, does not mean there is an actual provable difference between the HTC Vive and Desktop environments, as it is simply a difference in criteria. To assess this hypothesis, a Kruskal-Wallis test as defined by Kruskal and Wallis (1952) was performed on the gathered data. This test was performed to assess if the median score of each environment is statistically different from the others. In other words, to assess if the environment dominates the outcome of the scores.

The Kruskal-Wallis test first defines the null hypothesis that all medians are equal across groups. It calculates a critical χ^2 value from the χ^2 distribution based on the data's degrees of freedom (df) and targeted significance level (α level). In this case, with two degrees of freedom, and a desired $\alpha = 0.05$, this critical value is $\chi^2_c = 5.99$. If one had established $\alpha = 0.001$, then $\chi^2_c = 13.816$. If the H statistic yielded by the Kruskal-Wallis test is greater than this value, the null hypothesis defined above can be rejected with a confidence of 95%. It then provides an error value, called the p value, that is the probability of erroneously rejecting this null hypothesis.

The performed test yielded an H statistic of $H = 22.233$, and an error value of $p = 1.487 \times 10^{-5}$. This means that we can reject the hypothesis that the environment has no bearing on SUS

Table 5.11: Possible SUS grades per score

Grade	Score range
A	[95, 100]
B	[85, 95 [
C	[73.75, 85 [
D	[51.25, 73.75 [
E	[30, 51.25 [

scores with a confidence of 99.9% ($H > \chi_c^2$, df=2, N=93, $p < 0.001$). This was to be expected, as the analysis performed so far supports this. However this type of test does not yield information regarding which of the environments is responsible for this difference.

To obtain this information, a post-hoc test must be performed so that Kruskal-Wallis ranks are applied to all environment pairs, effectively establishing a pairwise set of hypotheses to be statistically analysed. The chosen method in this case was the Dunn test – a repeated Kruskal-Wallis test based on an independent null hypothesis for each pair, proposed by Dunn (1964). For each group of two environments, the null hypothesis stands that the medians of SUS scores are not statistically different. The p values gathered for each of these groups are presented in table 5.13. The Dunn test is capable of yielding both adjusted and unadjusted p values: the former are used to resolve intra-group ties that might lead to false acceptance of the null hypothesis, while the latter does not consider ties and is therefore a more conservative estimate. As such, only the unadjusted values are presented in table 5.13.

Using this information it is possible to reject two of the three established null hypotheses. The HTC Vive and Desktop environments do not show statistically meaningful differences, and therefore there is no evidence to reject the null hypothesis that either of these devices have a bearing on SUS scores. They can, because of this, be considered equivalent in terms of usability.

The Oculus Rift + Leap Motion environment, however, is shown to have substantial impact on SUS score. The hypotheses that state that this environment is similar to the HTC Vive and Desktop can both be rejected with 99.9% confidence. Naturally this leads to the conclusion that the Oculus Rift environment is the single most impacting environment in terms of SUS scores. With the ranks retrieved from table 5.12 one can infer that the Rift is the least performant in terms of usability.

5.3.2 Likert-scale Preference Survey Results

Due to the ordinal and discrete nature of a Likert scale, it constitutes a powerful tool to compare scores on a relative basis – i.e. if a score is lower than another – but fails at establishing the distance between points because the numerical value of each point is arbitrary.

Participants were asked to grade their agreement with seven statements on a scale of integers from 1 to 7, with 1 being "Disagree Completely" and 7 being "Agree Completely". The same statements were used for all environments. The statements and methodology are similar to the surveys employed by Farhadi-Niaki et al. (2012, 2013), although the statistical approaches differ slightly. The list of statements is presented below:

Table 5.12: Average SUS scores per environment.

Environment	Average score	Grade
HTC Vive + Wand-controllers	82.4	C
Desktop + Mouse and Keyboard	81.5	C
Oculus Rift + Leap Motion	62.2	D

- It is comfortable
- It feels expensive
- It feels time-consuming
- It is efficient
- It feels natural
- It is fatiguing to use
- It is easy to use

From these one can derive the main metrics to assess. For example, for the first statement, the goal is to assess comfort. For the last statement, one wishes to evaluate perceived ease of use. Furthermore it is possible to divide this set of statements into two groups: "positive" metrics which benefit from higher scores, such as comfort, efficiency, naturalness and ease of use, and "negative" metrics that benefit the overall satisfaction with lower scores, being expensiveness, time consumption and fatigue.

First and foremost, a summary of the results is presented in table 5.14. For each environment, three values are presented on a per-question basis: the first quartile, the median and the third quartile, with hopes of further discriminating the type of distribution these data follow.

One can already see how the HTC Vive environment is pressing the advantage in almost all fields, with the Desktop environment being tied or following close by. The Oculus Rift environment once again proves the be the least popular.

Following the same approach as before, a Kruskal-Wallis test was performed to evaluate if any of the environments had significant impact on responses. During this phase, the goal is to use this rank test to discover if the null hypothesis – stating that the median scores of each statement are equal among environments – can be rejected, on a per-statement basis. As before, with two degrees of freedom, and a desired $\alpha = 0.05$, the χ^2 critical value is $\chi_c^2 = 5.99$. If the H statistic is greater than this, the null hypothesis that environments have no bearing on the results is rejected.

Table 5.15 shows that every metric is controlled in some way by the choice in environment – for each statement, at least one of the environments has significant impact on participant opinion. The fact that $p < 0.001$ is true for every single statement indicates that the null hypothesis that postulates no environment influence can be rejected for every metric with a confidence level of over 99.9%.

Table 5.13: Dunn test results for SUS scores.

Environment group	p value	Significance at 95% confidence
HTC Vive & Desktop	0.6801	Not significant
HTC Vive & Oculus Rift	1.9202×10^{-5}	Very significant
Desktop & Oculus Rift	1.1262×10^{-4}	Significant

Knowing this, the fact remains that a post-hoc analysis is necessary to pinpoint which environments are actually impacting these scores more significantly.

Like before, a Dunn test was performed. This time, this test was conducted for each statement, allowing a discrete analysis of each. The results are presented in table 5.16

The most immediate conclusion, by inspecting the rightmost column, is that the Oculus Rift and HTC Vive environments are different in every metric with the exception of perceived expensiveness. This can be stated with a confidence of over 99.6%. If one does not consider fatigue, this confidence rises to 99.95%. Furthermore, the same conclusions can be drawn from the left column comparing the Desktop and Oculus Rift environments, with the exception of naturalness. This places the Oculus Rift + Leap Motion environment as the most different. This is also supported by the Desktop and HTC Vive environment comparison, as none of the other metrics (with the exception of expensiveness) can support the rejection of the null hypotheses with acceptable confidence. Even naturalness, the most comparable, would not resist a confidence level of 95%.

To further investigate the relative scoring of each environment, given that the pairwise Kruskal-Wallis tests performed by the Dunn analysis revealed that medians are statistically different under some circumstances, an assessment of environments by performance under each metric is presented in section 5.4.

Table 5.14: Metric score medians and quartiles per environment. For each entry, the first quartile is the leftmost value ($Q1$), the median is the middle one (M) and the third quartile the rightmost ($Q3$).

Metric	HTC Vive			Desktop			Oculus Rift		
	$Q1$	M	$Q3$	$Q1$	M	$Q3$	$Q1$	M	$Q3$
Comfort	4	6	7	2	6	7	1	4	5.5
Expensiveness	2	6	6	1	2	3	2	5	6
Time Consumption	1	3	4	1	3	4	2	5	6
Efficiency	5	6	6.5	2	6	7	1	4	5
Naturalness	3	6	6	2	6	6	2	4	5
Fatigue	1	2	3	1	2	3.5	1	4	5
Ease of use	4	6	7	3	6	7	2	4	5

Table 5.15: Kruskal-Wallis test results per metric.

Metric	H statistic	p value	Significance at 95% confidence
Comfort	20.211	4.085×10^{-5}	Very significant
Expensiveness	51.409	6.865×10^{-12}	Very significant
Time Consumption	20.811	3.027×10^{-5}	Very significant
Efficiency	39.728	2.362×10^{-9}	Very significant
Naturalness	18.249	0.000109	Significant
Fatigue	14.546	0.0006939	Significant
Ease of use	34.171	3.801×10^{-8}	Very significant

Table 5.16: Dunn test results per metric. The numeric values indicate the p value for each metric per environment pair. Significant p values for 95% confidence are in bold.

Metric	Desktop & Oculus Rift p value	Desktop & HTC Vive p value	Oculus Rift & HTC Vive p value
Comfort	$\mathbf{6.4298 \times 10^{-5}}$	0.8297	$\mathbf{1.5594 \times 10^{-4}}$
Expensiveness	$\mathbf{7.4555 \times 10^{-9}}$	$\mathbf{5.2347 \times 10^{-11}}$	0.4332
Time Consumption	$\mathbf{1.9443 \times 10^{-5}}$	0.4546	$\mathbf{4.261 \times 10^{-4}}$
Efficiency	$\mathbf{6.5707 \times 10^{-7}}$	0.3864	$\mathbf{5.2239 \times 10^{-9}}$
Naturalness	**0.01898**	0.05501	$\mathbf{2.0011 \times 10^{-5}}$
Fatigue	$\mathbf{3.5547 \times 10^{-4}}$	0.5315	**0.003226**
Ease of use	$\mathbf{1.4519 \times 10^{-5}}$	0.2196	$\mathbf{2.6477 \times 10^{-8}}$

5.4 Discussion and Comparative Analysis

Despite the lack of strong evidence indicating performance differences in terms of completion time, the collected mechanical data presented in section 5.2 indicates measurable behavioural differences between environments. The most immediate is, for example, the selection precision data presented in table 5.10. The Oculus Rift + Leap Motion environment is the only one to exhibit a hit-rate inferior to the miss-rate. This means that on average users had to correct every selection they attempted to make. Whether this comes from participants' lack of skill or systematic sensor error is irrelevant, as the fact remains that this hit-ratio is unacceptably low.

On the same note, the HTC Vive + wand-controllers environment proved to be superior, with miss-rates of under 10%, especially considering nearly half of the participants had never had virtual reality experience before. The mouse performed similarly, considering that several users complained about mouse sensitivity – a parameter that computer users end up customising eventually.

If not for user adaptation, these hit and miss results could be substantially worse, as evidenced by figures 5.12 through 5.14. All participants evidenced this property, consistently adapting to each new device. It is interesting to see how even though the final moments of the session are the most mechanically demanding, they exhibit fewer errors. This is further reinforced by the fact that all users had an adjustment session immediately before the measured test so that they could get used to the devices. The fact that all three experiments started with a significantly larger number of errors may be indicative that user learning requires training. This is because while the grasp on concepts like keyframes and overall application usage was immediately applicable, muscle memory for device usage may not necessarily have been sufficiently developed. It is possible that training and repetition of such tasks would improve user performance during early stages.

Regardless, the smoothness exhibited by dragging performance on the HTC Vive environment, as exhibited in figure 5.9 is in itself a statement to the mechanical advantages of this device set. For instance, mouse usage as seen on figure 5.10 shows that not everyone was capable of such smoothness. A peculiar mechanical side-effect present in mouse and keyboard usage and evident in the crow-stepped shape of the topmost dashed line of figure 5.10 is that users "forget" to move the camera with the keyboard. This means that, unlike the VR versions where user gaze can accompany cursor motions, in the desktop version users have to manually move the camera using the arrow keys. While some users attempted to frame the object and the target before the dragging operation, most users only realised they wouldn't be able to reach the target unless they moved the camera after they initiated the dragging procedure. This resulted in abrupt steps (that often needed to be corrected) because the cursor would follow the camera.

The Oculus Rift is not exempt from this problem either – due to the fact that users' hands have to be directly ahead of their head in order to be detected, more often than not dragging motions are accompanied by head movement. This inevitably confused some participants who also overshot the target while adjusting their perspective. The fast convergence of the solid line in figure 5.11, however, indicates that while it is easier to perceive the sensitivity of the Leap Motion controller

and quickly point at a target, it is harder to command, as it involves both head and hand motions.

This first part of the results has one last very relevant conclusion – it effectively demonstrated how useful dynamic annotations are during first contact with a 360° video piece, as evidenced by figures 5.6 through 5.8. These all show that user focus during the initial seconds of contact with the video is primarily directed at annotated areas. However, an interesting remark is that the HTC Vive environment is the one exhibiting the most vertical movement, especially when equated to the Oculus Rift, since both are VR head-mounted displays. Very few characteristics can explain this – either the HTC Vive's lighter and more comfortable feel lends itself better to exploration, or perhaps the Oculus Rift's proximity to user eyes allows more freedom of eye movement and therefore head motion is unnecessary. However video evidence shows that, due to the position of the Leap Motion sensor at the front of the head in conjunction with user willingness to keep their hands on screen, users end up not using the whole vertical space as it would be substantially more tiring.

Regarding the questionnaire data, several interesting remarks can be gathered, even though in this case the data is more self-explanatory.

There seems to be no clear victor at first sight, but almost all signs point to a loser: the Oculus Rift and Leap Motion controller environment had sub-par performance and below average satisfaction scores. Despite this, it evoked stronger emotional responses in the test participants, some of which verbally manifested that it would be their environment of choice if detection was more precise and consistent.

When comparing the other two environments, however, there is a relevant conclusion to be drawn: perhaps more important than finding which is better is to see if the HTC Vive environment is a suitable translation of a traditional Desktop environment. And indeed, none of the statistical tests performed can reject this – there is no evidence to support the claim that these two environments are different. They display exactly the same median scores for the Likert-scale based questionnaires (except for expensiveness, hardly justifying the comparison), and they have the same C ranking in the performed SUS surveys.

It is plausible then, that the HTC Vive and its wand-controllers are an adequate and satisfactory translation of traditional interaction metaphors to immersive 360° video manipulation in the context of VR.

5.5 Summary

This chapter presented the evaluation of the proposed solution by means of extensive testing and data analysis. After a short introduction, the steps taken to construct the experiments that conduct the aforementioned tests were elicited. Furthermore, the main goals of the expected results and the experimental protocol were described.

The extent of the gathered data was explained subsequently. A string of logical reasoning then began by exposing the results. Firstly, with a data visualisation approach in mind, collected information on user gaze was displayed. It was followed by measurements of device precision on a per-environment basis. After an analysis of error tendencies within the context of the various experiment tasks, information regarding hand motion was exposed and short, objective conclusions were traced upon it. To finalise, a statistical test on collected task times was presented.

After this, the data pertaining to participant surveys was promptly exposed, beginning with a deduction of System Usability Scale ratings, followed by a statistical validation of the impact of each environment on the distribution of scores. The remainder of the questionnaire – dedicated to directly comparing device sets – was then presented and its response analysed. The statistical analysis performed on these data allowed the ordering of each environment with respect to each other.

At the end of the chapter, the aforementioned ordering was elaborated upon, evoking the data presented in both previous sections to conclude the inferiority of the Oculus Rift + Leap Motion environment. Lastly, it was postulated that evidence suggests that the HTC Vive + Wand-controllers environment might be an adequate translation of traditional interaction paradigms to immersive 360° video in VR.

Chapter 6

Conclusions

The main challenge of this work was to study and conceive solutions that first allow the rendering of annotated immersive 360° video content in a VR environment, and then facilitate the editing of such content within a similar context. The proposed solution consists of a video annotation editing system fully contained in a VR environment. Furthermore it boasts compatibility with diverse input and output devices, of which the adequacy for navigating such environments was also studied and documented.

In the context of this dissertation, the research questions presented in 1.3 are split into three groups and tackled individually, all within the scope of the proposed solution. This system is formalised in chapter 3 through a requirement analysis and high-level design. It allows the authoring of 360° video annotation behaviour within an immersive virtual environment. Using this framework one can further the study of this novel content format, whether one approaches it from the playback perspective, or the interaction and usability standpoint. Furthermore, an initial refining of interaction techniques was conducted to enable both the transformation of such techniques for VR environments, and the development of new ones to accommodate expected behaviours not completely predicted by existing interaction.

It was precisely the pursuit of this latter frame of reference that resulted in the implementation presented in chapter 4. First, structural considerations were elicited regarding the rendering and three-dimensional object manipulation capabilities of the proposed solution. A mechanism to achieve correct spline interpolation on spherical surfaces was also described. To finalise this chapter, data storage protocols were defined and documented, as well as tracking mechanisms embedded in the software to aid in data gathering. This alone answers part of the research questions – the compatibility gap between technologies is mitigated with the non-destructive integration of the proposed system in already existing work-flows; the portability is ensured with the development of an open annotation format that can be trivially be interpreted by software.

The prototype itself is a statement to answer interaction concerns. It stands as a proof of concept that traditional techniques can be brought to VR. To support these claims the extent of gathered data is presented and analysed in chapter 5. In it, a description of the conceived experiments is also contained. Moreover, an interpretation of the obtained results is presented and its

implications discussed. These both validate the aforementioned claims that mean to answer the research questions regarding interaction, but also provide a basis with which to argue in favour of the proposed solution's usability. Hand gestures proved to be considered below average, unless paired with controller devices. Nevertheless, the preference for VR remains, and statistical evidence points to the adequacy of implemented techniques.

6.1 Improvements

Different types of improvements can be derived from analysing the development cycle and heeding spontaneous user feedback. For the former, one may look at the dependencies this system requires – i.e. the dependence on external decoders and HTML renderers. While the interfaces with these elements were carefully projected, it is not impossible that they are discontinued. Therefore, an alternative approach could be pursued in which these elements would be a native, integral part of the solution.

Additionally, previously researched techniques in the context of immersive 360° video playback could be incorporated into the editing pipeline in two ways: as guides for editors themselves, making use of such paradigms to streamline their work-flow; or as informational elements, a specific type of dynamic annotation to better guide the users, rather than just inform. Examples of these include arrows that point to annotations, regular subtitles that follow user gaze, and even a radar that accompanies the user in their experience and displays the relative position of annotations.

When one heeds user feedback provided during experiments, one will find useful input, provided it is approached sensibly. A particularly interesting usability suggestion means to solve the ambiguity of adding new annotations to the scene. The current process involves pressing a button from the main menu and a new annotation is automatically placed directly ahead of the user's gaze. The improvement consists of changing this procedure so that, when the button is pressed, users can place the new annotation with the cursor. The pointer (or laser) could change colours to indicate this mode of operation. The problem remains of dismissing or cancelling this procedure. This is something that would have to be furthermore studied.

Another practical improvement tackles the tediousness of placing multiple keyframes for complicated paths. The solution comes from other traditional 2D editing platforms, although not necessarily for video. Many illustration programmes allow the creation of vector paths by drawing them – users draw an arbitrarily complex path using a specific brush, and that path is processed and converted to a Bézier spline. With heavy re-factoring of keyframe creation and manipulation code, this could be made possible, although the correctness of the outcome in uncertain without experimentation. Again, a suggestion that would require further study.

Currently, while the 360° video can have audio, annotations are strictly visual cues, with no sound to accompany them. The addition of auditory components to annotations presents both

a major advancement in terms of content authoring and rich multimedia delivery, and a prolific environment to study immersive audio spatialisation and other related techniques that remain unexplored for VR.

6.2 Future Work

Considering all the suggestions from the previous section, perhaps the latter is the most immediately effective, and that would greatly improve the standing of such a solution in terms of authoring freedom. However, there are other considerations for future work, as follows.

With the massification of 360° video content delivery via the Web, smartphone-based interaction will possibly dominate the VR market. This means that new interaction paradigms will be needed to address the lack of input when the whole device is mounted on the user's head. While this does not pose a problem for 360° video editing – since not even regular editing is done on these devices, but on a dedicated computer instead – the lack of uniform inputs creates a problem for possible interaction with annotated content. This begs a question that is two-fold: firstly, how experience-enhancing is it really to augment annotations with interactive elements? Secondly, how can one trigger interactions when no input devices are available, except for user gaze?

Regardless of the answers to these questions, a Web-based player is an important asset in the advancement of immersive 360° video content delivery. Ensuring that annotated content can be delivered effortlessly and without service constraints (such as the installation of additional software) is essential to make immersive content delivery more investor-friendly.

It is hoped that the interaction techniques and the solution presented in this document inspire future work in this domain. Preferably a profitable one, so that more research and advancement can be fuelled.

References

Deepak Akkil and Poika Isokoski. Gaze Augmentation in Egocentric Video Improves Awareness of Intention. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 1573–1584, New York, New York, USA, 2016. ACM Press. ISBN 9781450333627. doi: 10.1145/2858036.2858127. URL http://dl.acm.org/citation.cfm?doid=2858036.2858127.

Mulloni Alessandro, Andreas Dünser, and Dieter Schmalstieg. Zooming interfaces for augmented reality browsers. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services - MobileHCI '10*, page 161, New York, New York, USA, 2010. ACM Press. ISBN 9781605588353. doi: 10.1145/1851600.1851629. URL http://portal.acm.org/citation.cfm?doid=1851600.1851629.

Steve. Aukstakalnis, David. Blatner, and Stephen F. Roth. *Silicon mirage : the art and science of virtual reality*. Peachpit Press, 1992. ISBN 0938151827.

Phillip J. Barry, Ronald N. Goldman, Phillip J. Barry, and Ronald N. Goldman. A recursive evaluation algorithm for a class of Catmull-Rom splines. *ACM SIGGRAPH Computer Graphics*, 22(4):199–204, aug 1988. ISSN 00978930. doi: 10.1145/378456.378511. URL http://portal.acm.org/citation.cfm?doid=378456.378511.

Douglas A. Bowman. *Interaction techniques for common tasks in immersive virtual environments: design, evaluation, and application*. PhD thesis, 1999.

John Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189 (194):4–7, 1996.

A. Brown, A. Sheikh, M. Evans, and Z. Watson. Directing attention in 360-degree video. In *IBC 2016 Conference*, pages 29 (9 .)–29 (9 .). Institution of Engineering and Technology, 2016. ISBN 978-1-78561-343-2. doi: 10.1049/ibc.2016.0029. URL http://digital-library.theiet.org/content/conferences/10.1049/ibc.2016.0029.

Grigore. Burdea and Philippe. Coiffet. *Virtual reality technology*. J. Wiley-Interscience, 2003. ISBN 0471360899.

Marc Busch, Mario Lorenz, Manfred Tscheligi, Christina Hochleitner, and Trenton Schulz. Being there for real: presence in real and virtual environments and its relation to usability. In *Proceedings of the 8th . . .*, pages 117–126, New York, New York, USA, 2014. ACM Press. ISBN 9781450325424. doi: 10.1145/2639189.2639224. URL http://dl.acm.org/citation.cfm?id=2639224.

Cristiano Carvalheiro, Rui Nóbrega, Hugo da Silva, and Rui Rodrigues. User Redirection and Direct Haptics in Virtual Environments. In *Proceedings of the 2016 ACM on Multimedia Conference - MM '16*, pages 1146–1155, New York, New York, USA, 2016. ACM Press. ISBN

9781450336031. doi: 10.1145/2964284.2964293. URL http://dl.acm.org/citation.cfm?doid=2964284.2964293.

Liwei Chan, Yi-Ling Chen, Chi-Hao Hsieh, Rong-Hao Liang, and Bing-Yu Chen. CyclopsRing: Enabling Whole-Hand and Context-Aware Interactions Through a Fisheye Ring. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, pages 549–556, New York, New York, USA, 2015. ACM Press. ISBN 9781450337793. doi: 10.1145/2807442.2807450. URL http://dl.acm.org/citation.cfm?doid=2807442.2807450.

Ricardo Cruz and Ricardo Fernandes. Journalism in Virtual Worlds. *Journal For Virtual Worlds Research*, 4(1), 2011. ISSN 1941-8477. doi: 10.4101/JVWR.V4I1.2110.

M. Csikszentmihalyi. *Beyond Boredom and Anxiety*. The Jossey-Bass behavioral science series. Jossey-Bass Publishers, 1975. ISBN 9780875892610. URL https://books.google.pt/books?id=afdGAAAAMAAJ.

Olive Jean Dunn. Multiple comparisons using rank sums. *Technometrics*, 6(3):241–252, 1964. doi: 10.1080/00401706.1964.10490181. URL http://www.tandfonline.com/doi/abs/10.1080/00401706.1964.10490181.

Omid Fakourfar, Kevin Ta, Richard Tang, Scott Bateman, and Anthony Tang. Stabilized Annotations for Mobile Remote Assistance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 1548–1560, New York, New York, USA, 2016. ACM Press. ISBN 9781450333627. doi: 10.1145/2858036.2858171. URL http://dl.acm.org/citation.cfm?doid=2858036.2858171.

Farzin Farhadi-Niaki, Reza GhasemAghaei, and Ali Arya. Empirical study of a vision-based depth-sensitive human-computer interaction system. In *Proceedings of the 10th asia pacific conference on Computer human interaction - APCHI '12*, page 101, New York, New York, USA, 2012. ACM Press. ISBN 9781450314961. doi: 10.1145/2350046.2350070. URL http://dl.acm.org/citation.cfm?doid=2350046.2350070.

Farzin Farhadi-Niaki, S. Ali Etemad, and Ali Arya. Design and Usability Analysis of Gesture-Based Control for Common Desktop Tasks. pages 215–224. Springer, Berlin, Heidelberg, 2013. doi: 10.1007/978-3-642-39330-3_23. URL http://link.springer.com/10.1007/978-3-642-39330-3{_}23.

T. Fautier. VR video ecosystem for live distribution. In *IBC 2016 Conference*, pages 28 (13 .)–28 (13 .). Institution of Engineering and Technology, 2016. ISBN 978-1-78561-343-2. doi: 10.1049/ibc.2016.0028. URL http://digital-library.theiet.org/content/conferences/10.1049/ibc.2016.0028.

Feng Zhou, Henry Been-Lirn Duh, and Mark Billinghurst. Trends in augmented reality tracking, interaction and display: A review of ten years of ISMAR. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 193–202. IEEE, sep 2008. ISBN 978-1-4244-2840-3. doi: 10.1109/ISMAR.2008.4637362. URL http://ieeexplore.ieee.org/document/4637362/.

David Fofi, Tadeusz Sliwa, and Yvon Voisin. A comparative survey on invisible structured light. page 90. International Society for Optics and Photonics, may 2004. doi: 10.1117/12.525369. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?doi=10.1117/12.525369.

James D. Foley. A standard computer graphics subroutine package. *Computers & Structures*, 10 (1):141–147, 1979. ISSN 00457949. doi: 10.1016/0045-7949(79)90081-6.

Mark Giles and B. Kerry. *Cyberbase SU2000 Technical Manual*. 1994. URL http://arianchen.de/su2000/files/su2000tech.pdf.

Raphael Grasset, Tobias Langlotz, Denis Kalkofen, Markus Tatzgern, and Dieter Schmalstieg. Image-driven view management for augmented reality browsers. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 177–186. IEEE, nov 2012. ISBN 978-1-4673-4662-7. doi: 10.1109/ISMAR.2012.6402555. URL http://ieeexplore.ieee.org/document/6402555/.

Sandra G. Hart and Lowell E. Staveland. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. *Advances in Psychology*, 52:139–183, 1988. ISSN 01664115. doi: 10.1016/S0166-4115(08)62386-9.

Morton L. Heilig. Sensorama simulator, August 28 1962. URL https://www.google.com/patents/US3050870. US Patent 3,050,870.

Rorik Henrikson, Bruno Araujo, Fanny Chevalier, Karan Singh, and Ravin Balakrishnan. Multi-Device Storyboards for Cinematic Narratives in VR. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 787–796, 2016. doi: 10.1145/2984511.2984539.

Dun-Yu Hsiao, Min Sun, Christy Ballweber, Seth Cooper, and Zoran Popović. Proactive Sensing for Improving Hand Pose Estimation. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 2348–2352, New York, New York, USA, 2016. ACM Press. ISBN 9781450333627. doi: 10.1145/2858036.2858587. URL http://dl.acm.org/citation.cfm?doid=2858036.2858587.

Izabelle Janzen, Vasanth K. Rajendran, and Kellogg S. Booth. Modeling the Impact of Depth on Pointing Performance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 188–199, New York, New York, USA, 2016. ACM Press. ISBN 9781450333627. doi: 10.1145/2858036.2858244. URL http://dl.acm.org/citation.cfm?doid=2858036.2858244.

Kourosh Khoshelham and Sander Oude Elberink. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. *Sensors*, 12(12):1437–1454, feb 2012. ISSN 1424-8220. doi: 10.3390/s120201437. URL http://www.mdpi.com/1424-8220/12/2/1437/.

Yongkwan Kim and Seok-Hyung Bae. SketchingWithHands: 3D Sketching Handheld Products with First-Person Hand Posture. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 797–808, 2016. doi: 10.1145/2984511.2984567.

Andreas Kolb, Erhardt Barth, Reinhard Koch, and Rasmus Larsen. Time-of-Flight Sensors in Computer Graphics.

Max Krichenbauer, Goshiro Yamamoto, Takafumi Taketomi, Christian Sandor, and Hirokazu Kato. Towards Augmented Reality User Interfaces in 3D Media Production.

William H. Kruskal and W. Allen Wallis. Use of ranks in one-criterion variance analysis. *Journal of the American Statistical Association*, 47(260):583–621, 1952. doi: 10.1080/01621459.1952.10483441. URL http://www.tandfonline.com/doi/abs/10.1080/01621459.1952.10483441.

Jihyun Lee, Byungmoon Kim, Bongwon Suh, and Eunyee Koh. Exploring the Front Touch Interface for Virtual Reality Headsets. aug 2016. URL http://arxiv.org/abs/1608.00447.

Huiying Liu, Shuqiang Jiang, Qingming Huang, and Changsheng Xu. A generic virtual content insertion system based on visual attention analysis. In *Proceeding of the 16th ACM international conference on Multimedia - MM '08*, page 379, New York, New York, USA, 2008. ACM Press. ISBN 9781605583037. doi: 10.1145/1459359.1459410. URL http://portal.acm.org/citation.cfm?doid=1459359.1459410.

Filip Malawski. Applying Hand Gesture Recognition with Time-of-Flight Camera for 3D Medical Data Analysis.

João Pedro de Oliveira Marques. Realidade Virtual e Jornalismo Imersivo: anotação dinâmica de peças noticiosas em vídeo 360º. Master's thesis, Faculdade de Engenharia da Universidade do Porto, 2016.

João Meira, João Marques, João Jacob, Rui Nóbrega, Rui Rodrigues, António Coelho, and A. Augusto de Sousa. Video Annotation for Immersive Journalism Using Masking Techniques. In *2016 23º Encontro Português de Computação Gráfica e Interação (EPCGI)*, pages 1–7. IEEE, Nov 2016. ISBN 978-1-5090-5387-2. doi: 10.1109/EPCGI.2016.7851189. URL http://ieeexplore.ieee.org/document/7851189/.

Paul Milgram and Fumio Kishino. A Taxonomy of Mixed Reality Visual Displays. *IEICE TRANSACTIONS on Information and Systems*, E77-D(12):1321–1329, 1994. ISSN 0916-8532.

Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. Telemanipulator and Telepresence Technologies, pages 282–292. International Society for Optics and Photonics, dec 1995. doi: 10.1117/12.197321. URL http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=981543.

Jens Müller, Roman Rädle, and Harald Reiterer. Virtual Objects as Spatial Cues in Collaborative Mixed Reality Environments. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI '16*, pages 1245–1249, New York, New York, USA, 2016. ACM Press. ISBN 9781450333627. doi: 10.1145/2858036.2858043. URL http://dl.acm.org/citation.cfm?doid=2858036.2858043.

Mathieu Nancel, Daniel Vogel, Bruno De Araujo, Ricardo Jota, and Géry Casiez. Next-Point Prediction Metrics for Perceived Spatial Errors. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology - UIST '16*, pages 271–285, New York, New York, USA, 2016. ACM Press. ISBN 9781450341899. doi: 10.1145/2984511.2984590. URL http://dl.acm.org/citation.cfm?doid=2984511.2984590.

Luís A. R. Neng and Teresa Chambel. Get around 360° hypervideo. In *Proceedings of the 14th International Academic MindTrek Conference on Envisioning Future Media Environments - MindTrek '10*, page 119, New York, New York, USA, 2010. ACM Press. ISBN 9781450300117. doi: 10.1145/1930488.1930512. URL http://portal.acm.org/citation.cfm?doid=1930488.1930512.

Rui Nóbrega. *Interactive acquisition of spatial information from images for multimedia applications*. PhD thesis, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa, 2013.

Michael Ortega, Wolfgang Stuerzlinger, and Doug Scheurich. SHOCam: A 3D Orbiting Algorithm. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, pages 119–128, New York, New York, USA, 2015. ACM Press. ISBN 9781450337793. doi: 10.1145/2807442.2807496. URL http://dl.acm.org/citation.cfm?doid=2807442.2807496.

Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich, Charles Loop, Qin Cai, Philip A. Chou, Sarah Mennicken, Julien Valentin, Vivek Pradeep, Shenlong Wang, Sing Bing Kang, Pushmeet Kohli, Yuliya Lutchyn, Cem Keskin, and Shahram Izadi. Holoportation: Virtual 3D Teleportation in Real-time. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 741–754, 2016. doi: 10.1145/2984511.2984517.

Amy Pavel, Dan B. Goldman, Björn Hartmann, and Maneesh Agrawala. SceneSkim: Searching and Browsing Movies Using Synchronized Captions, Scripts and Plot Summaries. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15*, pages 181–190, New York, New York, USA, 2015. ACM Press. ISBN 9781450337793. doi: 10.1145/2807442.2807502. URL http://dl.acm.org/citation.cfm?doid=2807442.2807502.

Augusto Román, Gaurav Garg, and Marc Levoy. Interactive Design of Multi-Perspective Images for Visualizing Urban Landscapes.

John Schlag. Using geometric constructions to interpolate orientation with quaternions. *Graphics Gems II*, pages 377–380, 1991.

Ken Shoemake, Ken, Shoemake, and Ken. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques - SIGGRAPH '85*, volume 19, pages 245–254, New York, New York, USA, 1985. ACM Press. ISBN 0897911660. doi: 10.1145/325334.325242. URL http://portal.acm.org/citation.cfm?doid=325334.325242.

Nathan Silberman and Rob Fergus. Indoor Scene Segmentation using a Structured Light Sensor.

Ivan E. Sutherland. The Ultimate Display. In *Proceedings of the IFIP Congress*, volume 2, pages 506—-508, 1965. URL http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.136.3720.

Ivan E. Sutherland and Ivan E. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I on - AFIPS '68 (Fall, part I)*, page 757, New York, New York, USA, 1968. ACM Press. doi: 10.1145/1476589.1476686. URL http://portal.acm.org/citation.cfm?doid=1476589.1476686.

Yuichiro Takeuchi and Ken Perlin. ClayVision: The (elastic) image of the city. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*, page 2411, New York, New York, USA, 2012. ACM Press. ISBN 9781450310154. doi: 10.1145/2207676.2208404. URL http://dl.acm.org/citation.cfm?doid=2207676.2208404.

Luis Valente, Esteban Clua, Alexandre Ribeiro Silva, and Bruno Feijó. Live-action Virtual Reality Games. jan 2016. URL http://arxiv.org/abs/1601.01645.

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, may 2013. ISSN 1424-8220. doi: 10.3390/s130506380. URL http://www.mdpi.com/1424-8220/13/5/6380/.

Yang Zhang, Robert Xiao, and Chris Harrison. Advancing Hand Gesture Recognition with High Resolution Electrical Impedance Tomography. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 843–850, 2016. doi: 10.1145/2984511.2984574.

Ning-Ning Zhou and Yu-Long Deng. Virtual reality: A state-of-the-art survey. *International Journal of Automation and Computing*, 6(4):319–325, nov 2009. ISSN 1476-8186. doi: 10.1007/s11633-009-0319-9. URL http://link.springer.com/10.1007/s11633-009-0319-9.

M. Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9):25–32, sep 2005. ISSN 0018-9162. doi: 10.1109/MC.2005.297. URL http://ieeexplore.ieee.org/document/1510565/.

Appendix A

Detailed Component Description

Having established the set of interaction and behavioural techniques described during the course for this chapter, one is now capable of further elaborating on the process flow structure described in section 3.3.2.

When describing the high-level architecture of the system in higher detail, it can be favourable to elaborate first a diagram eliciting important organisational figures. An example of such a diagram, allowing for a top down view of all the architecture, is the class diagram. While it is possible to go to arbitrary descriptive depths using modelling languages like UML with all their features, a more realistic approach was taken. The premise is that the value of modelling decreases over time as more effort is put into it.

A shallow model does not mean that it is incomplete, however. It should still contemplate the full breadth of the solution, but restrict the display to essential information. Figure A.1 shows the high-level organisational structure of the proposed solution. The remainder of this section is dedicated to describing this figure.

There are four major isolated groups, evident by their lack of association with other elements. The diagram is read form top to bottom with the superior classes being the primary owners of the association. For instance, *Annotation* and *AnnotationKeyframe* are associated, with *Annotation* being the master. This is not a typical component relationship, as *AnnotationKeyframe* objects are capable of operating within the *Annotation* scope. Due to the nature of the aforementioned association interactions, another way to interpret the relative superiority of connected elements is as representing their instantiating order and chain-of-command. In the previous example, *Annotation* objects would be responsible for instantiating and maintaining *AnnotationKeyframe* objects, as well as delegating any commands issued by *AnnotationComponent* objects.

The rectangle with a dashed-line contour indicates a group of elements that are not integral part of the core architecture – they either contain maintenance mechanisms that do not directly impact other objects, such as cursor event systems, and other measurement features, like the logger classes that gather and write-out data during each session.

Additionally, elements with a "$\multimap$ *MonoBehaviour*" tag indicate that they are sub-classes of Unity's base Component class. This means that objects of these classes can be assigned to game

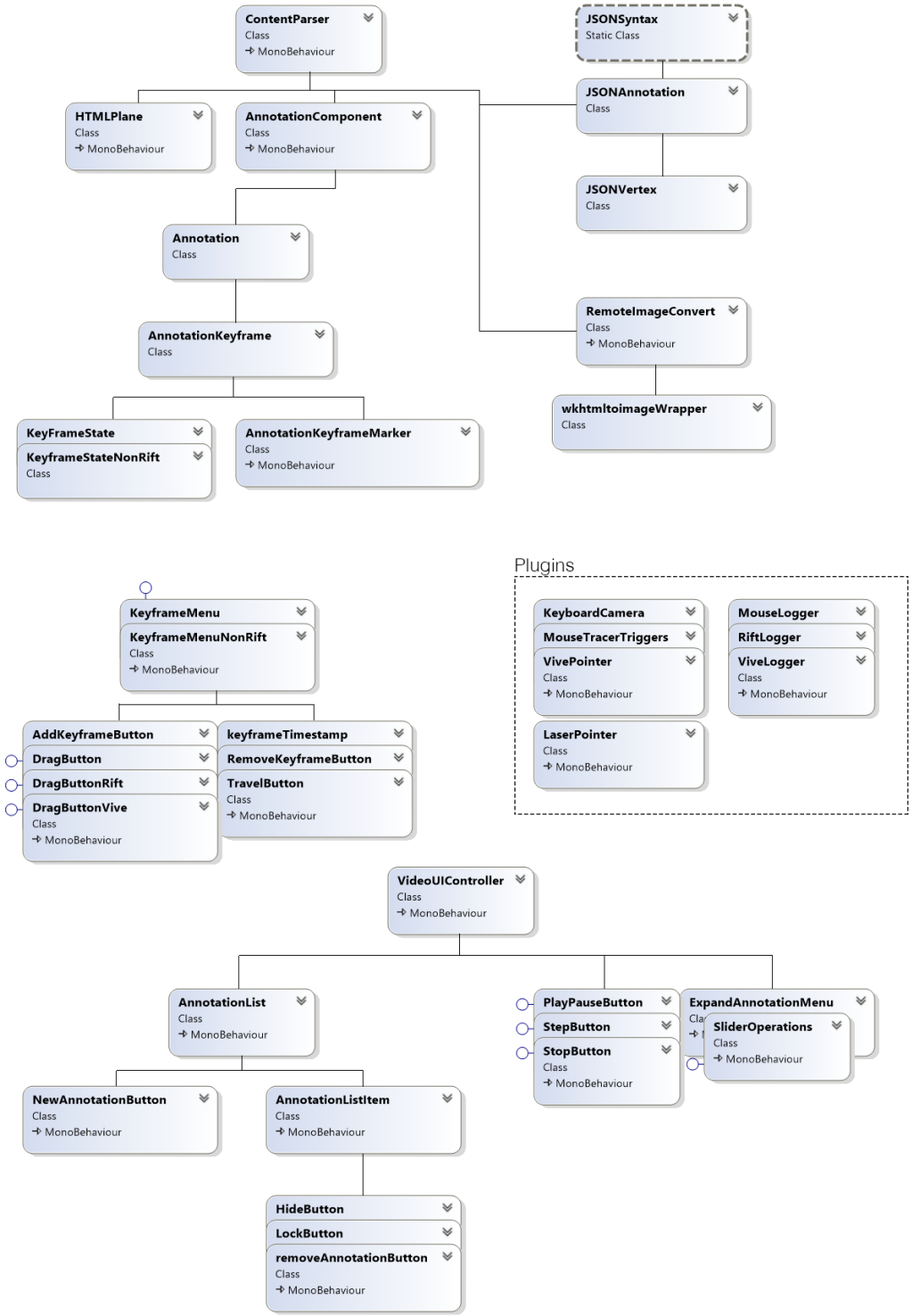


Figure A.1: Architecture diagram

objects in the hierarchy and behave as benefit from the same life-cycle maintenance as game objects do (e.g. they are granted start-up, fixed update within the game loop and clean-up methods to overload). This can serve different purposes. For instance, the logic behind a button's operation should be unequivocally attached to that button's object, while content importers should expose quick configuration options within the Unity editor.

A.1 Core Components

Two different elements stand as the topmost labels – *ContentParser* and *JSONSyntax*. The latter is a static class whose purpose is to define a syntactic common ground with which to operate on JSON objects, both on the textual file and on the programming sides. It wholly defines the names that values can take as is discussed in section 4.4.1, as well as each value's C# type. These definitions are used throughout the implementation to ensure consistent naming – should a value's name change in the data storage protocol, one needs only update it in this static *JSONSyntax* class, which will be transparent to each piece of code that accesses that value. This class is mostly used by *JSONAnnotation* and *JSONVertex* objects, which have the basic function of converting raw, textual JSON elements in string format to their C# counterparts, including the correct typecasting.

This *JSONAnnotation* class – and by extension, *JSONVertex* – are some of the elements directly controlled by the aforementioned *ContentParser*. This *ContentParser* class behaves in a fashion similar to a singleton, but retains the convenience of being a game object in the hierarchy. While this class has an instance running in the scene with all its configuration, it still behaves as a singleton by exposing mostly static methods. One such, already elicited, is the creation of *JSONAnnotation* and *JSONVertex* objects to hold an intermediary stage of the augmented subtitle file parsing.

Parallel to this, the *ContentParser* object also manages the *RemoteImageConvert* class, which features a game object in the hierarchy exposing configuration details and working as a singleton. The main purpose of this element is to concentrate a set of external tools and abstract their usage, while providing a single point of failure for such operations. While this class was designed to allow the addition of multiple tools, only one is actually relevant in the context of this thesis – the conversion of HTML files into images. As discussed in section 4.1.3, the mechanism chosen to accomplish this is a toolchain called *wkhtmltopdf*[1], of which the *wkhtmltopng* tool was used. This tool comes in the form of a Windows command-line binary, which the Unity engine can demonstrably invoke. To accommodate potential changes to external tools, the *wkhtmltoimageWrapper* class was created to serve as an engine-side wrapper for this asset. Should an update to this external tool be needed, this class is the only point that requires rectification.

Given the mechanisms stated above, one needs now an object to which to render the generated imagery. The *HTMLPlane* class was designed to address this. It can be attached as a component to a game object in the hierarchy, rather than existing as an object *per se*. This component is capable

[1] wkhtmltopdf Homepage – https://wkhtmltopdf.org/. Last accessed: 2017-04-14

of creating a mesh onto which the generated texture is applied, effectively creating a canvas for rendering annotation content.

Only one set of elements remains to fully describe the core components of the proposed solution. With the aforementioned classes, one is in possession of all the elements required to build a robust annotation data framework and rendering structure. The *AnnotationComponent* is the central element of this set. The *ContentParser* object is responsible for instantiating these components and assigning them to game objects in the hierarchy. As the pivotal element in the set of annotation-related objects, the *AnnotationComponent* serves as an intermediary between the objects in the hierarchy – i.e. the rendering – and their respective data structures.

These structures are two. The *Annotation* class represents a high-level data structure that holds annotation-wide variables such as its name and displayed image, as well as a list of its keyframes. The *AnnotationKeyframe* class defines exactly these – it serves as a data structure to hold variables and methods exclusive to keyframes that do not concern the annotation they belong to, such as timestamps and positions.

Moreover, each keyframe holds two additional components: a state that describes user interaction scenarios, such as selection or dragging that are important to menus; and a marker that represents each keyframe in the scene space hierarchy. The first, called *AnnotationKeyframeState* – or *AnnotationKeyframeStateNonRift*, for the Desktop and HTC Vive environments – exhibits this separation into two classes due to the differences in event types issued by the Leap Motion controller. This aspect was discovered and accounted for during platform experiments that took place before the design phase. Aside from the events, the functioning is identical – objects of this class hold the state of user interaction for each keyframe, providing a measurement of which interfaces to draw. The final element is the *AnnotationKeyframeMarker*, a component that can be attached to game objects in the scene to provide a visual indication of each keyframe's data. The purpose of this class is to provide a means with which to create new keyframe elements easily, while also serving as an intermediary to propagate user input to the data-oriented classes that see no exposure in the scene.

A.2 Contextual Menu Components

At the centre of the diagram to the left lies a group of elements that together compose the contextual keyframe menu designed in section 3.4.4 and thoroughly discussed in section 4.2.2. The topmost element of this group is again divided into two classes to reflect the differences in event types per device. All elements in this group are sub-classes of Unity's *MonoBehaviour*, which means they need an object in the hierarchy to function. In other words, they each control a specific element of this menu.

Starting with the topmost classes, *KeyframeMenu* and *KeyframeMenuNonRift* handle primarily the invoking and dismissal of such menus. They also manage the flow of commands between menu items and the keyframe object they belong to, serving as a bi-directional means of abstraction.

Additionally, objects of this class provide the other elements in the menu with data regarding the content.

A prime example of this interaction is the *AddKeyframeButton* class, responsible for both the "Add" button of this menu and the actual method chain for generating new keyframes. This is an operation that spans in-scene objects and data-only objects starting with the button itself until it reaches the annotation that will be getting a new keyframe. Naturally, a well-delineated hierarchy is the most efficient way to achieve this while retaining transparency.

To create a new keyframe at the current video time, however, this time must be known, as it is displayed by the *keyframeTimestamp* class object, an addition defended in section 3.4.4.2. This is one of many ways in which the *KeyframeMenu* classes become essential to transport data from higher in the chain-of-command to the elements that need it, and vice-versa.

Similar to the *AddKeyframeButton*, the *RemoveKeyframeButton* executes the simple task of removing the selected keyframe from the scene. This must naturally be propagated, as discussed, and a clean-up must be performed, to ensure none of the deleted objects actually remain in the scene.

An even simpler task, that does not require interaction with other elements besides the video, is the "Travel" or "Fast-forward" method that will force the current video time to match the keyframe's. This is achieved through the *TravelButton* class and object of this type should naturally be requesting the assistance of the *KeyframeMenu* classes, as discussed.

The "Drag" operation, however, is expected to become a different class for each input device as platform tests showed that the handling of dragging events by the Unity engine is not consistent among the development kits provided by Leap Motion and HTC Vive. In fact, most of the event handling for dragging operations had to be redone to provide the same experience across devices. The three classes *DragButton*, *DragButtonRift* and *DragButtonVive* were like so separated in order to provide custom event handling for each device without having to consider per-event device checks.

A.3 Main Menu Components

The group at the bottom of the diagram corresponds to the main menu, with which users have full control of the video, as well as the presence of annotations in the scene. First and foremost the true singleton behaviour of this menu begs for it to be directly incorporated in the hierarchy as a game object rather than instantiated through other components. As such, no central controlling element is needed, as the triggers to invoke and dismiss this menu can be established through game object configuration in Unity.

Rather, the topmost component in this group is one that exposes video flow control methods to this menu object, effectively equipping it with the necessary tools to provide this feature to the users. The *VideoUIController* is exactly this – a group of methods that touch upon only the necessary video flow control features. There is an additional aspect, derived from the game object's

uniqueness, to this component: it can be used to access the menu through Unity's internal component search, so it is not without interest to keep this component unique in order to retain this feature.

The menu, as described in section 3.4.4, is divided into two main parts: a set of video flow control operators, and a global annotation menu accessible through a drop-down button. Considering figure 3.15 one can see how both the set of control buttons and the actual drop-down trigger button can be grouped as a "collapsed menu". This explains the selected grouping in the diagram, with this group being isolated to the right-hand side, while the expansion *per se* and all its constituents are placed on the left.

The group on the right is itself also divided into two. The left-hand side column contains explicit flow control buttons, while the right-hand side contains the drop-down trigger button and the flow control slider. The reader will notice that on the aforementioned left column there are only three classes, while figure 3.15 shows four buttons. This is because the Step method – that allows travel, or seeking along the video – is the same for both frontwards and backwards operation, with the only change being direction. As such, only the *StepButton* class is needed for this operation, as long as it exposes a customisable direction parameter to the editor, so that it can be instantiated twice and contemplate both directions. This class uses one of the video flow control methods exposed by the *VideoUIController* class, called "step", which increments (or decrements) the current video time by a fixed amount. A comfortable value for this amount is one that provides sufficient granularity to justify it as an asset for the user, but at the same time not be too heavy on the media decoding engine.

Similarly, the "Play" and "Pause" methods work as one, since one can only pause the video if it is playing, and vice-versa. These can then work as a single button, depending on the video playback state. This is reflected exactly by the *PlayPauseButton* class. It is the most complex of all button classes as it needs to maintain its state internally. To achieve this it makes use of three methods exposed by the *VideoUIController* component: firstly, the ability to know if the video is playing or not, by analysing the decoder's state; secondly, the ability to pause a playing video; and conversely, the ability to resume a paused video. In order to provide an accurate representation of the button's functioning, the icon it displays should reflect the inner state. That is, it should display a right-pointing triangle when the video is paused (indicating that the video will be resumed if the button is selected), and it should display two vertical bars when the video is playing (indicating that the video will be paused if the button is selected).

The last button in this column is the "Stop" button, which should pause the video and rewind it to the start. This can be achieved using the *VideoUIController* class' methods for seeking (which, unlike stepping, can travel to an arbitrary point in time instantly) and pausing, like in the "Play/-Pause" case. This is contemplated by the *StopButton* class in the proposed solution.

The column on the right features the drop-down trigger button and the flow control slider. The latter is a simple slider that moves the handle autonomously to display the progress of the video. Users can then drag this handle to whichever position they like to change the current video time. The class responsible for this behaviour is *SliderOperations*, which makes extensive use of the

seeking functionalities of the *VideoUIController* component. Not only is this slider capable of responding to the dragging of the handle, it also interprets selection events anywhere on the bar as instant travelling of the handle to that position.

The last button on both this column and the whole right wing of the main menu diagram is the drop-down trigger. *ExpandAnnotationMenu* is the name of the class that grants it its behaviour. Being a drop-down trigger button, one can argue it functions as a toggle, much like the "Play/-Pause" button. As such, it also needs to maintain internal state and change its display accordingly. This is the only button on this side of the diagram that does not make use of the *VideoUIController* class, as it only affects which elements of the menu are visible.

On the left side of the diagram there is a relatively linear sequence of classes. First, the *AnnotationList*, whose only purpose is to organise and render a list exhibiting one unique element per annotation. Part of its organisational function is the delegation and centralisation of commands – selecting a button on the list, such as the "Delete Annotation" button, should propagate an event through the chain-of-command until the appropriate entity processes it and issues back a response. It is this *AnnotationList* class that serves as the intermediary, being responsible for updating the list's state.

All elements on this list are similar, and as such each entry is summarised in an *AnnotationListItem* class. Objects of this class hold information regarding an annotation's colour and name, as well as methods to carry out the commands that can be issued through its buttons. Additionally, each button has a class of its own – *HideButton*, *LockButton* and *removeAnnotationButton* are the three buttons that can be seen in figure 3.18, and whose functionality is described in section 3.4.4.2.

Finally, the last button on this group, and also the bottom-most button on the menu itself is the "New Annotation" button, which is responsible for creating a new annotation with placeholder data as described in section 3.4.4.2. While not strictly necessary, is connected to the *ContentParser* object, such that annotations created using this method are similar in structure to already existing ones.

A.4 Plugin Components

The last group of the diagram, delimited by a dashed-line rectangle, is the Plugin group. It is so called because it does not necessarily constitute a direct translation of the requirements and constraints elicited in section 3.1, and yet it is instrumental to the correct functioning of the proposed solution. It is also divided into two columns.

The column on the left contains classes that complement each device differently to accommodate eventual differences within Unity. For instance, the *KeyboardCamera* class provides a set of event handlers that allow users to control the facing of the camera using keyboard keys, effectively replacing the need for a VR HMD. The *MouseTracerTriggers* class, on the other hand, serves to provide a basis with which to trigger Unity-specific events like selecting and dragging using the world-space cursor. Furthermore, since both the HTC Vive and the Leap Motion controller feature

both hands, this class allows mouse operation to compensate one-handed usage by distributing other event types across both mouse buttons.

The *VivePointer* and *LaserPointer* classes for the HTC Vive and Leap Motion respectively are both equivalent to the *MouseTracerTriggers* class in that they both serve to re-implement parts of the Unity event system. However, unlike the latter, the VR classes also manage pointer objects themselves, such as the laser. Moreover, in both cases the emitted events reflect device-specific information, such as which hand was used to trigger the interaction.

Finally, on the right column there is a set of logger classes, one for each device. *MouseLogger*, *RiftLogger* and *ViveLogger* all collect data during the session and then export them to a file. The type of data that is collected is only different for the desktop version, as head position or hand rotation are meaningless in this scenario. However, the acquisition of these data – namely the interception of events and the extracting of their contents – is different across devices, and as such warrants a specific class for each. Furthermore, being Unity components, they can be attached to a game object in the hierarchy. Since there should be a scene for each device, these classes should never conflict.

Appendix B

Notes on Interpolation

This appendix provides mathematical deductions to supplement section 4.3 of the main document. It contains basic notions of interpolation as well as the deductions to reach spherical Catmull-Rom interpolation.

B.1 Linear Interpolation

For instance, if interpolation is manifestly linear, a set of three points will form a triangular path, and the object will move along the edges without any deceleration at the vertices. This of course requires that interpolation be piecewise, i.e. the overall interpolation function – which can be referred to as a "super-function" – is in reality defined by a set of multiple sub-functions: given a function $f : X \to Y$, there exist n sub-functions $f_i : X_i \to Y_i$, with $i = 1,2,...,n$ and $X_1 \cap X_2 \cap ... \cap X_n = \emptyset$, $X_1 \cup X_2 \cup ... \cup X_n \subseteq X$ that together define f by operating on multiple independent sub-domains. It is then trivial to see how linear interpolation using this technique can have very computationally simple – and therefore fast – solutions. Given three keyframes A, B, and C on triangle ABC, with A being the first keyframe, followed by B and then C, the object's path is given by the set of vectors $\{\overrightarrow{AB}, \overrightarrow{BC}\}$. As discussed, keyframes also store information about time, thus in reality becoming a four-dimensional point as such: $A = (x,y,z,t)$, with x,y,z being positional components, and t being a temporal one. It is still possible to use the $\overrightarrow{AB}$ notation since time has no bearing on the spatial vector defined by the position of two keyframes. However, time yields an essential tool in parametrising the super-function – the passage of time can be used to pinpoint which sub-domain X_i to operate on. Suppose one wants to find where in the edges of triangle ABC lies a point $D = (x_D, y_D, z_D, t_D)$ of which only time t_D is known. As discussed, the object's path is along one of two line segments in the set $\{\overline{AB}, \overline{BC}\}$, which make up the sub-domains X_1 and X_2 that fully define X. Given that $t_A \leq t_B \leq t_C$, one can infer if D lies within $\overline{AB}$ or $\overline{BC}$ based solely on t_D. For the sake of argument, assume that $t_A \leq t_D \leq t_B$. This means that $D \in \overline{AB}$. To compute the x_D, y_D, z_D coordinates, one can use equation 4.1 presented above, as such:

$$D = A + w(B - A), \quad w = \frac{t_D - t_A}{t_B - t_A} \tag{B.1}$$

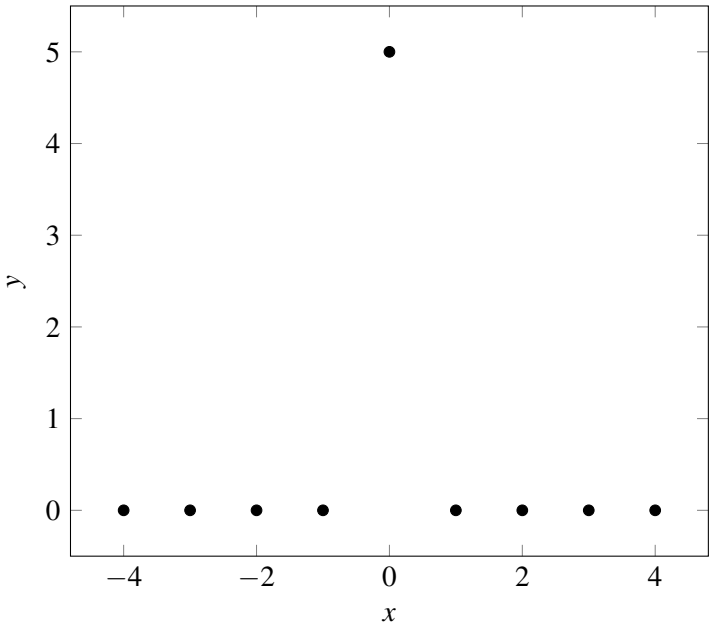
Using this technique, one can rapidly deduce where the object should be in 3D space based solely on time. This assures that at any given time t the position of each object is known.

B.2 Spline Interpolation

A property of linear interpolation that can formalise the problem of sharp, ragged paths is that *knots* – that is, points where the different sub-domains connect – are points at which the super-function is not differentiable. Since every sub-function is linear, knots provoke an abrupt change in slope that is incompatible with differentiation. While this is not a problem for computations and rendering, it certainly is indicative of how smooth – or not – transitions can be.

Consider a set of nine points on a two dimensional plane, of which eight are collinear and one is an outlier, i.e. does not fall within the same line as the others, as seen on figure B.2.

Suppose now these points represent keyframes and one wishes to connect them as a path would. For the sake of argument, assume that they are sequential from left to right, and equally spaced in time.



If one were to use linear interpolation as discussed in section B.1 the result would be that of figure B.1. Is is easy to see how simple this method would be to implement. Provided one knows on which sub-domain the desired point lies (by examining its temporal component), one can use equation B.1 to completely define that point.

Furthermore, looking at the shape of the function's transitions – evident at $x = \{-1, 0, 1\}$ – one can see that they are abrupt. This can be reinforced by the non-continuity of its first derivative.

Figure B.1: Linear interpolation.

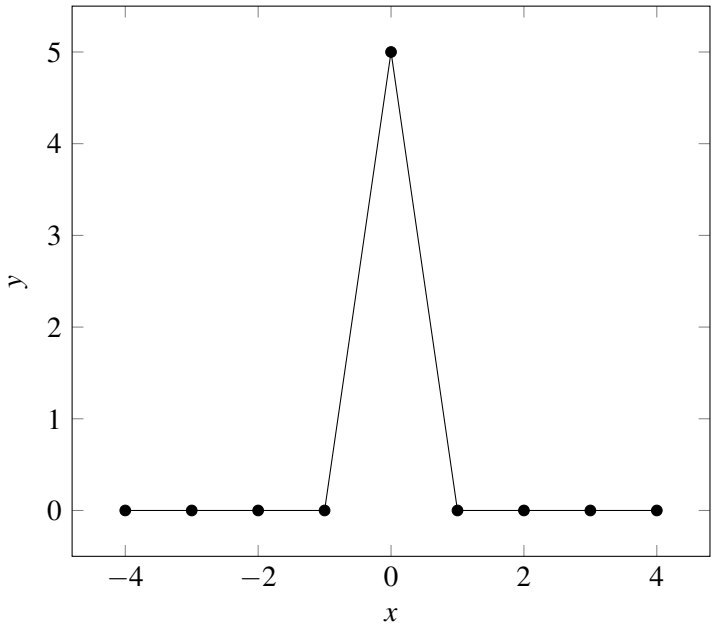


Figure B.2: Set of arbitrary collinear points and one outlier.

If one desires smoother transitions, one must find an interpolant super-function that maintains a continuous first derivative. A naïve yet valid solution is to apply polynomial interpolation. The process – called fitting – is one by which a polynomial p of degree n or less is found so that for a set of $N = n+1$ 2D points, p obeys $p(x_i) = y_i, \quad i = 1,2,...,N$. The attentive reader will remark that for this formulation to work an additional condition must be met: each point in the set to serve as control points must be unique, as no polynomial can intersect itself in 2D or 3D space. While this is unacceptable for keyframing qualities – as intersecting shapes are common – there is one additional remark to be made, evidenced by this method. The oscillation seen in figure B.3 is a behaviour evident when interpolating equally spaced points. It is called Runge's phenomenon, but its discussion is outside the scope of this dissertation. It is worth mentioning though that this oscillation is not expected behaviour when defining a path using keyframes, at least on the neighbourhood of the outlying point.

To solve the self intersection issue, a new interpolation approach is needed. It is possible to take the piecewise nature of linear interpolation and apply it to smoother curves. If instead of a linear equations – which are a degree one or less polynomials – one employs a higher degree polynomial, such as a third-order polynomial, one obtains a spline. A spline is a piecewise-defined super-function whose sub-functions are higher order polynomials.

This approach has two main advantages. Firstly over linear interpolation, as it is able to retain a continuous first derivative throughout, resulting in greater smoothness around *knots*. Secondly over polynomial interpolation, as it massively reduces the effects of oscillation.

The simplest form of polynomial splines is the cubic spline. Following the piecewise approach that results in equation 4.1, consider a point $D = (x_D, y_D, z_D, t_D)$ to be interpolated based on t_D.

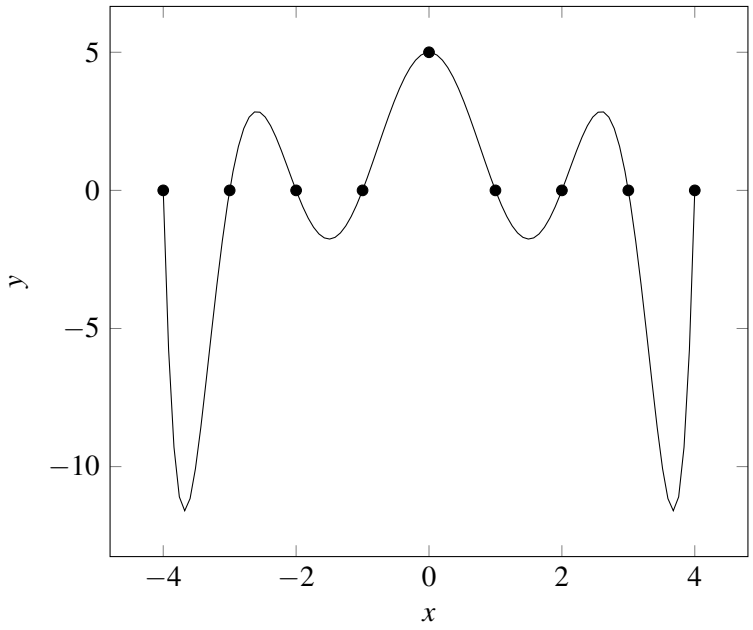


Figure B.3: Fitting eight-order polynomial.

One can the argue that the positional component of D can be written as $D_p = (x_D, y_D, z_D)$ and is therefore a function of t_D within the sub-domain defined by D's neighbourhood. The positional component of any point Y can then be written as a function of t, obeying the rule that $Y_p(t'_Y) = (x_Y, y_Y, z_Y)$, assuming t'_Y is already normalised according to its neighbouring points – e.g. X and Z, with $t'_Y = \frac{t_Y - t_X}{t_Z - t_X}$ and $t_X \leq t_Y \leq t_Z$ – to conform to $t'_Y \in [0,1]$.

Having established that $t \in [0,1]$ as before, and knowing Y is a point within an already defined sub-domain of a third degree polynomial, one can canonically write a parametrisation of Y_p as follows:

$$Y_p(t) = a + bt + ct^2 + dt^3 \tag{B.2}$$

One needs only find the correct coefficients $\{a,b,c,d\}$ to fully define any point Y with respect to t'_Y.

Back to point D of which only t_D is known, let's consider one wants to interpolate the position D_p of this point between the already defined points A and B, with of course $t_A \leq t_D \leq t_B$ still holding true, and therefore $t'_D = \frac{t_D - t_A}{t_B - t_A}$. Point D is then defined from equation B.2 as $D_p = Y_p(t'_D)$. It is known that point D could be coincident with points A or B, both being corner cases for $t'_D = 0$ and $t'_D = 1$, respectively. With this, one arrives to the following equations:

$$D_p(0) = A = a \tag{B.3}$$

$$D_p(1) = B = a + b + c + d \tag{B.4}$$

By taking the first derivative of $D_p(t)$ one then obtains the following:

$$D'_p(0) = b \tag{B.5}$$

$$D'_p(1) = b + 2c + 3d \tag{B.6}$$

And by solving this four-equation system one obtains:

$$a = A \tag{B.7}$$

$$b = D'_p(0) \tag{B.8}$$

$$c = 3(B - A) - 2D'_p(0) - D'_p(1) \tag{B.9}$$

$$d = 2(A - B) + D'_p(0) + D'_p(1) \tag{B.10}$$

One can see how the coefficients depend not only on the neighbouring points but also on the function's derivative at that point. In theory, since A and B are already defined, one can establish that by explicitly setting the derivatives at each point by means of a tangent, one could fully define a spline capable of smoothly and interpolating a point D between A and B.

The so-called "natural" cubic splines in specific are obtained expanding this method – by forcing the second derivative to also be continuous one obtains enough conditions to solve the equation system for both the points' positions and the derivatives at those points, and therefore there is no need to relegate establishing the tangents to additional control points. While this automates the process, the result may yet not be acceptable. As seen on figure B.4, an oscillation is still produced.

While the act of explicitly controlling the derivatives at each knot yields a degree of control not present in other types of interpolation, one must consider how this translates to keyframing.

For instance, this explicit control is present in advanced path-making systems such as those featured in illustration or digital painting programs. Typically this is done through composite Bézier splines, which allow for the control of tangents at each point on a per-segment basis. This method is arguably the most flexible, as one can virtually attain any desired degree of smoothness, be it continuous curves or sharp vertices.

Despite this, this type of control is not very practical for limited-interaction virtual reality scenarios. Considering a virtual environment with limited interaction, juggling knot and control points would make the interface unnecessarily confusing, especially among unfamiliar users.

A type of cubic splines, called cubic Hermite splines, attempt to solve the control point problem by taking the natural cubic equations system and detaching them form the equation system. Starting at equations B.7 through B.10, the coefficients can be determined by explicitly setting the value of the derivatives at each knot, as discussed. Cubic Hermite splines solve this by computing

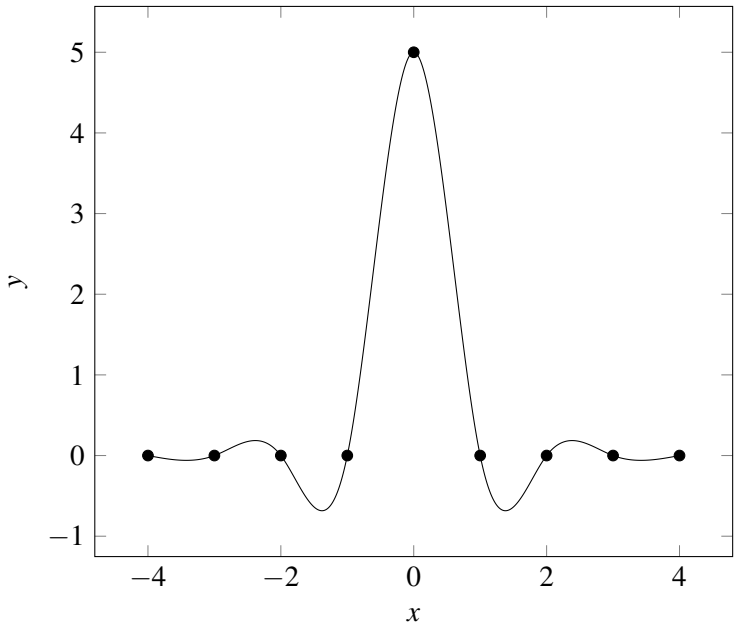


Figure B.4: Natural cubic spline interpolation.

the derivatives based on neighbouring knots, effectively using them as control points. For instance, a common method is to use a three-point finite difference to approach the derivative at that point. Other methods can take parameters like tension into account, effectively altering the shape of the curve.

One particular case of these splines, however, is the Catmull-Rom spline, named after Edwin Catmull and Raphael Rom. The tangent at a point is found by computing the slope of the secant line that intersects both of the immediate neighbouring points. This gets rid of propagating oscillations, as seen on figure B.5. In the specific case of an outlier, oscillation in its vicinity is inevitable. However with careful placement of the points this overshoot can be mitigated. The main advantage of this approach is that oscillation is completely local, and points that are distant from the outlier retain their linearity.

In his paper, Schlag (1991) describes a method for locally determining a point in a spline using de Casteljau's algorithm and Barry and Goldman's pyramidal formulation. The full discussion of the method is unnecessary. Knowing that the derivative at point P_i is given by the slope of the secant line $\overline{P_{i-1}P_{i+1}}$, one can write equations B.7 through B.10 as functions of P_{i-1},

Since one wishes now to obtain the value for the tangent at a point X, one needs to know the coordinates of point W right before it, and of point Y right after it. To find the tangent at point Y, point Z after it is also needed. For clarity and to generalise, let $P_i = X$, making $P_{i-1} = W$, $P_{i+1} = Y$ and $P_{i+2} = Z$.

P_i, P_{i+1} and P_{i+2}. Using these, and applying both de Boor's algorithm for evaluating B-spline curves and Neville's recursive algorithm for solving Lagrange polynomials as described by Barry et al. (1988), one arrives to an algorithm for recursively evaluating a Catmull-Rom curve. However, as Schlag noted, this notation can be compressed greatly as coefficients can be collapsed –

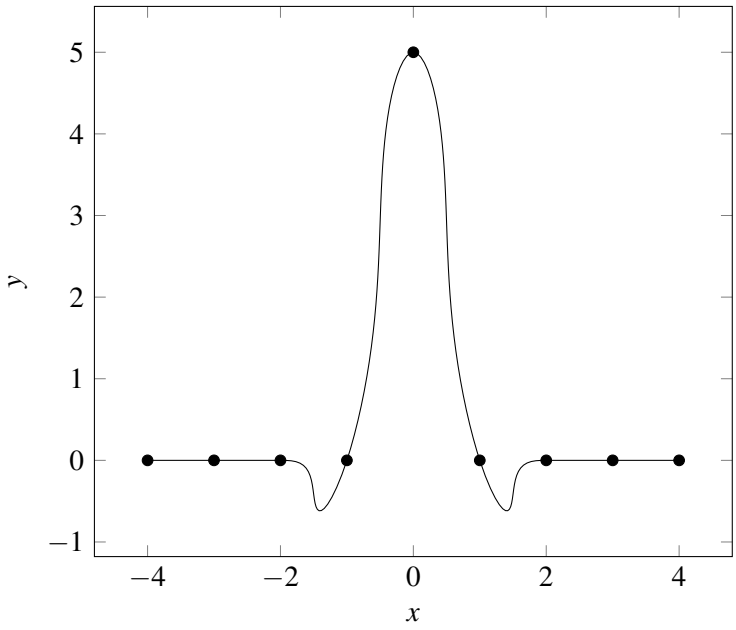


Figure B.5: Catmull-Rom interpolation.

since each tangent is yielded by a pair of points and their slope, several coefficients become neutral by summing to one. This happens because – using the case of Bézier curves and de Casteljau's algorithm – a spline can be evaluated at a point by means of recursive linear interpolation of control points. Consider, for instance, that equation 4.1 could be written as

$$lerp(p_1, p_2, w) \tag{B.11}$$

where p_1 and p_2 are points in space and $w \in [0,1]$. De Casteljau's algorithm becomes then a nested $lerp$ operation, like so:

$$\begin{aligned} Bz(p_1,p_2,p_3,p_4,t) = lerp(&lerp(lerp(p_1,p_2,t),\\ &lerp(p_2,p_3,t),\\ &t),\\ &lerp(lerp(p_2,p_3,t),\\ &lerp(p_3,p_4,t),\\ &t),\\ &t) \end{aligned} \tag{B.12}$$

This is of course, according to the original formulation, which is valid only for Bézier curves. This method can be called "balanced" as the value of w is always the same: t. However, as Schlag states, the compression of Barry and Goldman's algorithm yields a set of values for w that, when applied in their respective order to the operation described in equation B.12, produce the evaluation of a point in a Catmull-Rom spline.

As such, de Casteljau's algorithm can also be used to recursively evaluate a Catmull-Rom spline, as so:

$$\begin{aligned} Cr(p_1,p_2,p_3,p_4,t) = lerp(&lerp(lerp(p_1,p_2,t+1),\\ &\quad lerp(p_2,p_3,t),\\ &\quad \frac{t+1}{2}),\\ &lerp(lerp(p_2,p_3,t),\\ &\quad lerp(p_3,p_4,t-1),\\ &\quad \frac{t}{2}),\\ &t) \end{aligned} \tag{B.13}$$

This is, of course for the Euclidean space where points can move freely. When one desires to apply constraints to this space – as will be needed – one can assure proper Catmull-Rom interpolation as long as the target space features a *lerp* analogue.

It is not enough to simply project the Euclidean space interpolation into the target space. Consider for instance a spherical surface, onto which all interpolation will occur. Consider now a curve that crosses the sphere's centre, and then travels to the outside of the sphere and wraps back in again. This curve is not impossible to generate (by placing two knots at diameter's length), and yet is impossible to project, as the centre point has no analogue on the surface. Furthermore, on paths that are orthogonal to the spherical surface the projection is lossy, as distances travelled on the surface will be much smaller than distances travelled outwards.

B.3 Spherical Spline Interpolation

The solution then is to apply the Catmull-Rom interpolation described in equation B.13. For that to work on a spherical surface, one must find an analogue of *lerp* for this space. In his paper, Shoemake et al. (1985) describes a constant-velocity rotation on a spherical surface as a way to interpolate quaternion rotation linearly. The method is quite simply to linearly sweep the arc given by the curve where the sphere intersects a plane through the two endpoints and the origin of the sphere. This operation can be mathematically deduced as such:

$$slerp(p_1,p_2,t) = \frac{\sin(1-t)\theta}{\sin\theta}p_1 + \frac{\sin t\theta}{\sin\theta}p_2, \quad \text{where } \cos\theta = \frac{p_1 \cdot p_2}{\|p_1\| \cdot \|p_2\|} \tag{B.14}$$

Since *slerp* is the analogue for *lerp* on spherical surfaces, one can immediately employ the algorithm described in equation B.13 to obtain Catmull-Rom interpolated splines on a spherical surface. As mentioned, provided one has pinpointed the neighbouring points P_i and P_{i+1} and their

respective neighbours P_{i-1} and P_{i+2}, for a point one wishes to find based on t, one can directly apply the following:

$$\begin{aligned} Cr(P_1,P_2,P_3,P_4,t) = slerp(&slerp(slerp(P_1,P_2,t+1),\\ &\quad slerp(P_2,P_3,t),\\ &\quad \frac{t+1}{2}),\\ &slerp(slerp(P_2,P_3,t),\\ &\quad slerp(P_3,P_4,t-1),\\ &\quad \frac{t}{2}),\\ &t) \end{aligned} \tag{B.15}$$

This technique ensures constant velocity sweeping of spherical arcs, while generating smooth curves constrained to a spherical surface. These curves can be further customised by adding keyframes – or knots – and moving them in space and time, allowing for flexible and intuitive path-making.

Appendix C

Whisker-box Plot Reading Guide

A summary of the information contained in whisker-box plots is included in this section. This type of plot is useful in discriminating both the spread and the central tendencies of a set of data, but its semantics can vary. Across this document, these types of plots have an appearance similar to that of figure C.1.

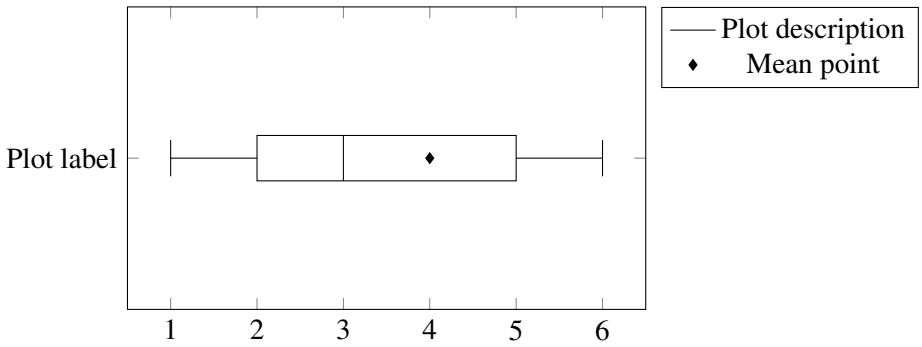


Figure C.1: Example of a whisker-box plot.

A whisker-box plot is a collection of five numbers, and an optional sixth. These five points are represented by vertical line segments, and the optional sixth by a single point. The five vertical lines are, from left to right: the minimum value (*min*) of the data-set, $min = 1$ in this case; the first quartile ($Q1$) below which 25% of the data point values lie, $Q1 = 2$ in this case; the second quartile, or median (M), for the data-set, in this case $M = 3$; the third quartile ($Q3$) above which 25% of the data point values lie, $Q3 = 5$ in this plot; and finally the maximum value (*max*) within the data set, placed at $max = 6$. The optional value that does not appear in all plots is the mean, or average (μ), represented here at $\mu = 4$. A summary of this can be gathered from figure C.2.

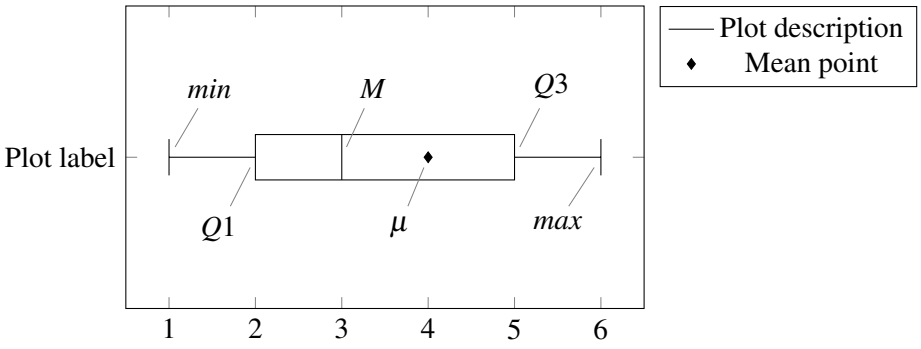


Figure C.2: Example of a whisker-box plot with point labels. From left to right: minimum (*min*), first quartile ($Q1$), median (M), mean (μ), third quartile ($Q3$) and maximum (*max*).