POLITECNICO DI MILANO
Corso di Laurea magistrale in Ingegneria Informatica
Dipartimento di Elettronica e Informazione


# Merging the Two Worlds: A Novel Approach to the Design of Mixed Reality Experiences

In collaborazione con
Electronic Visualization Laboratory
University Of Illinois, Chicago

Relatore: Prof. Pier Luca Lanzi
Correlatore: Prof. Angus Forbes

Tesi di Laurea di:
Marco Cavallo, matricola 817065

Anno Accademico 2015-2016

# Acknoledgements

I would like to thank all the people who have always believed in my crazy visionary ideas, supporting me during my work. I particularly thank Angus Forbes for his creative feedback and Geoffrey Alan Rhodes for all the interesting talks that inspired our work. I also want to thank Andrew Johnson for having given me the right ideas at the right moment and my advisor Pier Luca Lanzi for having introduced me to this amazing perspective on Computer Science.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| UIC | University of Illinois at Chicago |
| AR | Augmented Reality |
| MR | Mixed Reality |
| VR | Virtual Reality |
| GPS | Global Positioning System |
| A-GPS | Assited Global Positioning System |
| IMU | Inertial Measurement Unit |
| DPI | Dots Per Inch |
| SURF | Speeded Up Robust Features |
| SIFT | Scale-Invariant Feature Transform |
| ORB | Oriented FAST and Rotated BRIEF |
| CPU | Central Processing Unit |
| WGS84 | World Geodetic System 1984 |
| GPS | Global Positioning System |
| HMD | Head Mounted Display |
| CAVE | Cave Automatic Virtual Environment |
| IR | InfraRed |
| LAN | Local Area Network |
| RPC | Remote Procedure Call |
| JSON | JavaScript Object Notation |

| | |
|---|---|
| RSSI | Received signal strength indication |
| URL | Uniform Resource Locator |
| NAT | Network Address Translation |
| LCD | Liquid Crystal Display |
| WOW | Window On the World |
| AI | Artificial Intelligence |
| POI | Point Of Interest |

# Summary

In this document we present a novel approach to the design of mixed reality (MR) applications, which combines elements from markerless and location-based augmented reality in order to explore new types of interaction, leveraging a spatial representation of virtual content. In particular, we propose a method for developing both a mobile application, that can be used by people to see augmented content in the real world, and an editor application, which can be used to design and create MR experiences. In our presentation, we deal with the currently available mobile technologies and we discuss the integration of different techniques in order to improve tracking within the context of our location-based approach, in which every content has a defined position in real space. After discussing the advantages and the new interaction possibilities enabled by our method, we define the requirements for a MR authoring tool and we propose our own implementation, aimed at decoupling the editing process from the use of tracking techniques. We particularly focus on the mapping between real-world and virtual coordinates and on the possibilities enabled by real-time editing, which allows the designer to remotely preview the MR experience and to interact with users. We demonstrate the usefulness of our method by presenting two applications developed with our approach and by analyzing some real-world examples. Ultimately, we discuss the feasibility and the future of creating accurate MR experiences with the currently available technologies, by analyzing both quantitative and qualitative results obtained from our case studies.

# Sommario

Con il presente documento presentiamo un nuovo approccio al design di applicazioni di mixed reality (MR), combinando elementi di realtà aumentata markerless e geolocata al fine di esplorare nuovi tipi di interazione - basati su una innovativa rappresentazione dello spazio in cui vengono definiti i contenuti virtuali. In particolare, proponiamo sia un framework per sviluppare applicazioni mobili, che possano essere usate come tramite per osservare contenuto digitale nel mondo reale, sia un sistema di editing finalizzato al design ed alla creazione di esperienze di mixed reality. Durante la nostra discussione, trattiamo le attuali tecnologie mobili necessarie alla creazione di tali applicazioni e discutiamo l'integrazione di diversi sensori al fine di migliorare il tracciamento con un approccio ibrido, basato contemporaneamente su tecniche di elaborazione di immagine e su geolocalizzazione, nel quale ogni contenuto digitale possiede una referenza spaziale al mondo reale. Dopo aver discusso i vantaggi e le nuove possibilità di interazione offerte dal nostro metodo, definiamo un insieme di requisiti necessari allo sviluppo di un tool di editing per applicazioni di MR e proponiamo una nostra implementazione. Questa seconda parte si concentra in particolare sulla mappatura tra coordinate reali e virtuali e sulle funzionalità rese possibili dalla modifica a tempo reale dei contenuti digitali, che consente al designer di di avere un'anteprima della esperienza di MR e allo stesso tempo di interagire con gli utenti. Al fine di dimostrare l'utilità del nostro metodo, presentiamo due applicazioni mobili sviluppate mediante il nostro framework e analizziamo alcuni casi di studio ad esse associati. Infine discutiamo la fattibilità, l'accuratezza ed il futuro della creazione di esperienze di MR con le tecnologie attualmente disponibili, esaminando quantitativamente e qualitativamente i risultati ottenuti tramite i nostri test.

# Chapter 1

# Introduction

Since the dawn of humanity, man has always dealt with an environment characterized by physical objects, populating an inherently tridimensional world. Through the use of the five senses, man has been able to explore the bounds of phisicality and to interact with tangible elements. The way concrete objects exist and move in our spatial universe have always been guided by the laws of physics, creating in man an innate sense of perception of the reality in which he lives: elements of nature are supposed to behave in a certain way, daily activities are expected to produce determined results. Many actions appear natural and are taken for granted, like when our visual perspective changes because we are moving our head. We already know that, by applying a small force, we can move a small object situated on the table in front of us; we expect that, after launching an object up in the air, sooner or later it will fall down onto the floor; if we put our hand closer to the fire, we already forecast the possibility of burning our hand. Almost everything in our daily lives has become familiar or at least follows some rules from which we can expect a particular behavior. Thousands of years later, all of a sudden, the advent of computing gave birth to a completely new type of entity, intangible and characterized by totally different conventions. A new realm of reality was introduced and people had to start learning how to interact with digital content: objects were not physical anymore, but unreal - virtual. Like he did during all of his existence, man tried to adapt to this novelty and evolved, defining better ways to conceive these new elements and developing innovative interfaces to deal with digital content. In the attempt of bridging the gap that had been suddenly created, many approaches tried to uniform the real and the virtual, aiming at creating forms of interaction that could be perceived as "natural" by human beings - and, at the same time, at controlling the almost limitless potentialities conceivable

through the use of this technology. In particular, it turned out that man was now able to define his own rules in the virtual world, becoming himself the creator of the environment in which he lived. Despite the possibility of shaping the world as he wanted, the need of making it appear natural - and thus consistent with what already existed - still was a fundamental issue to be solved. This is why the study of mixed reality represents a possible way to address this problem, exploring at the same time the boundaries of interaction between what concretely exist and what we would like to exist.

## 1.1 The virtuality continuum

The terminology "mixed reality" (MR) was formally defined for the first time by Milgram and Kishino in their paper "A taxonomy of mixed reality displays", published in 1994 [61]. The two authors explore through their work the concept of presenting, within the same display environment, both a "virtual space" and a "reality"; at the same time, they propose a definition of the so-called *virtuality continuum*, whose simplified representation is shown in 1.1: completely real environments lie on the left side of the diagram, while completely virtual environments are situated at the right extremity - everything that falls in between along this continuum is called mixed reality.

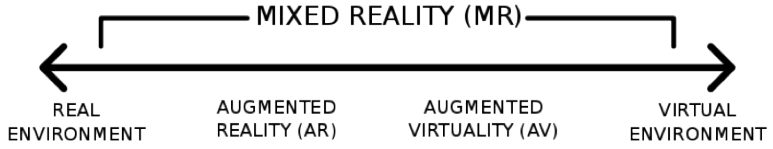


*Figure 1.1: Virtuality continuum as defined by Milgram and Kishino [61]: mixed reality consists in merging real world and virtual worlds elements to produce a new type of environment, enabling the co-existence and the interaction of physical and digital objects. According to this definition, augmented reality and augmented virtuality represent two subsets of the broader concept of mixed reality.*

Thus, mixed reality can be conceptually defined as the merging of virtual and real-world elements within the same display environment. According to Milgram's taxonomy, a further distinction can be made by identifying augmented reality (AR), where a real environment is augmented by means of computer generated graphics, and augmented virtuality (AV), which aims at augmenting with real elements a mostly virtual world. In relation to this, it is useful to clarify the distinction between the concepts of real and virtual:

- real objects "have an actual objective existence" [61]

- virtual objects do not formally or actually exist, but are synthesized by a computer.

As pointed out by Milgram and Kishino, despite virtual content is often presented with the aim of making it appear "realistic" to the user to give the impression it does not belong to an artificial world, the definition of the two concepts above is completely unrelated to the idea of "realism".

In order to bridge the gap between these two overlapping but inhrerently separated worlds, mixed reality leverages many different enabling technologies, such as special types of displays, tracking and registration techniques, user interfaces and visualization methods - which we will describe with more detail in *Chapter 2*. At the same time, we will try to introduce the reader to some of the various fields to which MR has been applied to over the last two decades, spanning from medical and urban planning applications to education and entertainment.

Most of the topics we are going to present in this document are related to the concept of enriching the real-world, adding virtual content to reality with the aim of providing to the user otherwise unavailable information. Despite there does not exist a rigid bound between the definitions of AR and VR, our work could be mostly classified as augmented reality. However, where appropriate, we preferred to always use the broader term "mixed reality" since in many cases our approach is still valid along all the MR virtuality continuum. An additional reason for this decision is related to the recent meaning assumed by this terminology, especially on the internet and outside research laboratories, according to which mixed reality is a sort of augmented reality leveraging more elements which originally belonged to virtual reality. Despite this definition denies Milgram's terminology, we believe it suits well our proposed approach and takes into account the improvements obtained by VR in the last few years. Considering all these motivations, "mixed reality" seems a more appropriate term for addressing our work.

## 1.2 Motivation

The cinematographic production of films like *The Matrix* (Warner Bros., 1999), *Minority Report* (DreamWorks Pictures, 2002) and *Iron Man* (Paramount Pictures, 2008) has progressively used the audience to futuristic sci-fi elements, comprehending innovative 3D user interfaces and gesture-based interaction with virtual objects in tridimensional space. Acknowledging the rapid evolution of technology, these ideas have created great expectations in the general public, but unfortunately most of them still remain only post-

processing video effects, leading to a sort of sense of disillusion. However, motion-pictures have greatly contributed as a source for inspiration and, as time passes, man has discovered more ways to attempt to fill this gap between reality and expectations for the future. Considering our personal passion for sci-fi movies and film production, we personally believe that the recent improvements in technologies like virtual and mixed reality, despite their current limitations, is a huge step forward in this process, so that this research field is gathering more and more attention from a wide spectrum of stakeholders.

The possibility of creating ourselves something that does not exist in reality, interacting with it like it was a real entity and defining our own rules in the world we are shaping is definitely thrilling. It is not anymore just about expressing our creativity and fantasy by adding content to a video with *After Effects* [2] after it has been recorded, but it involves a completely new way to perceive and interact real-time with an enhanced reality. Mixed reality is enabled by a full stack of different computer technologies that need to be combined together: it is something different, an interdisciplinary field mixing subjects such as computer vision, sensor tracking, computer graphics and user interfaces, requiring the researcher to develop a minimum background in all of them and to know how they communicate among them. The possibility to learn many different technologies at the same time and the chance to work on some of the most creative and fun topics of computer science led us to this research.

Since we tried for the first time to develop an application with the Vuforia [96] library, we realized that our interest was also in creating something that concretely worked on a physical device, used daily by billions of people. This is why we preferred an approach with a significant implementation part, aimed at making mixed reality more available to everyone and thus addressing issues on a nowadays very common type of handheld device: the smartphone, with all its continuous improvements and all its still compelling limitations. In our research, we also made use of more expensive technologies to demonstrate our contribution, but a special consideration is always kept towards the daily technology we face every day. Our aim is to contribute as we can to a very complex and often still unshaped field, in order to make mixed reality more easily accessible to everyone. The feeling is that, despite many concepts were already defined 20 years ago and haven't changed radically since then, this historical moment in technology is introducing many new ideas, eventually allowing us to be more influential in shaping the future of mixed reality in the years to come.

## 1.3 Contribution

The research presented in this document involves many different aspects related to mixed reality, spanning from a concrete implementation on physical devices to a more abstract generalization for presenting virtual content. Our particular approach tries to merge the lessons learned from previous works in augmented reality and to add some elements originally belonging to virtual reality, trying to conceive mixed reality as an experience defined in a tridimensional space where each virtual object is bound to a real world location and where each real element corresponds to specific virtual coordinates. Differently from previous AR works which solely relied on estimating the relative position of a user with respect to a fiducial, our study is aimed at defining ways to present virtual content that take into account context-specific characteristics and environmental features, which allow us to estimate the absolute pose of a mobile camera and to leverage the relationships between user and virtual objects and among virtual objects themselves. Instead of activating independently virtual content based on specific trigger events, for instance the identification of a pattern image, we want instead to model an entire parallel virtual world which is constantly overlapped to the real one based on the movements of the user: within certain accuracy constraints, this method enables presenting virtual content more freely in 3D space, even in situations in which not many tracking features are present in the environment. This is made possible by creating an abstraction for currently existing tracking technologies, that we combine in a single virtual space in in order to exploit the benefits and correct the drawbacks of each one.

Following a first part in which we present how our paradigm can be applied to implement a mixed reality mobile application, we leverage the same concepts to provide a new way to conceive the authoring of MR experiences, presenting the implementation of an editor application that allows easy real-time customization of virtual content. In the wake of this innovative authoring tool, we also explore new possibilities to study the behavior of users, preview a mixed reality application from their perspective and allow direct interaction between them and the designer.

## 1.4 Document structure

The present document is structured in order to first introduce the reader to the field of mixed reality, aiming at giving a basic historical background on the enabling technologies and on the common issues and limitations en-

countered by MR in the last 20 years. After having introduced some related works, the document will describe all the steps involved in our proposed approach, specifying how we leveraged available technologies to provide our contribution to this research field. With the aim of providing more real world examples and to demonstrate the feasibility of our method, we will then present a chapter dedicated to two concrete applications that we developed with our approach. Finally, we will summarize our work in the final chapter, by also taking some considerations about the present and future of mixed reality applications. Overall, the document structure appears as following:

- *Chapter 1*, Introduction: basic definitions and motivations that brought us to this research, with additional information about the nature of this document.
- *Chapter 2*, Related Work: brief histocarical background on mixed reality and its evolution over the years, arriving to the current state-of-the-art technologies.
- *Chapter 3*, Mobile Application: presentation of our approach to the development of MR mobile applications, with a particular focus on tracking methods.
- *Chapter 4*, Authoring Tool: definition and implementation of the requirements for developing a real-time editor for MR.
- *Chapter 5*, Case Studies: presentation with qualitative and quantitative results of two mobile applications we developed by using our proposed approach.
- *Chapter 6*, Conclusion: final considerations and summary of our research work, with a discussion part involving the future of MR.

# Chapter 2

# State of the Art

This chapter is aimed at providing the reader a simple background on mixed reality, presenting the technologies involved in this research area and the solutions that were developed during the years to address common issues. During our discussion, we will also deal with the possible applications of MR and we will show how the evolution of technology over the last decades is rapidly improving the possibilities it may offer to an always broader public. As we mentioned in *Chapter 1*, the terminology "mixed reality" has become popular only recently and its definition has slightly changed over time. Despite most of the works presented in this chapter are specifically addressed to augmented reality, we decided to use anyway the broader term mixed reality in order to comprehend in a more general way all methods that mix virtual content with the reality we perceive.

Despite the first works involving augmenting reality go back to the 1960s, when for the first time Ivan Sutherland presented computer-generated 3D graphics on a see-through display [84], only in the 1990s enough work was produced in order to define MR as a research field. After the already mentioned taxonomy proposed by Milgram and Kishino in 1994 [61], Azuma et al. published in 1997 a survey [12] aimed at describing more precisely this field and at summarizing the work that had been done until then. In the same years, some annual conferences on MR were organized, among which we can mention the *International Symposium on Mixed Reality* (ISMR) and the *Designing Augmented Reality Environments* workshop; other organizations like the *Mixed Reality Systems Lab* in Japan also started focusing on MR with relevant investments and the first open-source toolkit (called *ARToolkit*) for rapidly building this type of application was developed. In 2002, ISMR was united for the first time in Germany with the *International Workshop on Augmented Reality*, giving birth to the *International Symposium on*

*Mixed and Augmented Reality* (ISMAR), which still nowadays represents one of the most influential conferences on MR worldwide. From the early 2000s technology has improved enormously, however most of the original definitions and taxonomies conceived by Milgram, Azuma and other important authors are still very actual, allowing us to see the evolving solutions to the instrinsic issues related to the concept of MR.

## 2.1 Enabling technologies

Among the technologies required in order to produce a mixed reality experience, a first fundamental aspect is the way virtual content is added to the reality and presented to the user. In 2001, Azuma et al. defined in their paper "*Recent Advances in Augmented Reality*" [11] three main categories of displays for merging the virtual and real worlds:

- Head-worn displays (HWDs), which are meant to be mounted on the head of the user, rendering specific imagery in front of their eyes. Also referred to as Head Mounted Displays (HMDs), they should be ideally not larger than a pair of sunglasses and can be further classified in optical see-through and video see-through: the former type generates an overlay on a transparent display, whereas the latter makes use of an opaque display, onto which the background for the virtual content is represented by the video captured from a head-worn live camera. A slightly different sub-category could be represented by *virtual retina displays*, which involve directly drawing on the user's retina the virtual content through low-power lasers.

- Handheld displays, which represent the category we will mostly deal with in this document. They are generally flat-panel LCD displays that provide video see-through augmentations of the real world, acting as a "window on the world" (WOW) that overlays virtual content onto the video stream of a camera.

- Projection displays, representing an alternative approach in which the virtual information is directly projected onto the real objects. This is made possible by using one or more room-scale overlapping projectors that enable visualizing virtual content for multiple users even without special eyewear, but with many types of contraints that we won't consider in our discussion.

In the early 2000s, Azuma already noticed the possible limits of MR displays, defining the properties that still nowadays technology is trying

to improve. For instance, HMDs should be characterized by a small size and weight in order to portable and be naturally worn by a user like they were a pair of glasses. Common issues among see-through displays are often related to low resolution and field of view, but also insufficient brightness and contrast [11]. At the same time, video see-through displays suffer from parallax errors, since the camera is generally mounted at a different location than the real position of the eyes, producing sometimes a noticeably different view; even focusing the eyes at a particular distance (eye accomodation) needs to be considered. Ultimately, cost has always been a huge limit for the mass production of these type of displays.

## 2.2 Registration and tracking

A fundamental concept for mixed reality is called *registration*, which involves the ability of placing virtual content in order to make it appear in its correct real-world location. In order to achieve this, a MR application needs to track the user's viewing position and orientation: the accuracy of estimating the posture of the subject is the key parameter that defines how the application can reasonably superimpose virtual information onto the real-world. We can define three types of approaches that try to address this problem:

- Visual tracking, based image processing or computer vision techniques: considering video-based systems have a digitized image of the real environment, it may be possible to detect and use environmental features to enforce registration. Visual tracking often relies on modifying the environment with the addition of fiducial markers, aimed at providing a more reliable tracking.

- Sensor tracking, involving motion sensors like accelerometers, gyroscopes and GPS but also relying on other type of measurements, like geomagnetic sensors or laser rangefinders.

- Hybrid approaches, combining the two types of tracking above in order to exploit their strenghts and compensate their individual weaknesses.

Visual tracking is based on the idea of identifying features in the environment (digitalized as a video stream), tracking their position and use it to apply corrections aimed at enforcing a proper registration [12]. However this is definitely not an easy task, as detection and matching need to happen in real time with high requirements of robustness. In order to achieve a reliable tracking through image processing, the environment may need to

be modified with "artificial" features, which we will generically refer to as "fiducials": according to Azuma [12], in the second half of the 1990s, thanks to the use of LEDs or special markers, it was already possible to achieve an optimal accuracy of up to one pixel when superimposing virtual content. Fiducials may be represented by any environmental physical element that could be used to compute the relative projective relationship between itself and the video camera, in order to enable tracking, alignment, and identification. Owen et al. list in their paper [68] some examples of fiducials along with a definition of the criterias an ideal fiducial should respect: it should support determining unambiguously its position and orientation with respect to a calibrated camera, without favoring particular orientations; it should be easy and fast to detect with simple algorithms, independently of the camera specifications; finally, it should be uniquely marked and not mistakable with other elements present in the environment. Since the 1990s, the most used type of fiducial has probably been represented by binary markers, generally characterized by a synthetic square composed by a black border with an inner binary matrix, from which it is possible to compute an identifier (e.g. a 4x4 matrix would allow using 16-bit identifiers): this type of fiducials has the great advantage of being easily trackable with simple algorithms and sufficient robustness under many lighting conditions; its main drawback is that, being artificially created, binary markers need to be printed and placed manually in the environment. A second category of fiducials is represented by image patterns, consisting in a set of flat features extracted from a given image: if these features are detected in the environment, they can be used for tracking. Common use cases for image patterns involve advertising and interactive books, where the patterns are printed like in the case of binary markers; however, natural feature detection is aimed at searching these features even in environmental elements which are not necessarily artificially created. A last important type of "fiducial" is given by 3D model detection, which tries to perform tracking on three-dimensional features, often specified through the definition of a mesh.

With visual tracking, registration is only possible when one or more fiducials are available in the video stream obtained from the camera, since otherwise we would not be able to estimate the pose of the camera. Additionally, despite visual tracking could be enough to ensure registration for some applications, it does only provide the relative locations of the objects and the camera. For instance, by knowing the relative pose of the camera with respect to a fiducial, we could render some 3D graphics on top of it; but if we wanted to know their absolute position in space, visualize virtual content not strictly related to that fiducial or allow a more complex inter-

action with multiple differently positioned virtual elements, we would not have enough spacial information about the surrounding environment.

Ultimately, some approaches rely only on the use of sensors in cases in which precision requirements are not very high or in which the aim is simply to show geolocated information. A classical example is related to the task of showing points of interest (POIs) around the user, even if some applications tried to deal with more complex geolocated content. The main drawback of these approaches is related to the sensors accuracy, which only recently is becoming acceptable when dealing with consumer hardware.

## 2.3 Interfaces and visualization

After defining the technologies needed to augment a scene with virtual content, it is also fundamental to consider how users will interact with those objects and how to effectively present information. Since we are dealing with non-physical objects, interaction may not be immediately natural for a user.

In the early 2000s, Azuma et. al [11] categorized two possible ways to interact with virtual information: one is to use eterogeneous devices, combining simultaneously the advantages of different displays according to the requirements of the application; the other option is to introduce tangible interfaces in order to provide haptic feedback and to support a more physical, direct interaction with the world. Performing natural actions in mixed reality with the use real objects and tools was originally conceived as a possible solution to fill the gap between virtual and real worlds. Over the next years, an increasing number of applications tried instead to leverage gesture and kinesthetic control, aiming at developing intuitive control mechanisms that take inspiration from human behavior [15].

Regarding the display of information, issues like error estimate visualization and data density definitely need to be taken into account. Since registration errors are often unavoidable and sometimes significant, content may be placed in incorrect locations and in many applications the user should be aware of this. One solution is generally to visually display, based on expected tracking and measurement errors, the area where the object could reside inside the screen space. As Azuma points out, another possibility is to try to make registration errors "less objectionable to the user", for example fading a virtual object, according to some probabilistic function, in case we are not sure if it needs to be occluded by real objects. If the number of virtual elements displayed increases too much, another issue is the unreadability of information on the display, that may become cluttered,

preventing the correct perception of the two merged worlds. A possible solution to limit data density is filtering information in order to keep in view the important elements; at the same time, knowledge about the environment could be used to ensure virtual content is not occluding other information or important parts of the environment, without which the mixed reality effect would become meaningless to the user.

Other research in this field has focused instead on advanced rendering techniques. Ideally, in some applications virtual augmentations should be indistinguishable from the real-world environment, and automatic extraction of information such as illumination and reflectance can help achieving higher quality renderings. However, real-time photorealistic rendering is generally unfeasible with the wearable hardware, so some trade-offs must be considered. Another interesting field of research involved in MR is related to the concept of *mediated reality* and consists of segmenting individual objects in an unmodeled real-world environment, deleting them and eventually replacing the with virtual objects.

## 2.4 Human factors and perception

Another important aspect in the design or MR experiences involves the study of human factors. Technological limitations, registration accuracy, miscalibration and physical properties of the display can greatly influence how the user perceives the use of a MR application, especially in cases in which it is required to use it for a longer amount of time. Arguably the most significant factor is represented by latency: excessive delays may cause irreversible registration errors capable of making a MR application completely unusable. Still in relation to registration errors, depth perception is very important to understand where virtual content is concretely located; some solutions to improve depth registration involve the use of stereoscopic displays and the study of how eye rotations affect perception. The performance of a task executed in MR is also greatly influenced by user adaptation to this new type of interaction and, in prolonged use, by eye strain and fatigue. All these factors need to be taken in serious consideration both for the development of the concrete MR application and for the design of the display, with the aim of creating a comfortable and at the same time efficient experience for the user.

## 2.5 Possible applications

According to the survey presented by Azuma [12], early applications of mixed reality involved inspection and industry, for instance aimed at helping assembly-line workers perform their tasks on specific hardware; medical applications focused instead on aiding doctors during surgery operations or illnesses detection. In the first years of the XXI century, the development of new mobile technologies allowed MR to be applied to navigation, providing situational awareness and contextual information retrieval, often based on geolocated content. At the same time, the first commercial applications were introduced, mostly dealing with 2D content for overlaying advertisements, rendered real-time during broadcast video of sport events. Some attemps of creating collaborative MR environments were also tried to support remote and collocated activities, generally involving entertainment or the integration of already existing tools: in this case users were meant to see the same augmentations and a "shared understanding of the virtual space" was suddenly required [12]. Finally, MR turned out to be useful for learning and providing information, but also for entertainment, as demonstrated by the real-life videogame ARQuake [70].

## 2.6 What has recently changed

In the last decade, many things have changed in the world of computer science and technology. Hardware is becoming way more efficient and cheaper, while software and drivers availability is definitely larger than ten years ago. Many new technologies have also emerged and have gradually changed how people behave in their daily life. In this section, we will briefly discuss how the the key aspects of MR mentioned earlier have changed in the last decade and how this will possibly reflect on the development of mixed reality applications in the near future.

The first enormous change has involved the display technology. While until ten years ago a 640x480 (VGA) camera resolution was considered more than enough, in 2016 many smartphone models now have 2K video resolution, while commercial HMDs like HTC Vive [40] have reached a resolution of 1080x1200 pixels per eye. The improvements made by television technology, which is now trending towards the standard of 4K resolution, have also helped in enhancing aspects that involve light emission,: since the introduction of OLED (Organic Light Emitting Diode) technology, displays require much less energy to operate, are thinner and can even be foldable, on top of providing better contrast and less eye strain.

For what it concerns visual-based tracking, great advances have been made by computer vision. With the improvement of feature-based tracking methods, MR is not required to use anymore regular fixed-dimension markers for detecting where to put virtual contexts. Many markerless feature detection libraries such as Vuforia [96] have now become publicly available, while algorithms for identifying not only pattern images but also 3D objects are being introduced. Another noticeable innovation involves the real-time extraction of features from scenes to identify the relationship between the camera and the real-world coordinate system, especially in unprepared environments. Despite the early research about *Simultaneous Localization and Mapping* (SLAM) goes back to the early 1990s, only in 2007 a first convincing implementation for monocular mobile devices was presented at ISMAR with *Parallel Tracking And Motion* (PTAM) by Kleinet al. [43]: through the use of this method, based on the concept of structure from motion, it is possible to build and update a map of the environment while simultaneously keeping track of the position and orientation of the camera. Within certain constraints, SLAM allows to completely get rid of classical fiducials in the environment, by providing a full 6D camera pose. On the trend started by PTAM, other remarkable steps in the recent history of computer vision were represented by *Fast semi-direct monocular visual odometry* (SVO) [29] and by *Large Scale Direct monocular SLAM* (LSD-SLAM) [23]. One of the great limitations of the diffusion of these approaches out of research labs involves the computational power required to run some algorithms on mobile devices. In 2007, Klein et al. showed that PTAM can be executed at 30 FPS on an Iphone 3GS, however we had to wait until 2012 to have a publicly available libray (Metaio [55]) support it for the first time. Recently, even Vuforia [96] and Kudan AR [44] have introduced their own SLAM implementation, respectively supporting plane detection and walls identification. This has finally allowed even non computer vision experts to build their own application and hopefully this trend will continue with more consumer oriented solutions, fostered by the continuously improving mobile hardware.

Regarding the user interaction, research has focused even more on the development of natural user interfaces and on improving the feeling of presence, originally belonging to the realm of virtual reality. Interacting with digital information using natural hand gestures has become much easier after the introduction of cheap depth sensors like Microsoft Kinect [58] and Leap Motion [63], relatively available even on the user market, that respectively allow to track the position of human body and hand joints in space. Thanks to these technologies it is easily possible to convey the illusion that users can directly manipulate 3D objects using their hands or interact with

them using the natural movement of their body, in an intuitive and easy to learn way [15]. The wider availability of drivers and public APIs has also contributed to make developers' life much easier when creating applications with this type of interfaces. Future research will probably involve improving the stability but also the quality of the control interfaces.

Many things have changed even for what concerns human factors. First of all, the hardware improvements of mobile and wearable processors have significantly decreased latency, enabling most consumer devices to reach at least 30FPS of framerate with the most common tracking techniques. The simultaneous enhancements of displays and VR headsets for what it concerns resolution and lighting technology have slightly decreased eye strain, while HMD design has become more lightweight and ergonomic (let's consider Google Glass [32] and Microsoft Hololens [57] for instance, which also incorporate on board processors). The commercialization of the first VR headsets for the general public has lead to improvements in comfort and in grahical user interfaces design, with the aim of making them more and more tolerable by everyone. At the same time, the success of cheap VR viewing tools like Google Cardboard [21] and the introduction in 2016 of the support for 360' videos by *Youtube* and *Facebook* have brought an even bigger change, related to people's mentality more than to technology itself: nowadays it is becoming more and more socially acceptable to use a headset or a mobile device for visualizing virtual content - and this is mostly due to the availability and cheaper price of these tools, which are also better advertised by the media (mostly in relation to marketing and entertainment). While many technologies ten years ago were still limited to research laboratories and unknown to many, now they are slowly becoming available on the consumer market (eventually with limited features) and people are getting used to hear their name. A relevant example is represented by the very recent launch of Pokmon GO [45], that in just two months has already brought SLAM-based augmented reality on the smartphones of millions of people, increasing their awareness of this type of technology.

Finally, as far as application fields, MR has really expanded its target public: many systems have matured beyond research prototypes and have lead to possible uses in engineering and architecture, for instance aimed at previewing the positioning of new buildings or furniture in an environment; marketing has greatly increased the use of MR, especially related to hyperlinking information and to associating advertisements and magazines with interactive content, with the goal of engaging users and making them more affectionate to a product. At the same time, research about collaborative MR is improving with the creation of multi-user interactive user

interfaces and hybrid types of tracking, while there is always greater interest for gaming applications, in particular after the presentation of Microsoft Hololens [57] in 2016. Another relevant potential of MR has been leveraged by digital storytelling, where narratives and information enrich in interactive ways real-world elements, often aiming at providing context-aware learning in many different fields.

## 2.7 Authoring tools

There are many challenges involved in creating meaningful narratives that rely on overlaying virtual content on top of real-world objects. Currently, interfaces for facilitating these experiences lack important functionality due both to technical challenges and to the difficulty of understanding ahead of time how users will respond to the various components that make up the experience. A primary task in MR involves determining how best to overlay virtual content on top of real-world objects or within real-world spaces. Different approaches include identifying features on objects, keeping track of user and device locations, and incorporating additional sensor data from beacons, cellphones, or cameras to increase the accuracy of placing virtual content. The problem is exacerbated in dynamic setting containing many users within complex environments, and fully understanding the real-world environment and the location of the users within it in real-time is not possible. Thus, many MR applications are event-based only, and consist of creating overlays on top of specific environmental features detected by a camera, and determine only a relative pose estimation of the device but do not accurately define the position of virtual content in space. While effective in some situations, this limits the possibilities of the MR experience and the types of interactions a player can have with virtual objects. Another challenge involves registering objects to a specified real-world position, a task that greatly depends on the tracking accuracy achievable with the current technology. That is, heterogeneous devices may have a different representation of virtual content and may be affected by different environmental conditions; because of this, MR applications generally do not allow interaction between multiple devices.

Creating MR applications thus requires an in-depth knowledge about several different technologies. Programming toolkits, such as *arToolkit* [8], are meant to facilitate the development of such applications, nonetheless require low-level skills that are often impractical for content developers to learn. Considering the potential of MR, effective authoring tools are needed so that developers and artists can quickly create and customize MR experi-

ences. A basic challenge when editing MR content is difficulty of previewing it, as most commonly the position of virtual content is related to a single device with specific characteristics. This makes it difficult to reason more generally about the MR experience more multiple users and to determine if the virtual overlay is correctly positioned onto a live video feed of a real-world scene. Many solutions to this problem involve prototyping MR experience from within the mixed reality itself. For example, Rekimoto et al. [77] propose a method based on 2D printable binary matrix markers aimed at providing landmarks to registering information on a live camera stream. The idea of using binary markers for prototyping MR application has been widely adopted since it provides reliable tracking in many situations. A downside to this approach is that is necessary to modify the real-world environment through the placement of fiducial markers. Poupyrev et al. [73] propose an authoring interface aimed at an easy and effective spatial composition where digital objects can be arranged within a small MR environment. the designer, through the use of an HMD display with a mounted camera, could see both real elements and virtual objects simultaneously and perform basic operations on them by combining the use of multiple binary markers, used as physical controllers for the interface. Lee et al. [48] extend this concept of "tangible" MR interaction by adding cubic marker-based props for performing authoring tasks in indoor environments. Höllerer et al. [39] propose instead a method for authoring outdoor MR experiences, leveraging the combination of a HMD see-through display with a handheld computer. As Etzold et al. [25] shows, recent progress in smartphone technology has enabled easier and more accurate ways to interact with and position virtual content. Langlotz et al. [46] explore on-site modification of content through interaction via a mobile device. Other alternative approaches include work by Haringer et al. [38] which extends the Microsoft PowerPoint XML protocol for defining the behavior of an MR application. Yang et al. [101] explore how to leverage mobile phone gestures to perform editing in small environments. However, as noted by Hampshire et al. [37], the development of a MR application is often a long and non-intuitive task which imposes significant limitations on the creative expression of the designer. Instead of using an mixed reality environment to author MR experiences, we propose instead a virtual reality editing environment. This allows us to get rid of the use of markers entirely and to utilize a spatial representation of content through a partially modelled representation of the environment, making the development of MR applications independent of the tracking issues encountered in previous solutions.

# Chapter 3

# Mobile Application

In *Chapter 3* and *Chapter 4* we will show all the steps involved in our our general approach, describing also eventual requirements or environmental constraints take we have taken into account during our work. After providing an overview of how our method is conceived and implemented, we will deal with more real-world examples in *Chapter 5* (*Case Studies*) by applying our approach to the two applications *Chicago 0,0* and *Digital Quest*.
As previously mentioned, our work involves both the creation of an innovative authoring tool for geo-located mixed reality and the development of the actual mobile application that we will use to produce concrete results and comparisons with other previous approaches. From now on, we will refer to the former as "*editor application*" (or "authoring tool") and to the latter as "*mobile application*" (or "mobile framework"). Our discussion will start in this chapter by presenting our mobile framework and then continue in the next chapter with the authoring tool; we will demonstrate during our dissertation how they are strictly related and how the need for an editor application arises from the advantages (and at the same time the complexity) provided by the mobile application.
During our work, we used for our implementation many different technologies, spanning from writing C++ plugins for handling mobile sensors to using WebGL [64] wrappers or higher level programming environments such as Unity [92]. We will try to approach problems first from a code-agnostic point of view and then we will eventually compare the technologies that we concretely used for our personal implementation, meaning that anybody could implement in many different ways the concepts that we are going to present. In our case, considered its easiness of dealing with 3D graphics and the need of having both native access to sensors and a possibility to deploy cross-platform applications, we decided to implement our work mostly with

the Unity development platform and for this reason our discussion will be mainly oriented to a certain type of development process. We will clarify the cases in which we used different technologies, such as Three.js [89].

## 3.1 Mapping the two worlds

Before starting to discuss specifically our mobile implementation, we want to do a brief introduction about our idea of "mapping the two worlds" one to each other, a recurring concept in our work that involves both the mobile and the editor application. This idea is strictly related to the classical implementations of location-based augmented reality, in which every augmented content is associated to specific geo-coordinates in the real world. As it is possible to see in Figure 3.1, in our work we want two define two parallel worlds, the real one (where the user is located), characterized by positions on Earth and concrete objects, and a virtual one, containing all the content that does not really exist but that we would like to add to our real-life experience. By overlapping the two worlds with an acceptable accuracy, defining a sort of one-to-one mapping between them, we are able to make real world elements interact with virtual elements, a concept that we like to call "merging the two worlds".

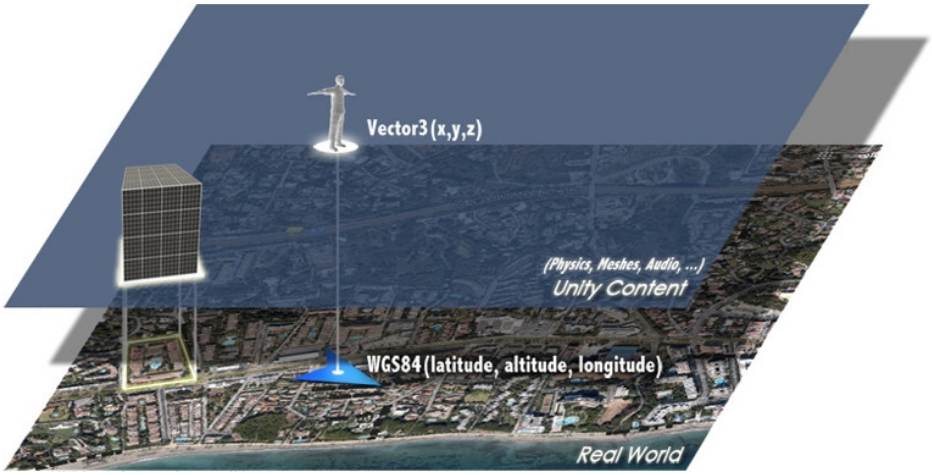


*Figure 3.1: Mapping the two worlds. While a user walks in the real world, a virtual camera moves and rotates accordingly in the virtual scene, defining which elements need to be rendered on top of the live camera stream of the player's device [53].*

In our case, we refer to the Earth coordinate system provided by the standard World Geodetic System 1984 (also known as WGS84 [65]), commonly used in cartography and in navigation - in particular by the GPS

system. Overall, what we need is to find a correspondence between Earth coordinates to virtual units, passing from latitude, longitude and height to a representation in the 3D space as a (X,Y,Z) vector. In our case, we define the (X,Z) plane to represent the ground and the Y axis perpendicular to that plane and increasing with the height, which is defined as heigth from the ground in that specific position (and not from the sea level for instance). In order to define orientation, in our virtual world we assign the geographic North to correspond to the positive value of the Z axis. Our solution is to define a fixed point on the ground plane as the origin of our coordinate system and then compute all the other points in relation to that one. The (0,0,0) point in the virtual world could correspond, for example, to the initial location on Earth (latitude and longitude) of a user who is using our application. So, after storing our initial WGS84 coordinates as $(lat_0, lon_0)$, we can calulate the horizontal position of other elements with the following formula:

**Data:** Latitute and longitude of a location
**Result:** [x,z] offset from the origin
$x_{offset} \leftarrow \frac{lon * 20037508.34}{180} - x_0$
$z_{offset} \leftarrow \log\left(\tan\left((90 + lat) * \frac{\pi}{360}\right)\right) / \frac{\pi}{180} * \frac{20037508.34}{180} - z_0$
**if** *initial coordinates* **then**
  $x_0 \leftarrow x_{offset}$
  $z_0 \leftarrow z_{offset}$
  **return** [0,0]
**end**
**return** $[x_{offset}, z_{offset}]$

**Algorithm 1:** Coordinates to virtual units conversion, by extending and generalizing the algorithm proposed by MapNav Geolocation Toolkit [53].

Using the above solution, we obtain an equivalence of 1 unit in the virtual world for every 100 meters in the real world. Considering that we would like to operate on smaller scale content and the fact that Unity is less accurate with small numbers (especially regarding clipping planes), we decided to multiply all positions by 100 on all the 3 axis, obtaining a correspondence of 1 unit per 1 meter. We will show in the next chapter with some real examples the level of accuracy that can be obtained with this method, that is considerably affected by floating point approximations and distances, since it deals with projecting Earth's shape onto a plane. However, considering storing values as doubles and assuming the application will not generally be used on a very large area with respect to the initial point, this method provides good results for our intended purposes.

After having mapped positions and rotations in the two worlds, we can represent our real-word user as a camera in the virtual world, assuming a position equivalent with the one of the user and a rotation equivalent with the orientation in space of the user's device. In brief, the camera in the virtual world moves with the user and represents the real mobile camera of the device: if the user walks North by 5 meters, the position of the virtual camera will optimally be increased by 5 units along the $Z$ axis; if the user looks left and rotates his mobile device accordingly, the virtual camera will as well rotate counterclockwise by 90 degrees around the vertical $Y$ axis. By enforcing on the virtual camera the same parameters (e.g. the field of view) of the real one, we can simply render onto the video stream provided by the mobile camera what is currently seen by the virtual camera, creating a sort of moving "window" on the virtual world that we will leverage to generate the mixed reality effect.

## 3.2 The dual camera approach

According the the principle of "mapping the two worlds" just presented, our mobile application is meant to estimate the position and orientation of the device in the world, eventually rendering the corresponding virtual content on top of the live video stream provided by the mobile camera: it ultimately represents the medium through which user are able to experience mixed reality. Differently from other augmented approaches that solely rely on geolocation or on marker-based tracking, our hybrid implementation combines different types of tracking in order to achieve a reasonable estimation of the camera pose in a wide range of situations. In particular, we propose a method that we called "dual camera approach", based on the use of two virtual "helper cameras", that do not concretely exist but act as placeholders for the values computed by a same number of tracking methods. In *Sections 3.2.1*, *3.2.2* and *3.3* we will demonstrate how this approach is applicable to different tracking algorithms and extensible with new techniques in order to improve its performance in specific situations.

As we mentioned earlier, our approach aims at determining the absolute pose of the camera in an "always-on" fashion, meaning that the lack of tracking features in the environment can reduce the accuracy of our estimation but should not interrupt our continuous tracking. On top of the live camera stream, we want to render the output of the main virtual camera, which is characterized by a position and a rotation in 3D space and whose field of view matches the one of the mobile device. This camera moves and rotates in the virtual space as the user walks in the real world, creating a sort of

parallel world that coexists and overlaps with the normal one, enabling the augmented reality effect.

The absolute pose of the camera is computed by weighting the parameters of two helper cameras (hence the "dual-camera" approach), defined in the virtual world as follows:

- the *ARCamera*, whose pose is computed from the tracking of predefined features found in the live mobile camera stream;
- the *SensorCamera*, whose position and rotation in space are defined only through the sensors available on the mobile device.

In order to provide a clearer explanation and not to confuse similar terms, we will refer to the hardware mobile camera as *MobileCamera* and to the virtual camera that renders the final augmentation as *MainCamera*, whose position and orientation in space will be computed by weighting the parameters of the two helper cameras.

### 3.2.1 ARCamera

In the case of the ARCamera, we used a common markerless approach to estimate its pose when a fiducial is detected through the real camera of the mobile device. Though, as we show in the next paragraph, our method is also appropriate for other kinds of pattern-based tracking algorithms, our current implementation stores a pattern object with the feature points of a user-defined image by using the ORB descriptor-extraction algorithm. In order to match the feature points of the pattern with the live camera input, our pattern detector performs several steps using the grayscale version of the input image: (1) descriptors are extracted from the live stream video frame for the detected feature points and are matched against pattern descriptors; (2) ratio tests are used to remove outliers and the homography transformation is found using inlier matches; (3) a warping step allows the computation of a refined homography that is then multiplied by the previous one to obtain a more precise transformation; and finally, (4) the pattern corners are transformed to the image coordinate system to make pattern locations on the input image. On top of this computation, we take into account the position, orientation and dimension in the real space of the image provided as a fiducial, thus allowing absolute positioning of the ARCamera.

As an example, imagine we are defining as a fiducial a flat, vertical image of a panel attached to a wall on a particular building. That panel would be characterized by a location in space, by its height from the floor and by its instrinsic width and height. The markerless approach stores

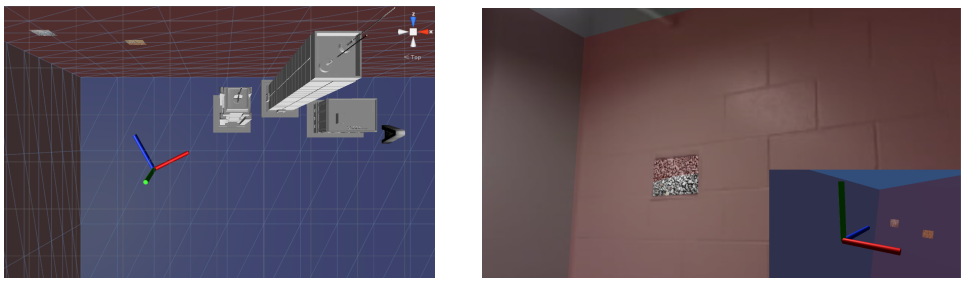

*Figure 3.2: On the left, the position and orientation of the mobile device is shown at real-time within a partial representation of our laboratory, identified by planes indicating floor and walls: this pose corresponds to the ARCamera and is computed indoors only by tracking (one of) the two fiducials on the upper wall. The image on the right shows instead what the device camera is actually seeing (i.e. the wall with the fiducials) plus a semi-transparent representation of the walls, whose overlapping with the real ones can be used to check the estimation accuracy.*

relevant features of that image and tries to match them with what the mobile device camera is seeing. If a match is found, we can compute the pose of the ARCamera relative to that panel. If we did not consider the additional information we provided about the panel, then we would not be able to determine the correct case in which it is placed vertically on a wall (and the camera is looking horizontally) from the case in which it is placed horizontally on the floor (and the camera is looking down). However, since we have these details about our fiducial, we can instead calculate the correct distance from the object and absolute pose of the camera, which we know should look at that specific fiducial in the world from a particular angle and distance. Moreover, our technique supports detecting multiple fiducials simultaneously, a feature that is useful for maintaining a correct camera movement once a tracker is lost. The main limitation of the ARCamera is that it is enabled only in presence of a tracked pattern: if the user doesn't have geolocated fiducials within the field of view of his mobile camera, we are unable to estimate its pose solely from the parameters of the ARCamera.

### Markerless AR

In this paragraph we will present how we implemented our own marker-less augmented reality system, demonstrate also that our approach is independent of the computer vision algorithms used for estimating the pose of the camera. Thus we will deal both with already existing libraries and with our custom implementation, that will show the basic steps necessary for this type of applications. Our approach could be used also with binary markers,

but we decided to focus on marker-less AR since it uses the actual environment as input and will be later leveraged in our two case studies.
During the last decade, many AR libraries have been developed: some of them had a very short lifespan, while some others were more succesful and also produced Unity plugins on top of their original C++ or Java code. Among those, we decided to test Vuforia [96] and ARToolkit [8]: the former, developed by QUALCOMM [74], is at present the most popular and most used worldwide for many types of commercial applications, while the latter is mostly used in academic research due to the fact its code is completely open-souce. We decided not to use Metaio [55] due to its recent acquisition by Apple [6] and the shutdown of its services. Both libraries have their own Unity plugin and provide many different functionalities, but in this work we are interested only in marker-less, pattern-based augmented reality. In particular, our goal is to position the camera in 3D space according to its relative pose to a real planar image (so we will not consider objects with complex shapes), a feature provided by the two libraries with different terms of use and limitations. The third and last library that we decided to take into consideration is *Kudan AR* [44], which, despite less-know, in the last few years has achieved great improvements and has recently become more popular.

*Figure 3.3: Sample image uploaded to Vuforia Target Manager: on the left we can see the original picture, on the right the image is turned to grayscale and features are rendered on top of it.*

In Vuforia's documentation, what we previously defined as *fiducial* is referred to as *Image Target*. In order to enable other services like cloud image recognition, Vuforia expects the developer to manage Image Targets from its website: the image that we want to track is uploaded and receives a rating, which expresses the richness of features that can be tracked for that specific image. For what concerns natural features detection, Vuforia's

Image Targets are expected to be rigid (non flexible) and non-shiny; highly textured images generally contain stronger detection and tracking ability, whereas images with poor contrast, organic shapes or repetitive patterns generally provide poorer results [95]. In Figure 3.3 we show a simple image and its relevant features according to Vuforia. Despite Vuforia's Target Manager could be useful to keep a mobile application lightweight when a huge amount of Image Targets need to be registered, in our case it is very limiting since it forces us to rely on an external online service that does not allow any type of customization, and also generates a new downloadable Dataset file which needs to be reimported in Unity every time and cannot be modified. The use of the library is normally intented for rendering a 3D mesh on top of the recognized image: the content appears when the image is tracked and disappears when the target is lost. Despite Vuforia's API acts as a non-extensible "black box" and offers a very limited API, it allows us to retrieve at least the pose of the camera and if our fiducial has been detected or not. Unfortunately, no information about the accuracy of detection is provided, but still it represents an easy way to obtain the camera pose. Other issues that we need to consider when using Vuforia in Unity are the fact that its library locks the camera field of view and clipping plane to default values that cannot be modified, posing some additional limitations when rendering object that are too close or too far. In order to solve this, we created our own camera in Unity configuring manually the clipping planes and retieving the FOV form Vuforia's projection matrix.

Ultimately, Vuforia has another big limitation: it is not possible to apply image pre-processing on the live input from the mobile camera before it gets analyzed by the tracking algorithm. While this could not represent an issue on general purpose applications where targets are simple printed images, it can make the tracking very unreliable in case of Image Targets that do not comply with Vuforia's ratings - we will show a concrete example of this when we will discuss our case study *Chicago 0,0*.

This was the main reason we also considered the use of ARToolkit, more appealing by the fact that it is completely open-source and its C++ code can be modified and build with custom personalizations. In ARToolkit's documentation, the feature we are interested in is called *Natural Feature Tracking* (NFT) and is mostly oriented towards planar textured surfaces. Like Vuforia, the provided images should have a reasonable amount of fine detail and sharp edges, with a low degree of self-similarity and high spatial-frequency; large areas of single flat color with blurred or soft detail will probably not track well [9]. ARToolkit provides a standalone external tool to train images, whose code is publicly available (so we know which algorithms

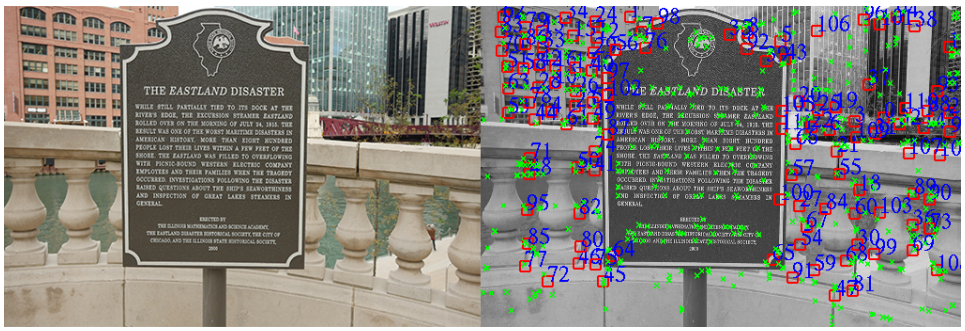


*Figure 3.4: Visualization of the NFT Dataset generated by ARToolkit on a sample input image. The features outlined with red squares are used in continuous tracking, instead the ones with a green cross are used to initialize tracking [9].*

are used by the library) and allows us to personalize parameters such us the size of our fiducial and the desired target DPI resolution. By modifing the C++ code and building it for the single target platforms, we would have complete personalization of the code, especially for doing pre-processing on the mobile camera input stream. However, understanding the source code of the library is not straightforward and time constraints limited our exploration of ARToolkit internal workflow, especially because for our work we are mostly interested in deploying crossplatform application (currently we only used AndroidNDK [4] for building for Android). Overall, even personalizing the properties associated to the fiducials, we obtained results very similar to the ones obtained with Vuforia. The only great advantage was, in our case, the possibility to retrieve a percentage of accuracy while detecting a fiducial, that could be used as an index for taking decisions at runtime.

In relation to our case, the library Kudan AR provides two advantages: one is the introduction of a discretely reliable implementation of the *FastSLAM* algorithm [62], which we will leverage in *Section 3.3* to extend our method; the other one consists in its markerless image-based tracking feature, which does not rely on a cloud based solution nor on external services like Vuforia. Unfortunately the pattern images are not configurable like in ARToolkit and the library itself is not previewable in Unity, however it may turn out useful in cases like *Chicago 0,0*, where we want to keep the threshold for recognition low: according to our tests, Kudan AR identifies patterns in a much wider range of environmental situations than Vuforia (limiting the problem of oversampling that we will describe in *Section 4.2.3*), however provinding a slightly less robust tracking.

The last implementation we tried is completely custom and relies on

OpenCV [67], an open-source computer vision library that allows access to many useful algorithms for AR. Currently, we tried both a direct implementation in Unity with a C# wrapper, that links dynamically to the Java and iOS versions of OpenCV, and an Android-specific implementation in C++, built for the mobile architecture thanks to Android NDK. In the former case performances are a bit degraded comprared to building a native library by using the OpenCV C++ implementation, but the code is cross-platform and easier to maintain. Despite for most of our tests we used Vuforia due to its simplicity, we will show anyway the steps performed by our algorithm, that could be very useful for customizing the application in future works. The current implementation takes inspiration from the book written in 2012 by Baggio et al. [13].

Our first step is to apply image recognition in order to search the mobile camera video input for a particular bitmap pattern, that should be detected even if it is scaled or rotated. The approach should possibly be brightness-invariant and use a grayscale version of the input frame. Since comparing directly the pixels of the pattern and the test image is infeasable due to perspective transformations, we need to find "interesting" parts of the images (that we will call *features*) and try to match them. Starting from the image we want to use as a fiducial, we need to apply a feature extraction algorithm, that generally leverages corner or edge detection or the derivatives of the image gradients. For each feature detected, we store its center point, radius, orientation (which will allow us to have a rotation invariant image detection) and score (a value indicating the "quality" of the feature point). For simplicity, we will call feature points "keypoints" from now on. Among the most famous algorithms we can cite SURF [14] and SIFT [50], but they are patented and not available for free commercial use; we will use instead ORB [80], a modified FAST [79] feature detector that has fixed keypoint size but is able to estimate keypoint orientation. It is generally very used on mobile devices due to its lower CPU requirements. From each keypont we extract one fixed size vector called *descriptor*, that allows us to consider a much smaller amount of data when comparing two images; the algorithm that we use as a descriptor extractor is generally the same that will be later used as a feature detector. In order to store the data of a train image, we define a data structure called *pattern*, holding the list of features with their extracted descriptors and two vectors of correspondences between 2D and 3D pattern positions.

The next step consists in matching the feature points of a train and a test image (respectively our pattern image and video frame), finding correspondences in a way very similar to searching the nearest neighbor from

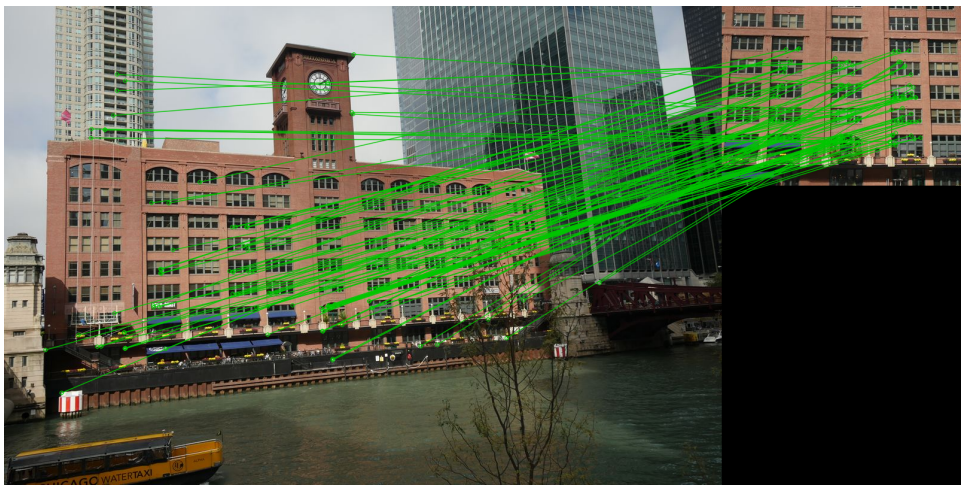

*Figure 3.5: The matching algorithm tries to find the raw correspondences of the train image on the right in the test image on the left.*

one set of descriptors to each element of the other set. In our case we used the OpenCV implementation of the so called *Flann-based matcher*. During this step, some mismatches can happen: false-positives represent wrong correspondences between keypoints, whereas false negatives represent missing matches between points that are on both images. Though we cannot do much about false negatives, we can minimize false positives by applying techniques like *cross-match filtering* and *ratio testing*. A second pass of outlier filtration is done by applying the RANSAC [28] algorithm, that we use to find the homography transformation from points of a pattern to the test image coordinate system: considering the reprojection error calculated by the algorithm for the homography matrix, some other keypoints may be marked as outliers. Ultimately, our result can be refined by warping the test image by leveraging the homography previously estimated and try to find more accurate pattern corners. By multiplying the old homography with the new refined one we generally obtain a more precise final homography matrix. Baggio et al. [13] in their book suggest to consider a pattern correctly identified if at this point at least 25% of the patter features are found in the test image.

The last step is estimating the pattern pose and solving the so-called *PnP problem* by leveraging 3D-to-2D point correspondences. In our case, we simply assign four 3D points to the corners of our rectangle input image and use the OpenCV *solvePnP* function to get the final pose of the camera relative to the test image. The camera-intrinsic matrix, necessary for obtaining a natural-looking augmented reality effect and holding parameters

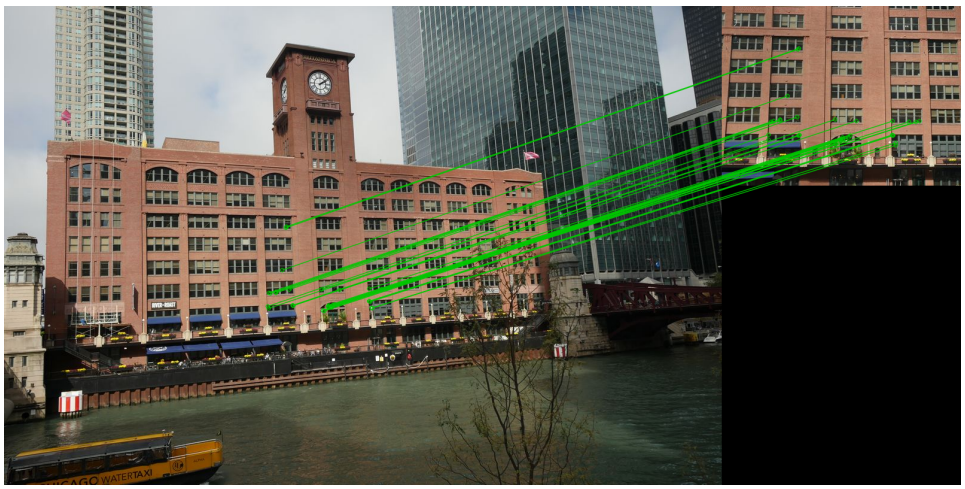

*Figure 3.6: Refined matches after applying cross-filtering and the RANSAC algorithm.*

such as focal length, distortion coefficients and principal point, is currently computed offline by recurring to Vuforia's API. In Figure 3.7 we show on the test image the final reprojection of the pattern image.

This last method represents the classical approach to marker-less AR. It also represents a good way to understand how already pre-packaged libraries work and a possibility to build our own code without having dependencies on external services that may be sooner or later discontinued. Despite we used this as a general-purpose approach (with less convenience in comparison with Vuforia and ARToolkit), its main advantage is that in future implementations we could extend it in order to make it more specific to particular situations, like building recognition for the *Chicago 0,0* case.

### 3.2.2 SensorCamera

Differently from the ARCamera, he SensorCamera does not rely on any computer vision algorithm, but uses instead the data provided by the internal sensors of the mobile device - allowing us to track the movements of the device independently from the visual features present in the environment. In particular, the compass, accelerometer, gyroscope and A-GPS information are combined in order to retrieve both position and orientation in absolute coordinates. Most current smartphone devices have their own native sensor fusion algorithms that can compute orientation in the coordinate system of the device, mostly by leveraging the gravity vector, the geomagnetic field, and the rotational acceleration (whose drift is sometimes already natively corrected). For this reason, we provide fallback countermeasures to support

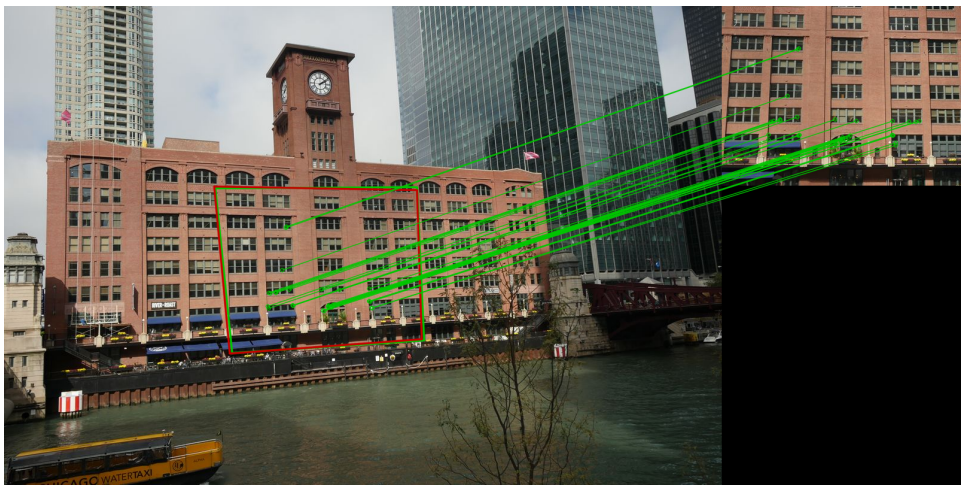

*Figure 3.7: Final matches after homography refinement. On the test image on the left we can observe the reprojection of the pattern image on the right.*

older devices, which may however have limited degrees of freedom if more sensors are absent. Regarding instead the position of the camera, we convert the position retrieved from the GPS to virtual units. The requested horizontal accuracy can be dynamically set in order to try to obtain more accurate measurement based on necessity, even if real accuracy depends heavily on the instrinsic characteristics of the surrounding environment. Though our current implementation does not comprehend sensor fusion techniques to smooth GPS information in combination with accelerometer and gyroscope, any technique could be applied on top of our method (e.g., visual odometry [29], step detectors [102], or multisensor odometry [19]) to improve horizontal positioning. An intrinsic limitation of the SensorCamera is that it loses one degree of freedom on the vertical axis and needs to be set to approximately the height of the user since GPS vertical accuracy is very low (notice that our system uses height relative to the ground, not relative to sea level). Just as the ARCamera may be disabled when no pattern is detected, the SensorCamera loses horizontal positioning if the GPS becomes unavailable (e.g., indoors) or can have a incorrect heading if magnetic fields influence measurements, even if this issues can be mitigated by sensor fusion. Optimally, the two helper cameras would be completely overlapped, having the same position and rotation. Unfortunately this is not so common in real life experiments and in the next sections we will describe our approach to exploit the information available in order to define the MainCamera that renders the augmented scene.

### Orientation in space

With Inertial Measurement Unit (IMU) we generally refer to an electronic device that integrates more sensors in order to estimate the force and angular rate of an object, and sometimes also the magnetic field that surrounds it. Every smartphone nowadays is equipped at least with a three-axes accelerometer, which is mostly used to track motion and to compute the device orientation (e.g. landscape vs portrait mode): for instance, by analyzing and filtering the raw data provided by this sensor, it is possible to compute the linear acceleration of the device in a particular direction, detect significant motions, estimate the "Up vector" by taking into account gravity and also provide simple methods for step detection and counting. The second relevant sensor generally present in an IMU is the gyroscope, which provides the angular rotation of the device along its three axes, helping us to understand better its rotational behavior and relative orientation. Each device has generally its own hardware specifications for these sensors and often provides to the developer a way to access both raw and pre-computed values, to which the owner company may have applied filtering algorithms to remove bias and drifting inaccuracies. By combining the above sensors with a magnetometer (also known as "digital compass"), which provides the orientation angle with respect to the North pole, it is possible to estimate the absolute orientation in space of the device as a tridimensional vector $(rotX, rotY, rotZ)$. In our case, we considered both *Android* and *iOS* smartphones: the former allow the developer to access this information through the so called *Rotation Vector*, which requires us build a specific native plugin for Unity; the latter exposes directly a public interface to Unity, easily accessible through the *Gyroscope.attitude* attribute. Since the two type of smarthphone use different types of axes reference, in our application we had to convert the iOS orientation in order to correspond to the Android one; additionally, we implemented a fallback function for Android devices that tries to combine all available sensors when one of the required ones is missing, in order to achieve orientation even on older devices (but possibly loosing one degree of freedom during rotations). A very useful sensor fusion library that we used under certain circumstances is Google Cardboard SDK [21], that is generally used for VR applications but, thanks to our implementation, can eventually be used for determining the orientation of the SensorCamera after tracking has been lost, by manually aligning the Cardboard values on the vertical axis: since this library does not make use of the geomagnetic sensor and computes only relative rotations, it could be useful for our extended tracking algorithm in situations with significant magnetic interferences.

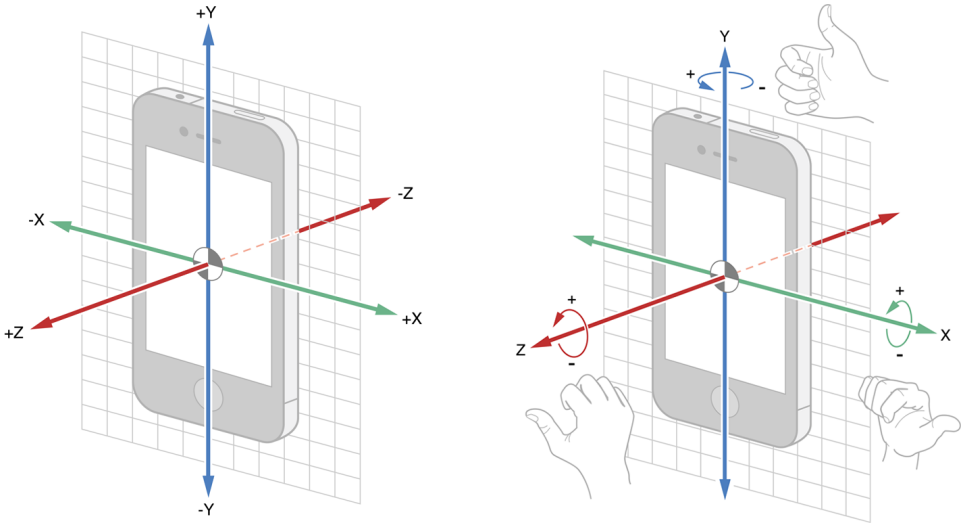


*Figure 3.8: Axis references for the accelerometer (left) and the gyroscope (right) measurements on an iPhone [5]*

### 3.2.3 Camera pose estimation

To combine the information provided by the two helper cameras (ARCamera and SensorCamera), we need to implement a dynamic way to intelligently estimate the final pose of the MainCamera. Our algorithm takes into account specific situations where one of the two cameras is preferred over to the other or where both cameras are merged in order to verify their consistency. By remembering that the SensorCamera is always enabled, while the ARCamera activates only in presence of a pattern to be tracked, we can define the following four primary situations: fiducial found, fiducial lost, multiple fidcucials, and no fiducials.

#### Fiducial found

When the input video stream matches the pattern of one fiducial, the ARCamera activates and assumes a position in space calculated considering the coordinates and size of that fiducial, which, without considering the intrinsic accuracy of the tracking algorithm, may represent the greatest source of accuracy. If we assume those properties are correctly set by the creator of the application, the ARCamera pose can be considered in this case more reliable than the one provided by the SensorCamera. After a few frames in which the ARCamera stabilizes its position, we store the difference in orientation between the two cameras as $\triangle r = r_{ar}^{-1} * r_s$, where $r_{ar}$ and $r_s$ are respectively the rotation in space of the ARCamera and of the Sensor-

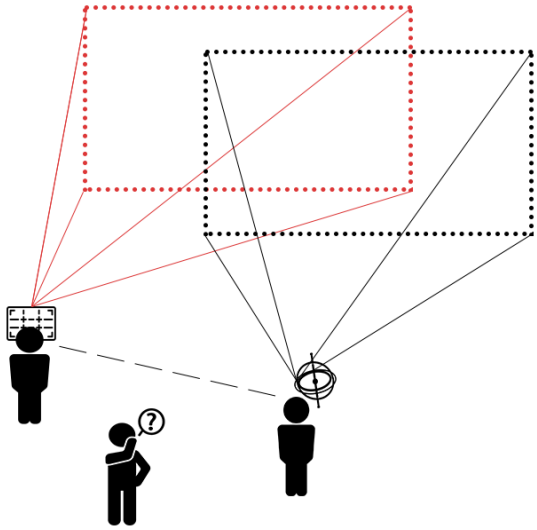

*Figure 3.9: Our goal is to estimate the absolute position and orientation of the camera of the mobile device in space. In order to achieve this, we try to combine information from two helper (virtual) cameras: the ARCamera (left), based on image tracking, and the SensorCamera, which relies on mobile sensors. Optimally the two cameras would have the same pose, but in all other situations we need to intelligently weight their contribution to the final pose estimation based on the current context.*

Camera. Under the assumptions above, the MainCamera pose is set to be the same of the ARCamera, which in our tests proved to be more reliable than the SensorCamera, as the sensors could be affected by magnetic fields and GPS inaccuracies. However, there are cases in which interpolating with specific weights the poses of the two helper cameras could give better results. For example, when the dimensions for fiducials are not so accurate or when there are false positives applying the tracking algorithm. These tracking errors can be found simply by splitting the $\triangle r$ matrix over the three axis and considering how much the ARCamera rotation differs from the SensorCamera (i.e. if the former is looking forward and the latter is facing down, probably there has been a false positive in tracking and the augmented content should not be rendered). Our algorithm enables flags for each object in order to set its behaviors in case of known environmental issues in that area.

### Fiducial lost

When a fiducial is lost, the ARCamera is disabled and we need to rely on the SensorCamera for extended tracking. As far as rotation, the $\triangle r$ matrix

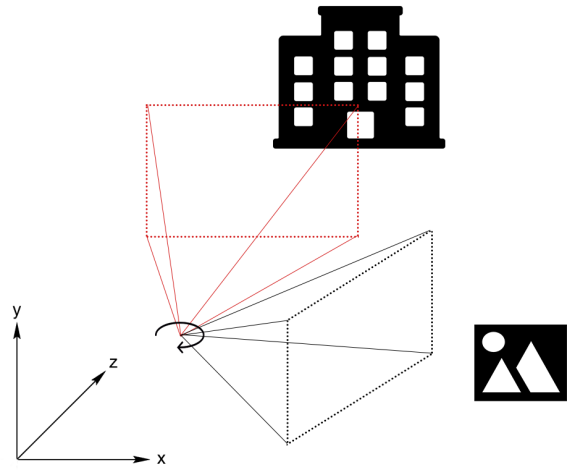


*Figure 3.10: The $\triangle r$ matrix can be imagined as the difference between the rotation of the two helper cameras, as if they were in the same position in space. In the example above, the user rotates the device to his right and looses the tracking of a building. While the ARCamera becomes inactive, we can leverage the SensorCamera and the $\triangle r$ matrix to know how much the user rotated from the direction in which tracking was lost. This way, considering the absolute position in space of elements, we can still render virtual content nearby the user even if no tracking information is available. At the same time, we can evaluate the coherence of what we are rendering by analyzing both how the two helper cameras differ from each other and how they are located and oriented in 3D space.*

is used for maintaining the correct orientation of the main camera even if the tracking is lost or the two helper cameras are not correctly aligned (especially in case of magnetic interferences): the new main camera rotation is simply computed as $r = r_s * \triangle r$, guaranteeing a smooth continuation of the movement performed by the user. By filtering out eventual sudden rotations around the global vertical axis, we can prove this extended tracking solution is not affected by compass calibration problems. It is very useful to leverage this situation in cases in which we have few good fiducials that are not in the direction of which we want our augmented content to appear. Let's consider the user is in a poorly textured area where the only reasonable fiducial is a building, which is 90 degrees left to the object we want to show: in this case we can direct the user to look at the building, correct his position with the ARCamera pose computed from the fiducial and then make him turn right to the virtual object; he will probably lose tracking but by relying on the SensorCamera the algorithm will allow the object to be seen anyway, taking into consideration the delta rotation the user has performed since he lost the tracking Figure (3.10). This situation works under the assumption that the

user simply rotates the device around without moving. If, after a fiducial has been lost, the user starts walking around, the main camera position can be updated basing on available sensors: if GPS is available with an acceptable accuracy and refresh rate or if other sensor fusion techniques are available for the SensorCamera (e.g. step detection, visual odometry), the user can get closer to the augmented content with varying performances; if the horizontal movement is not retrievable from the SensorCamera, we are not able to predict his movements and so will need to make the content disappear with a transition when a relevant linear acceleration is detected.

#### Multiple fiducials

Our algorithm supports tracking multiple markers at the same time. The ARCamera pose is computed from the first pattern detected until its tracking degrades (or the fiducial goes out of screen), then it is automatically switched to the other available ones with the same smoothing function mentioned earlier. These multiple tracking images are positioned with relative accuracy in the virtual world of the application environment. This is particularly useful for keeping an accurate camera pose and delay. It is also fundamental for indoor pose estimation, since in that case the GPS signal would not be available (even if that solution could be solved by other means, for instance by using bluetooth beacons). If more than one tracker is detected at the same time, the main camera will still be locked to the ARCamera, otherwise it will consider relying on the sensors for extended tracking until the new pattern is detected.

#### No fiducials available

When no fiducials are available, the algorithm may behave differently based on different settings. In some cases, we would like augmented content to be precisely positioned only relative to a fiducial and not be affected by potential sensor inaccuracy. According to this situation, the algorithm does not show virtual elements if no fiducials is used for pose correction, even if the user is in the correct position to see the object in space. An example could be when we want a user to focus on a building used as a fiducial and then direct him to look to his right and make the object appear. Maybe we prefer the user to see the content only from that point of view and with a certain accuracy, and not to show it if the user is simply roaming around. This case is also meaningful in the creation of a narrative, where the order of which virtual content is displayed is meaningful.

On the other hand, we could simply make the MainCamera correspond to the SensorCamera when no fiducials are available. This would mean that the user location will be used for showing nearby virtual elements, with all the already mentioned limitations in case we are not exploiting sensor fusion for improving horizontal accuracy. Obviously this feature can be performed outdoors only if a GPS signal is available and indoors only if other methods (e.g. beacons) are used for estimating an absolute position in space. Another option involves showing content based on its size and distance from the user: like with classical POIs in location-based augmented reality, we could, for example, allow only the display of distant content in order to limit the effect of sensor inaccuracy.

## 3.3 Smoothing and filtering

A recurring issue with the markerless approach is the instability of the camera pose. Even when a pattern is identified and the mobile device is kept still, the ARCamera will never stay in the same exact position. These small movements can sometimes be considered acceptable if the purpose is to render an augmentation on top of the detected pattern. In our case, the more distant a virtual object is positioned from a fiducial, the more the inaccuracy in the camera pose estimation will influence the overall positioning of elements in space. This is why it is fundamental to smooth the movements of the virtual camera so that if the device is not moved the object will remain in their position. However, simple smoothing is not sufficient since there are moments in which the movement of the camera needs to be instantaneous, such as when a fiducial is first detected. In these cases smoothing would result in a virtual object entering or leaving the scene too slowly, since the ARCamera could have been potentially just enabled after having previously been inactive in a completely different position.

User movements introduce further considerations. We can expect users to move their device slower when they are focusing their attention on a virtual object, but they will probably put down their device quickly as soon as they are done with it. So smoothing the movement in this case would leave virtual objects on screen for more time than necessary. We define a threshold for the allowable delta movement that triggers the smoothing feature; if the user moves too fast, smoothing is simply not applied. For the ARCamera, smoothing is applied both through linear and spherical interpolation both to camera position and orientation respectively:

$$pos_{smoothed} = pos_{old} + (pos_{current} - pos_{old}) * \frac{k_1}{dist} * time_{frame} \tag{3.1}$$

$$rot_{smoothed} = rot_{old}(rot_{old}^{-1} * rot_{current})^{\frac{k_2}{dist} * time_{frame}} \tag{3.2}$$

Here, *pos* represents the position as 3D vector and *rot* represents the orientation as a quaternion; $k_1$ and $k_2$ are two constants that are divided by the distance from the current fiducial, since the more distant we are the more smoothing factor is required; and $time_{frame}$ is the time needed to render the previous frame. This method averages positions over time also to prevent unexpected instantaneous movements that can happen for few frames, especially if the detector erroneously sees the pattern as flipped or inconsistent.

As for the SensorCamera, our current implementation does not yet provide by itself additional sensor fusion techniques for improving the horizontal accuracy, so we apply a permanent smoothing of the GPS position very similar to the one performed for the position of the ARCamera, but without considering distances from eventual objects in the scene.

## 3.4 Dynamic resource management

Knowing the spatial location of the virtual content and of the fiducials allows us to manage the resources of the mobile device more wisely, thus limiting energy requirements that could quickly drain its battery. Pattern matching and tracking are known to be operations that are computationally onerous. With our method we know precisely when a user is near to a fiducial and thus we can choose to activate the feature only in certain circumstances or on a limited number of patterns, instead of processing continuously the input video stream and trying to match it with the entire database of pattern descriptors. Similarly, only virtual content close to the user will be loaded into the scene.

On the other hand (at least for outdoor applications), this method requires a prolonged use of GPS, which could be responsible for quickly draining the battery of a smartphone. In order to avoid this, we propose a method for dynamically requesting a different accuracy and refresh rate to the A-GPS, which relies also on cellular network and Wi-Fi stations for positioning. Given the first coordinates of the user and knowing his distance from the virtual content, we can decide to keep GPS requirements low, for instance by enabling only the cellular network and Wi-Fi station triangulation: this

would not provide a good accuracy (which would anyway be useless considering we don't have content to show), but would still allow us to know if the user is in a significant position for the overall narrative. Even when the application is tracking a pattern, since the main camera mostly assumes the properties of the ARCamera, no particular GPS accuracy is required. However, when tracking is lost and the user starts moving or when the user is approaching some virtual content, we need to demand an improved accuracy and refresh rate, which the operative system and hardware will try to provide if possible. This way we will utilize the energy intensive tasks only in the proximity of content in order to have better precision as far as horizontal positioning.

## 3.5 A user-centered framework

By leveraging elements from both markerless and location-based AR we are able to obtain many advantages that will allow us to offer a better experience to the user. This can be enabled by the presence of multiple tracking sources which guarantee more data about the environment, and that we can then constrain selectively thanks to our own knowledge of the site and the expected behavior of the users.

As we mentioned earlier, having an absolute geolocation for overlays and fiducials allows us to load and unload content based on user location, pruning the dataset of pattern images to be considered and at the same time freeing memory on the mobile device, guaranteeing a straightforward form of dynamic resource optimization. This allows also the display of content even if no camera tracking is available, assuming environmental characteristics allow us a good accuracy from sensor estimation. Since the application knows the real position in space of the virtual content, we can also enable different behaviors based on proximity to the user or distance between related overlays.

Estimating the absolute orientation in space of the mobile device and computing the $\triangle r$ matrix enable us to derive important information about the behavior of the user. For instance, we can know how far and in which direction a user is moving away from a tracked pattern, and, if desired, direct him to go back to that specific object. Otherwise, we know at which orientation the user can find other overlays and we can direct him towards them instead. This allows us to render content on screen even if tracking has been lost or is not available at a specific orientation. For example, we can direct a user towards a reliable fiducial and then tell him to rotate by a certain angle in order to find virtual content that otherwise we would not

have had the possibility to show due to the lack of tracking features in that particular direction. By leveraging absolute orientation data we can also detect incoherent situations, trying to avoid cases in which content is shown when it shouldn't (false-positives) or vice-versa (false negatives); we are also able to know the presence of potentially overlapping virtual content, giving the user the possibility to visualize only parts of it at a time. Moreover, we can produce intelligent camera stabilization, providing a run-time optimized smoothing to avoid jittery movements.

We will discuss in *Chapter 5* how these advantages are reflected in particular on the overall user experience of *Chicago 0,0* and how they enable the creation of smarter interfaces, both for the users and the designers. In particular, we want to note again that our aim is not to improve existing tracking techlogies, but to combine them with the goal of improving the overall experience offered to the user. In the next section we will show instead a recent extension to our dual-camera approach, that shows how our approach can be generaled by the addition of a new camera representing a different type of tracking.

## 3.6 The triple camera approach

Thanks to the dual-camera approach presented in the previous section, we are able to combine mobile sensors and the recognition of predefined patterns in order to obtain a continuous camera tracking. Hovever, there are situations in which, in absence of recognizable patterns, the accuracy given by the SensorCamera alone is not sufficient and we would like some form of robust tracking that does not rely on a prior knowledge of the environment: it may be useful to collect and track relevant features collected at run-time from the surrounding area, of which we assume no prior knowledge - a possible answer to this requirement is represented by Simultaneous Localization And Mapping, better known as *SLAM*.

Since the demonstration performed by Klein et al. in 2007 [43], many advances have been made in the development of SLAM techniques [29] [23], but almost none of them reached the smartphone consumer market: they mostly remained confined to research laboratories or applications involving Robotics. Only in 2014, Metaio library [55] proposed a first publicly available feature implementing Klein's PTAM algorithm, which had already been proved to run smoothly on some mobile devices seven years earlier. Through this function, after the user had performed a small horizontal calibration movement with his smartphone, it was possible to detect a plane onto which place virtual objects: by creating at runtime a map of features of

the environment, incrementally added to the ones used in the initialization step, Metaio's implementation allowed every developer to completely get rid of fiducials for the first time, giving the possibility to position virtual content in totally unprepared situations. This feature had the big limitations of not working well in poorly textured environments and of requiring the user to initialize manually the algorithm. Little time later, Metaio presented an improved 6DOF SLAM technique aimed at tracking the movements of user in room-scale environments, however it remained available just for a short amount of time until the company was acquired by Apple. Vuforia [96] relatively recently proposed its *Smart Terrain* feature, which represents a simplified SLAM implementation aimed at detecting a plane and eventually some 3D props placed on it. However, only in 2015 Kudan AR [44] finally introduced a library supporting an efficient implementation of the FastSLAM algorithm, allowing on high-end devices a decent markerless tracking with a framerate of about 20-30FPS, according to our tests on a LG G3 device. The advantages of monocular SLAM on smartphones are huge, since they guarantee a completely markerless approach that is executed at runtime on the currently available features, thus performing consistently with environmental changes as lighting and weather. Despite Kudan's implementation is still not completely robust and sometimes has a noticeable latency of nearly 50ms, we considered very useful to apply it to our approach in order to improve it in two main situations:

- SLAM has proved to be efficient in room-scale environments and, assuming the presence of well textured environmental features, it can be used for small movements of the user in all the cases in which we do not have visual tracking information, i.e. when we are relying solely on the device sensors. Normally we would need to make use of other sensor fusion algorithms, since GPS positioning definitely has not enough refresh rate and accuracy to allow the user a realistich way approach closely to a virtual object. In this case, it would also allow exploring on mobile devices new types of interaction in which the user directly interfaces with virtual content in more natural ways, for intance with hand gestures and body movement.

- Our tests in downtown Chicago showed that SLAM can also be useful for incrementing the accuracy in the estimation of the camera pose by tracking distant or lighting-critical environmental features that it would normally be impossible to track with the pattern-based image recognition approach.

This is why it can be very helpful to define a third "helper camera" in our approach, to be taken into account for the final camera pose estimation in the same way we did with the other two helper cameras. Since the new *SLAMCamera* requires a much more demanding computational power with respect to the previous two helper cameras, we can imagine it as an auxiliary camera that gets enabled and comes to help only in determined situations in which it could efficiently contribute to the overall user experience. We have currently identified the following enabling cases, that respectively address the two situations mentioned above:

- When we are relying solely on the SensorCamera because of the absence of fiducials in the environment, we wait for the user to arrive at a reasonable distance from a virtual object and then we activate SLAM to make that object stick to that position. In this case the content keeps the same position even in precence of sudden GPS errors and the user can get close to the object thanks to the small movements computable with SLAM; the content would probably not be tied exactly to the desired location, depending on the horizontal accuracy at the moment in which SLAM is activated, but if the object is allowed to appear within a certain area this approach is completely reasonable. Alternatively, with the same assumptions, it is possible to activate SLAM and then insert the object in the nearby location which is most suitable for tracking.
- When dealing with fiducials, SLAM can be activated as an additional way to extend tracking, like we did when considering the relative rotations of the SensorCamera. For instance, when a fiducial is lost, it can be activated to estimate the relative pose of the camera to otherwise untrackable environmental features, allowing us to compute how much the user has moved away from the fiducial.

In *Chapter 5*, after having focused mostly on our dual-camera approach in the first two case studies, we will consider in a dedicated section how the addition of the third helper camera can bring advantages to both applications *Chicago 0,0* and *DigitalQuest*.

### 3.6.1 SLAMCamera

As we previously mentioned, our implementation makes use of the Kudan AR [44] library. Among its main features we can list its incredibly fast initialization time, that does not need the pre-registration of the environment to be tracked, and its performance in low-textured environments and

in difficult lighting conditions. As a simple comparison, the previous SLAM implementation proposed by Metaio [55] and based on PTAM [43] required the user to perform a predefined movement in order to calibrate the device camera and, differently than Kudan, was incredibly unreliable in proximity of walls or desk surfaces.

Kudan framework is based on the idea of keeping a fixed virtual camera in space (aimed at rendering the scene), around which content will be moved and rotated based on tracking. However, this behavior, despite acceptable for very simple applications, is definitely non compatible with our concept of mapping the two words: in our case, in fact, the camera moves according to the movements of the device, while content has an intrinsic position, independent from tracking. In order to make Kudan useful to our purpose, we need to perform the following steps:

- We leverage the SensorCamera to retrieve the orientation in space of the device based on sensors and we associate it to the current live camera video.
- We use Kudan plane detection feature to identify a reference plane, whose relative position and rotation in space are computed by the library based on arbitrary features and geometrical properties gathered at runtime.
- Normally Kudan framework would suggest to place content directly in the reference system of the detected plane, that is moved and rotated around a fixed point (Kudan rendering camera) according to the estimated pose of the device with respect to environmental features. Instead of relying on the plane anchored in camera space, we consider the position of the rendering camera in the reference system of the detected plane: by translating and rotating it in order to make it correspond to the position in which the content is placed in the global coordinate system, we obtain a transformation matrix tbat allows us to define the pose of our SLAM camera with respect to the detected plane.
- At this point, we can simply ignore the previous need of moving objects around the rendering camera - since now we know the position of our SLAMCamera with respect to real-world features detected during the algorithm initialization.

This way, the SLAMCamera now moves around the reference system identified by the algorithm: our next goal is to make that reference system

correspond to the virtual content we want to show, otherwise the behavior of the SLAMCamera would still be independent from the geolocation mapping we defined in *Section 3.1* and just render content at arbitrary positions. In order to better understand how the SLAMCamera is positioned in 3D space, we need to consider that the SLAM algorithm finds a reference system which projected origin in normalized screen coordinates corresponds to (0.5, 0.5) and which orientation with respect to our XZ plane is dictated by the gravity sensor of the mobile device: in brief, if we wanted to render the plane found by the algorithm, it would have its center in correspondence of the center of the device screen and an orientation optimally coinciding with the floor that we would normally consider using the SensorCamera; the main differences consists in the orientation of the detected reference system along the global Y axis and the distance between its origin and the SLAMCamera, which depends on perspective and has the contribution of making rendered objects appear smaller or bigger. Thus, a non-precise alignment of the detected reference system would cause virtual corrent to be rendered with incorrect rotation or scale. Up to now, We have found two possible solutions to this problem:

- Let's consider a situation in which no fiducials are available and only the SensorCamera is enabled. By leveraging the facts that the algorithm returns a reference system whose origin corresponds to the center of the screen and that the computation can be run at at least 20Hz (according to our tests on an LG G3 device), we can wait for the user to get close enough (within a predefined radius) to the virtual content that we want to show. At that point, we wait for the SensorCamera to align to the content within a certain angle $\alpha$ and then we start iterating the SLAM algorithm until the estimated pose of the SLAMCamera resembles the one of the SensorCamera, so that they would singularly render content with the same scale. This process can be performed by considering a proximity threshold or defining the position of the MainCamera by interpolation. Once this happens, the MainCamera simply renders what the SlamCamera is seeing without loss of continuity.

- The second method is simply to ignore the problem and let the SLAMCamera be enabled at an arbitrary moment in which we would like to increase tracking accuracy - for instance when a fiducial is found or lost or when virtual content is close. In this case we don't mind the absolute position in space of the SLAMCamera, but we want to leverages its relative movements and rotations. We fundamentally reuse the

same idea we described in *Section 3.2* for combining the SensorCamera and the ARCamera, by computing a $\triangle r$ matrix encording relative rotations. Differently from the case of the SensorCamera, we can now consider even relative position modifications, thanks to the tracking sensibility of the SLAMCamera. In brief, by recalling *Section 3.2.3*, we redefine our fallback method to be ARCamera ¿ SLAMCamera ¿ SensorCamera for the estimation of the pose of the rendering camera (MainCamera).

In both previous cases, we defined when the SLAMCamera is enabled but not when it is disabled. Our test show that, after anchoring to the original environmental features, the algorithm often looses tracking after rotations of the mobile device that go beyond 100°, especially if they are fast (similarly to the ARCamera). Sometimes it is able to reckon, but completely loosing accuracy on tracking, causing rendered content to scale wrong or to shift significantly in position. This is why when we detect these situations we use the same method of the $\triangle r$ matrix we presented in *Section 3.2* for determining the pose of the MainCamera, leveraging difference in rotations detected by the SensorCamera, much less sensible to fast rotations. After the SLAMCamera has been disabled, it can be re-enabled according to the behaviors explained above.

Despite the main part of Chapter 5 is dedicated to our dual-camera approach, we consider the addition of the third camera a significant improvement, especially for enabling new user interaction possibilities - which we will discuss in *Section 5.4*.

# Chapter 4

# Authoring Tool

As we demonstrated in the previous chapter, our method may bring many different advantages in comparison to other classical augmented or mixed reality approaches, in particular with respect to the user experience. However, despite its apparent simplicity, it requires to define rigorously the dimensions and the exact location of the virtual objects that we want to appear in the world and, most of all, of the fiducials that we will rely on for tracking. Without a precise definition of these elements in space, the computed pose of the camera would be largely affected by inaccuracy and virtual objects would not appear in the correct position to create a realistic MR effect. On top of these specific needs, we would generally like to have some sort of authoring tool for designing our MR experiences. Like we mentioned in *Chapter 2*, the creation of MR content has always suffered the lack of an easy-to-use editor that could satisfy the definition of rigourous behaviors in mixed reality. Relatively few tools have been oriented towards general-porpose MR production, and most of them were related to specific company constraints or limited in the number of customization possibilites. Among them, many are mostly intended for defining triggers and information to be displayed for location-based AR (e.g. a POI to be shown in correspondence of a geo-coordinate) or for establishing which 3D mesh should appear when the device detects a particular pattern image. Almost none of them allows a spatial definition of virtual objects, especially in relation to each other and, more importantly, provides ways to preview the mixed reality experience as a whole with sufficient realism and accuracy. Overall, we would like to have an authoring tool with the following basic characteristics:

- Ease of use, so that each one could learn how to use with little effort, in order to make the creation of MR experiences available to everyone. Many previous works were in fact intended for people working at a

specific project and required accurate knowledge about the actual implementation technology. On top of this, it should allow the creation of content in a natural way, with the secundary goal of speeding up the entire development process.

- A good level of abstraction, able to encompass multiple aspects of MR: it would be very useful to define a unique tool for designing those that are the moment are often considered different types of MR (such as location-based, marker-based and natural feature detection based). Just to give an example, if we currently wanted to define the behaviour of a 3D mesh appearing on top of a picture we could use Vuforia [96], but, if in the same application we also wanted to define a tag based on a real-world location, we would need to integrate another library such as Wikitude [100], eventually with the need of creating a different application. At the same time, we would like our authoring tool to provide a high level interface independent from the actual implementation used on the client device. We would like something that could define MR exeperiences to be used on smartphones as well as on HMD devices. In order to allow this, we would probably have to define a common protocol to be used by applications for loading the MR content.

- Customization. This broad concept may refer to several different aspects intended by the different flavors of MR experiences, but in our case we will consider common customization features like the possibility to insert different types of augmented content and define their position, size and orientation in space or in relation to a fiducial. Later in this chapter we will also deal about customizing the appearance and the behaviour of virtual content in relation to the user and to other virtual objects.

- Possibility to preview the mixed reality experience before its deployment, simulating what a normal user would see during its execution. This would allow a better design, making run-time adjustments available based on the real user experience and also speeding-up the development process. It would be relevant for remote testing in particular, since one common disadvantage of MR applications is that they often require a significant amount of time for onsite testing.

- Modularity and extensibility, in order to eventually add new functionalities in the future for supporting new types of interactions in MR.

In this section, as a continuation of the method we presented for defining our mobile application, we will show our approach to the creation of a MR editor respecting the characteristics mentioned above. We will try to explain the reasons behind each implementation decision and try to highlight the main advantages of our method. Differently from previous works, our spacial approach leverages its similarities with virtual reality in order to provide a more realistic authoring experience. Our discussion will be focused on our current implementation in a CAVE2 [27] hibrid reality environment, but at the end of the section we will also deal with porting the editor to the new VR headsets, such as the Oculus DK3 [66] and the HCT Vive [40]. In addition to implementing the requirements previously listed, we will also expand our research to new interaction possibilities made possible by our approach.

## 4.1 The CAVE2 environment

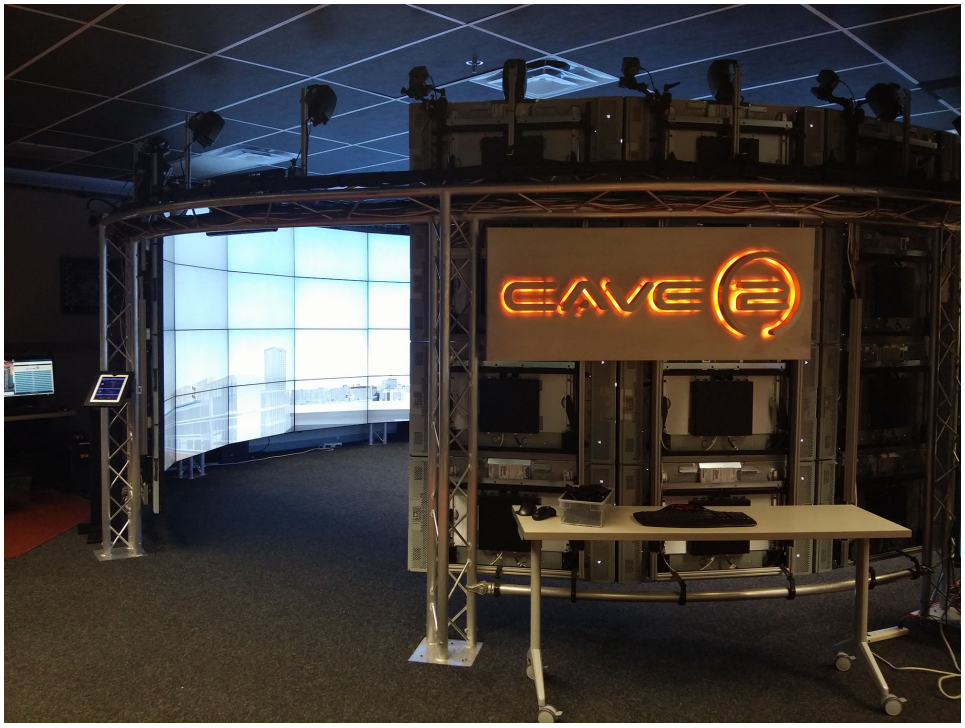


*Figure 4.1: A picture of the CAVE2 environment that we used for implementing our method. This technology belongs to the Electronic Visualization Laboratory (EVL) of the University of Illinois At Chicago [27].*

CAVE2 [27] is a hybrid reality environment developed by the Electronic Visualization Laboratory of Chicago in 2012 ad is the successor of the orig-

inal CAVE [17], developed by the same laboratory in 1991. CAVE2 is an immersive system characterized by high resolution tiled display walls that enable visualizing both two and three dimensional information, providing great flexibility for mixed media applications spanning from virtual reality to scientific visualization. The system has an 8 feet tall cilindrical shape with a diameter of 24 feet, consisting of 72 "near-seamless, off-axisoptimized passive stereo LCD panels" [27], for a total coverage of approximately 320 degrees of panoramic environment. CAVE2 is able to display information with a resolution of up 74 Megapixels in 2D mode and is characterized by 20/20 of (horizontal) visual acuity. It is designed to support multiple operating modes, including virtual simulation and hybrid interaction, creating new opportunities for users to collaborate using both 2D and 3D. Regarding the underlying hardware, the system is built with a 36-node high-performance computer cluster, where each node takes care of controlling two displays; among these, one node acts as a master and is directly accessible to developers. Other specifications involve a 20-speaker surround audio system, that can be used for generating 3D audio, an optical tracking system composed of 10 IR cameras and a public internet interface with 100-Gigabit/second bandwidth. In particular, the camera system is capable of tracking in space different types of user-defined markers inside all the circular inner space, providing their position and orientation in space. Regarding the software architecture available to programmers, the system is based on Omegalib [91] - a middleware software that enables the support of OpenGL, OpenSceneGraph and Vtk. Only recently, a wrapper for Unity has been introduced in order to enable additional interaction possibilities.

Our choice of using CAVE2 for implementing our authoring tool is based on various considerations related to the high resolution and to the interaction possibilities offered by this technology. The space inside CAVE2 is large enough for making the user feel comfortable during the design process, having the possibility to move or to sit on table collaborating with other users. At the same time, the almost circular panoramic view can be seen with 3D perspective with easiness of movement, by rotating the head like we would do with a VR headset or by moving and turning around. Compared to other technologies like Oculus Rift [66], CAVE2 induces less stress in the user and allows longer development sessions. On top of this, by using a pair of special 3D tracked glasses, the 3D perspective can be adjusted according to the position in space of the user, that can come closer to the interested content and perform editing with greater accuracy thanks to the high resolution of the screens. The system offers millimetric-precise tracking features, giving the possibility to track the movement of the user and most of all of eventual

wireless controllers, that we will exploit for interacting with the system. Ultimately, a Kinect v2 [58] is positioned at the bottom-front of CAVE2, with the possibility of tracking the joints of up to 6 users inside the environment. Regarding the software implementation, we decided to leverage the Unity 3D wrapper provided by Machdyne [54], the company that currently commercializes CAVE2. In order to run a Unity application in CAVE2, the executable needs to be uploaded to all the 36 nodes of the cluster, which are synchronized by the master node: overall, 36 application instances will be running at the same time, communicating with each other through the high-bandwitdh LAN. Each instance will have a duplicate of the main camera of the instance running on the master node, but with a different orientation: in a configuration file, all the positions in space of the screens are stored and each camera renders the scene in such a way to display on a specific pair of screens the correct content for that position. This causes a set of limitation in the use of cameras and user interfaces in Unity, but does not influence anyhow the results of our work. Another important implementation detail that we had to consider involves syncronizing calls to procedures on the nodes of the cluster: for example, if we are creating an object in a random position, each node of the cluster will compute its own different value, thus leading to the presence of up to 36 different objects in the scene. In this case, the master node needs to take care of the computation and perform an RPC call to the other nodes, specifing the already computed position for the new object.

### 4.1.1 Setting up the virtual scene

The main idea behind the development of our editor is the same about which we dealt at the beginning of the previous chapter: creating a mapping between the virtual and the real worlds. The concept consists in recreating in CAVE2 a virtual environment resembling the real one, to which we can add our own digital content. This environment, as we did for the mobile application, is characterized by 1-to-1 scale with the real world: this means that, with a certain approximation, a concrete object of width 2 meters needs to be represented in the virtual environment as an object of width 2 units and vice-versa. This is very similar to what we showed in *Section 3.1*, with the only difference that now the real world is not represented by a live camera stream anymore, but by digital content, like in a virtual simulation. So one first issue is that in this environment both real and virtual elements will have a digital representation as 3D meshes and will need to be somehow distinguished. An important prerequisite for our approach to work in the

*Figure 4.2: The 3D model of the city of Chicago that we will use for our experiments. The scale of the model is approximately 1 to 1, so that one unit in virtual space corresponds to one meter in the real world.*

best conditions is having a 3D representation of the real environment that we want to augment, so that it is possible to see how our content is conretely positioned in space. Despite this could seem a very strong assumption, the trend of the current years expects 3D models to become more and more available to the public. Obviously models with a higher accuracy would allow more precise editing, but script-generated models from open map providers could be a good start; other alternatives could involve loading meshes from Google Earth [33], Bing Maps [56] or external sources. In our specific case, we relied on a 3D texturized model of the entire city of Chicago, that we used both for the *Chicago 0,0* and the *Digital Quest* experiments. By loading the model on top of a specifically generated map of Chicago and scaling it accordingly, we achieved a rough 1-to-1 scale with the real environment, of which we will provide numerical accuracy results in the next chapter. Though this solutions may not seem easily scalable, in *Section 4.6* we will show how our research comprehends also a prototypal web-based editor that leverages Google Street View imagery [36], without the need for 3D models, and we will demonstrate how the approach to the problem does not change.

## 4.2 Defining virtual elements

As a first step, we need to select and classify which type of content needs to be represented in our editor application. The main idea is to have a representation of the real world in which we could add our augmented virtual content, so we will need to include both a 3D recontruction of the real environment and a representation of what we previously defined as virtual objects. Our goal is to allow the designer to insert the virtual content in the scene so that a user, when running the mobile application, will see that content in the corresponding real world location defined by the designer. In simple words, we can imagine the mobile camera of the user as directly connected to a virtual camera moving around the scene built with the editor, leveraging the real to virtual world location correspondences but rendering only the virtual content: in fact, the user will not see the 3D representation of the world that we have in the editor, but the live video stream from his mobile phone.

In addition to the content that will concretely provide the "augmented" effect, we also need to deal with fiducials, that will need to be accurately positioned in space according to their real-world position, in order to provide a higher precision in the pose estimated by the *ARCamera*. Indeed, as explained in the previous chapter, our approach relies on geolocating both virtual content and fiducials with the highest possible accuracy.

### 4.2.1 Types of elements

As mentioned above, excluding the 3D representation of the environment, we define two main categories of elements inside our editor: virtual objects and fiducials.

Virtual objects may represent every kind of content we would like to add to our MR experience. In particular, our current implementation enables the user to insert the following elements:

- Bidimensional images, oriented in the 3D space, that the user might be able to see only from a specific position defined by the angle of view. For instance, we may want an image to be seen only from one side and to appear transparent from the other one. Alternatively, images could also be defined as billboards, rotating as the user moves in order to always face him.

- Bidimensional videos, loaded from internal storage or streamed through a web server; the same consideration we mentioned for images holds for videos, since they represent bidimentional content positioned in a tridimensional space.

- 3D meshes, including both static and animated tridimensional models. Two examples could be including respectively a new virtual building near to already existing ones and adding a butterfly flying in a contrained area of a park in the virtual scene.

- Spatial audio content, that will need to be played according to the position of the user: for instance, in the mobile application we may want the audio to increase its volume when the user gets closer to its source; we may also want to make so that the current orientation of the user influences the ways the audio is played, as to make him more aware of the direction to take in order to reach its source.

For what concerns the fiducials, we define them as elements belonging to the real world that allow us to determine the position of the user: knowing their position in space enables the placement of virtual content with respect to them. We currently defined two types of fiducials:

- Bidimensional pattern images used for tracking, the same one ones we dealt with when we defined the behavior of the *ARCamera*. In the virtual scene, they are represented in a very similar way to the bidimensional images representing virtual content: the main difference is while the latter will be rendered as visible content on the mobile

application, the former will never be visible to the user and will only be used for tracking. Pattern images can indeed be seen only by the designer in order to allow him to position and orientate them in space to improve tracking.

- Placeholders for bluetooth beacons, which are physical devices that can be put in the real world to allow an estimation of the position of the user indoors or simply enable custom behaviours based on the proximity of the user to the device. Despite, in our case studies we will not deal with great detail the implementation of this feature, we will demonstrate how our approach represents an abstraction able to comprehend even this type of technology.

As we will show in the next paragraphs, our approach is intended to be extensible: so, new type of virtual content or fiducials could be added in the future in order to broaden the available ways to display augmentations inside the application.

### 4.2.2 Protocol definition and basic architecture

In order to allow the display of virtual content and the correct definition of fiducials, we need to define a common protocol for each of these two categories of elements. This protocol will need to be adopted both by the editor application, which will have to store the information about the scene, and by the mobile application, which will have to load the content and the behaviors defined with the editor. Despite we could treat virtual content and fiducials in a very similar way and even if they may appear almost undistinguishable in the editor application, we preferred to maintain a clear distinction when storing the data involving the MR experience, mostly due to their very different purpose.

As we already highlighted many times, the first common characteristic of virtual objects is their position in space, which is mapped to real world coordinates. Considering the 1-to-1 correspondence we created between the real and the virtual world, it may seem legit to define their position as a $(X, Y, Z)$ tridimensional vector. However, this approach results to be unfeasible, since our mapping works in relation to the initial coordinates given by the mobile or by the editor application. This means that every time we build our MR experience, the center of the virtual world system will be placed in a different position and so objects positioned at a specific geolocation will always correspond to different positions in the virtual system. That is why we need to store the position of our virtual content with the usual latitude

and longitude value in a bidimensional vector $(lat, lng)$: when the scene is loaded, those coordinates are transformed to virtual units; when the scene is modified or saved, the virtual units are converted back to geolocations. Instead, for what concerns orientation, we defined our virtual X axis to always match the geographic North direction; at the same time, the Z axis is parallel to the Equator and the Y axis is along the direction that goes from the center of the Earth to the current position, so it is always perpendicular to the surface. In this case, we can define absolute rotations around these axis that do not require further translations: so we can store orientations simply by considering the rotation of virtual content with respect to the virtual coordinate system. While in our application we leverage the use of quaternions for rotations, we decided to store this information as Euler angles, so as a tridimensional vector denoted as $(rotX, rotY, rotZ)$. Another common property of virtual content is its size: we want to store if an object is large 2, 5 or 10 meters and we want to render it on screen so that the perspective reflects these dimensions. Since in our system we can measure easily virtual units and we have a direct correspondence with real worlds measures, we could potentially store this information in meters. However, since every 2D or 3D asset in Unity is characterized by its own size even before its instantiation inside the scene, it turns out to be much easier to store the scale of virtual objects with respect with their original size. For instance, when a virtual object is added in the editor, its scale is set to the identity vector $(1, 1, 1)$, which may correspond to different dimensions based on the original properties of the asset, that may have been created with different external pieces of software. By calculating the bounds of the mesh in the editor, we can always compute and show to the designer how many meters each object measures in the three dimensions; still, it is easier to let the designer modify the size of the object and store the modifications in the value of its scale, which is computed natively by Unity. For this reason, we decided to store the size of objects as the tridimensional vector $(scaleX, scaleY, scaleZ)$.

On top of position, orientation and scale, which characterize every virtual object, many other properties are specific of the type of content that we want to show. For example, the designer may want to add different behaviours for audio and video content: while an audio should be played automatically when the user is at least $x$ meters distant from its source, we would eventually want a video to start playing only if the user arrives in front of it and clicks on it with his finger; we may also want to define if an audio is only played once or generates a loop, or decide the way its volume is influenced by the position and orientation of the user (e.g. linear or exponential decay). For 2D images, we may want to set the angle defining

the correct perspective from which the content needs to be seen. It could be useful, also, to leave the designer the possibility to define the activation of specific behaviours and animations for 3D meshes. Since the supported types of content and interaction may be extended in the future based on the needs of users and designers, we decided to define a flexible data structure so that the editor uses only the minimum amount of information needed to edit the MR experience and the designer can further customize elements by inserting its own modifications. Thus, each object is characterized by one numerical *type* attribute and by a variable number of parameters that are specific of that type. The implementation of the mobile application will need to take care of how to handle this information: when loading the list of objects to insert in the MR experience, it should first interpret the type of the content and then render it dynamically based on its specific properties. For instance the mobile application, when loading an object of type audio, should implement its own logic to create an audio object in the specified position in space, and then apply the optional properties defined by the protocol for that type of object (e.g. decay, volume, activation distance, play mode...).

Additionally, we should be able to specify to the mobile application how to retrieve the content that it needs to render on screen: it may be already available on the device or it may be required to download it from an external server, then eventually caching it locally. For this reason, we added a field *id* that uniquely identifies an object and a field *media*, that indicates where the mobile application should search for an asset if it is not available on the local storage (e.g. it could provide the *URL* from which to download it).

For what it concerns the definition of fiducials, we used the same approach. They are commonly characterized by position, orientation and scale, but may belong to different types with different specific properties. For instance, the estimation of the ARCamera pose may take into account eventual parameters defined for pattern images, whereas beacon placeholders can be characterized by their activation distance (in RSSI level). As before, our first concern is to leave to the designer the maximum flexibility of extending the behavior of the application, guaranteeing a simple way of adding additional variables that may be taken in consideration at runtime on top of the already existing ones.

Considering all the requirements mentioned until now, we decided to store all the information in JSON format, which is human readable and easily modifiable, but also widely supported. On top of some system information, the JSON file contains a simple list of the virtual objects and of the fiducials that need to be considered during the MR experience, along with their

specific properties. In the code snippet below, it is possible to see a very simple example of configuration file. In the chapter *Case studies* we will show how it can be easily extended to provide more advanced features based on the specific requirements of an application.

```
"data": {
        "title": "MR Example",
        "last_edited": 1460851200,
        ...     //other generic information
        "objects": [{
                {
                "id": 1,
                "asset": "InfoPanel", //content to load
                "type": 1, //bidimensional image
                "position": {41.867272, -87.675434}
                "rotation": {90, 37.4, 0},
                "scale": {2, 2, 2},
                "properties": {
                        "angle": 30,
                        "area": 10,
                        "visibility": "fading",
                        "transparency": 1
                },
                "media": "",
                "cache": "no"
                }, {
                "id": 2,
                "asset": "3DAudioEffect", //content to load
                "type": 0, //audio content
                "position": {41.867687, -87.654148}
                "rotation": {0, 0, 0},
                "scale": {1, 1, 1},
                "properties": { //specific of this type
                        "volume": 0.8,
                        "distance": 30,
                        "decay": "exponential",
                        "directional": "yes"
                        "visualize": "no"
                },
                "media": "http://.../assets/file_audio.ogg",
```

```
                "cache": "yes"
        },
        ...
        ],
        "fiducials": [{
                "id": 1,
                "asset": "BuildingFacade", //associated fiducial
                "type": 0, //pattern image
                "position": {41.867496, -87.673975}
                "rotation": {90, 22.7, 0},
                "scale": {3.4, 3.4, 3.4},
                "properties": {
                        "smoothing": 3.5,
                        "reliability": 1,
                        "distance": 30
                }, {
                "id": "BuildingFacade", //associated fiducial
                "type": 0, //pattern image
                "position": {41.867496, -87.673975}
                "rotation": {90, 22.7, 0},
                "scale": {3.4, 3.4, 3.4},
                "properties": {
                        "smoothing": 3.5,
                        "reliability": 1,
                        "distance": 30,
                        "threshold": 0.25
                }
        },
        ...
        ]
}
```

In the first simple architecture we propose, we rely on a server storing the configuration for the MR experience and in multiple clients requesting the configuration file to load virtual content locally. In particular, the mobile application and the editor application interact with the central server in the following way:

- The editor application is used to create and edit the MR experience. The information about the position and the eventual behavior of virtual content is stored in a JSON configuration file, which is uploaded to

the central server. When the scene needs to be edited again, the editor downloads the configuration file from the server and loads it, enabling additional customization of the scene; once done, a new configuration file is generated and re-uploaded to the server.

- The mobile application contacts the server and, if necessary, downloads the configuration file and interprets it. For each virtual object and fiducial present in the configuration file, the application loads needed assets and inserts them into the scene. After this moment, the mobile application checks for appliable behaviors based on the properties defined in the JSON file. If the mobile application has the requirement of not needing internet connection, the configuration file can simply be stored locally and loaded from memory each time the application is run.

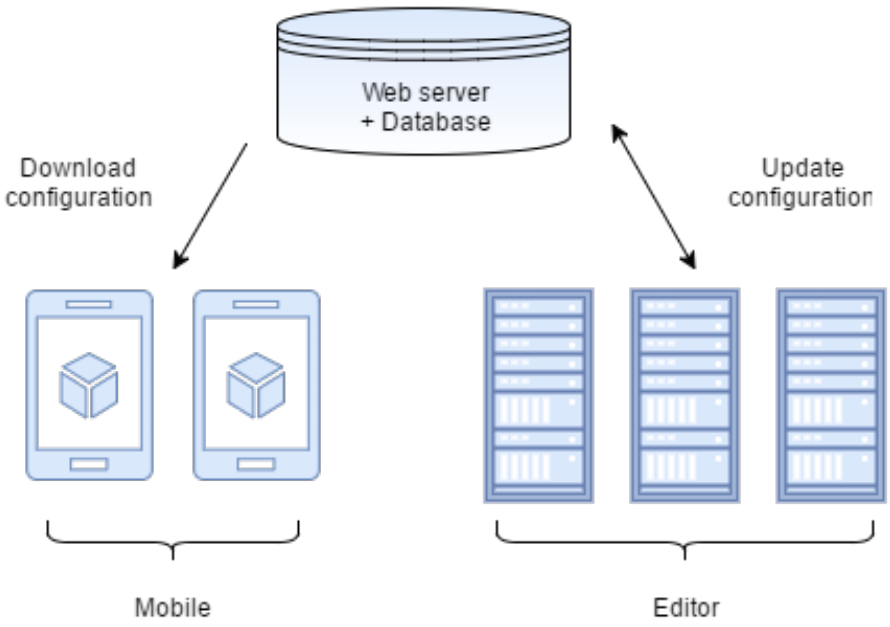


*Figure 4.3: In a first possible configuration, the editor application running in CAVE2 uploads the configuration file to a separate web server. The mobile devices can then connect to the web server, request the configuration file and finally load the virtual content inside the MR application.*

In our implementation of this very basic architecture (Figure 4.3), the server was running APACHE with PHP backend and MySQL as a database for tracking connections and users. However, many other configurations are

*Figure 4.4: In the image on the left, it is possible to observe the 3D glasses that need to be worn by the user of the editor application and the two available controllers. On the right, the model of the Wand controller is shown on screen: by pressing the 'X' button, the user can generate a ray to facilitate the selection of objects.*

possible and, few paragrahs later, we will present another architecture for supporting more advanced features.

## 4.3 User interaction

In this section we will go deeper in the details of the user interface offered by our editor application, analyzing the available types of interaction with which the designer can customize virtual elements. We will describe first how we leveraged the technology offered by the CAVE2 environment and how this impacts on the editing experience, then we will describe the single features aimed at adding and modifying virtual content.

### 4.3.1 Interacting with the environment

After launching the editor application from a dedicated computer located outside the CAVE2 environment, all the 36 nodes of the cluster activate and start rendering the application on the 72 displays. The 3D perspective inside the environment is controlled by taking in consideration the position and movements of the person using the application: this is achieved by leveraging the camera system of CAVE2, which tracks the markers attached to a pair of passive 3D glasses that the user has to wear during all the editing session. This way, we know the position and rotation in space of the head of the user, enabling us to move the camera inside the Unity scene accordingly, while a plugin called *getReal3D* (produced by the company *Mechdyne* [54]) automatically takes care of correcting the perspective for all the displays. We will demonstrate later how the possibility of walking around a concrete,

immersive, hybrid reality environment has some significant differences with respect to perform the same actions in standard virtual reality.

Interaction with the editor application is mostly performed through wireless controllers. Aside from the 3D glasses shown in Figure 4.4, we can see the two devices we mainly used for our experiments: one is represented by an *Xbox One* [59] controller and the other one is *Playstation Move* [81] controller. Both of them are modified with the addition of markers that enable us to compute their position and orientation inside CAVE2 by leveraging the camera system; we mapped the buttons of both of them in order to perform all the actions required by the editor. The main difference is that, while the Xbox controller allows the use of a greater number of buttons and joypads, the Playstation Move controller can be used just with one hand. Since we considered using some features of the editor much more natural by using only one hand, in the following paragraphs we will decribe only how interaction is performed through he use of the latter, which we will shortly call "*Wand*" controller.

Inside the editor application, the controller is represented by a 3D model that moves according to the real movements of the device. In addition to this, pressing buttons on the controller causes the respective buttons on the 3D model to highlight as a simple form of visual feedback. We will start by the describing the most important function in the editor, which is strictly related to the pose of the *Wand* controller: selection. Since the tasks involve selecting and editing content in an inherently tridimensional space, we relied on the possibilieties offered by the CAVE2 environment in order to use the Playstation Move controller as a 3D input device. By pressing and holding the 'X' button, a infinite ray is cast from the front of the *Wand* in a direction parallel to the one of the device. Whenever that ray enters in contact with an object inside the scene, it stops in that point producing a simple particle effect that shows the user that he is pointing towards a specific object. For simplicity, from now on we will refer to this ray as "the laser" and we will call "pointer" its intersection with an object. Thanks to this, the user can always know which is the direction in space of the controller and can use it as reference when selecting content or interacting with the user interface. Our experiments show that without any sort of visual feedback, it is almost impossible to determine precisely where the controller is pointing at, issue that is completely solved thanks to the mechanism we just introduced.

The main idea behind the use of this type of controller is to make almost all actions directly available through its use, in order to have as much as possible a natural and at the same time fast way to perform tasks. From these considerations, we defined a minimal user interface system that can be

activated by pressing specific buttons on the controller. Since the shape of CAVE2 is circular, normal 2D interfaces would greatly suffer from perspective and parallax problems. For this reason, we created a resizeable curved user interface: placed in the 3D virtual space with a distance from the user that is equal to the one that separates him from the walls of the environment, the UI is deformed along the $Y$ axis in order to stick perfectly onto the displays even if the perspective changes. The main menu of the editor can be activated by pressing quickly the 'X' button and is composed of a simple window that allows the user to select through some dropdowns which type of content to insert inside the scene. Elements are filtered based on their type: the user can choose among the different types of virtual objects (2D images, videos, audio and 3D models) and fiducials (pattern images and beacon placeholders). By pointing the *Wand* at the dropdown of the UI window, the user can select between the available elements by pressing the 'X' button; then, a second dropdown is populated dynamically with a list of predefined content of the chosen type available for being inserted. When the user points at an element of the user interface, it becomes highlighted in order to show where he is pointing at, even if he is not pressing the 'X' for activating the laser feedback. This is achieved by explicitly defining colliders for each UI element and by exploiting the raycasting functionality of Unity, that allows us to identify which objects are hit when casting a ray in a direction. After having selected an element to insert into the scene, the user can simply move the pointer to an "Add" button and then press 'X' to instantiate that particular object in front of himself. In the current implementation, all the elements that can be added to the scene are predefined media assets, located in a particular folder of the master node of CAVE2, where the application loads them dinamically. By themselves, these assets do not contain any type of additional information more than their size (in pixels for images or relative units for 3D models) or eventually their animation properties. For instance, when the user selects the type "3D model", the application loads in the second dropdown the list of all available 3D objects, among which the user may choose "Horse"; after selecting the "Add" button on the UI, a horse appears in the scene, in front of the user. At this point it is possible to close the main menu by pressing the 'O' button on the *Wand*. This operation can be repeated multiple times and there is not a theorical limit to the number of elements that can be added to the scene.

Another fundamental operation performed through the controller is the camera movement. Differently from headset-based VR applications, the camera in the scene does not rotate with the head of the user. In fact, since the displays cover nearly 320 degrees of field of view, the user can simply

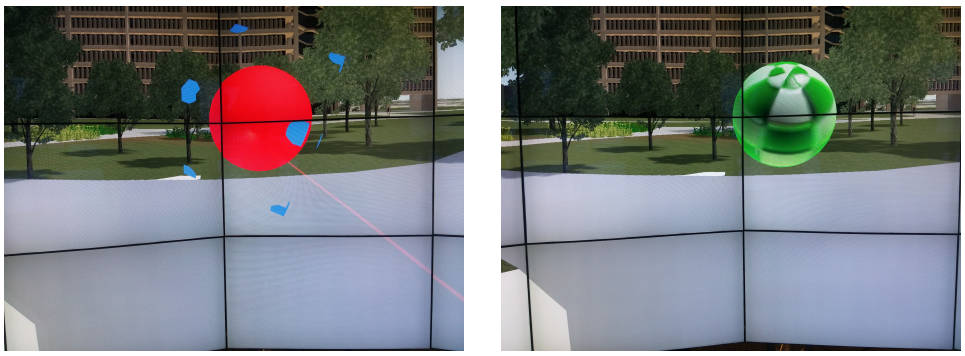

*Figure 4.5: In the image on the left, coloured corners appear around an object to show that the controller is hovering that particular content, that can now be selected; on the right, it is possible to observe how the default shader of an object is replaced in order to show which objects are currently selected.*

observe the rest of the scene from the lateral and back screens. However, there may be cases in which the user would prefer not no rotate his head and rotate the scene instead (we also need to consider that CAVE2 does not cover all 360 degrees); at the same time, the enviroment allows limited space where to move, so a way to move the camera around the scene is needed. The control of the camera, complimentary to the one performed by the perspective correction mentioned before, is defined through the joypad of the controller: moving it left or right rotates the camera along the vertical axis, while moving it forward or backward allows moving the camera along the (X,Z) horizontal plane, in the direction which the designer (wearing the tracked pair of 3D glasses) is looking at . Optionally, with a combination of buttons, it is also possible to move the camera up and down (in order to "fly" over the scene) and to increase or decrease the speed of movement.

### 4.3.2 Objects customization

After having inserted a virtual element inside the scene, the main task to be accomplished by the designer is to position it in the correct location in which it has to appear when a user will run his mobile application. In order to do this, an object first needs to be selected. Selection is performed by pressing the 'X' button on the controller when it is aiming at a particular object. On top of the feedback provided by the laser, four coloured corners appear around an object when the user hovers on it with the controller, in order to show the elements that he is about to select (Figure 4.5, left). Once the selection button is pressed, the material of that element is replaced by a shader meant to highlight the object and to make it distinguishable

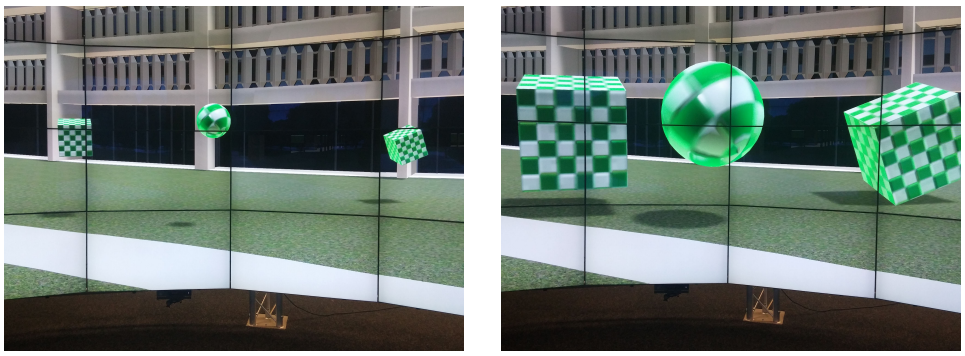

*Figure 4.6: In the above images it is shown how it is possible to select and edit multiple objects at the same time: after selecting the three elements (left), performing a scaling operation produces the result shown in the right image.*

from the other ones, as it is possible to see in Figure 4.5, right. At the same time, a window containing information about the object pops up on the curved UI: it displays the identifier and the type of the object, its size in meters (height, width, depth) and its location (latitude and longitude); two buttons on the UI allow to reset the object rotation and scale and to delete the object from the scene. In this first implementation, only one window is displayed on the UI at a time: for instance, if the main menu is open, the activation of the object details window will make it automatically disappear. To deselect an object, the user can simply press the 'O' button, that will restore the default shader of the selected element and will make the details window disappear. As it is possible to observe in Figure 4.6, our editor application allows selecting and editing multiple elements at the same time. By remembering that only virtual objects and fiducials are selectable (it is not currently possible to interact with the 3D reconstruction of the real-world environment), it is possible to define the following actions to be performed:

- Horizontal translation: by pressing the 4 arrows present on the *Wand* controller, it is possible to move the selected objects along the horizontal $(X, Z)$ plane. In particular, the forward and back arrows move the object along the direction that connects the center of CAVE2 to the object, whereas the left and right arrows allow movements in the perpendicular direction.

- Rotation: combined rotations around the global $X$ and $Y$ axes are enabled by using respectively the up / down and left / right arrow buttons.

- Scale: by pressing the up and down arrows, it is respectively possible to increase or decrease the local scale of objects, making them bigger or smaller while maintaining their original proportions.
- Vertical translation: in this mode, it is possible to increase or decrease the height of selected objects (global Y axis) by pressing the up and down arrows.

All the operations mentioned above are shown in Figure 4.7. When a selected object is modified, its information on the details window are updated real-time to reflect changes, so that the user can decide when to stop editing.

Since these actions are activated by the same buttons on the controller, we defined them as four separate editing modes of which only one can be activated at a time. When one or more elements are selected, it is possible to cycle through these editing mode by pressing the 'L1' button. In order to tell the user which editing mode is currently set, a respective icon is shown on the UI. During our tests, we also tried putting the selection of the editing mode on the UI, however it lead to significantly longer operation times with respect to pressing the 'L1' button. In relation to this, we must also consider that positioning the UI is a critical decision: placing windows too close to the currently edited content may occlude important areas of work, while putting it too distant will increase the time needed to reach it with the controller; at the same time, UI elements should be big enough in order to be easily selectable, since selecting small objects with the 3D controller requires the user to have a very high accuracy, increasing the effort needed to perform these simple tasks. For the above reasons, our general approach, when possible, prefers natural and quick ways of interaction, leaving the UI as minimal as possible at the cost of eventually decreasing the learnability of such actions.

While the actions defined above can be used for very precise modifications, in general we rely on a much more natural type of interaction, that leverages the particular shape and one-hand control of the controller we have chosen to use: when an element is selected, it is possible to press again and hold the 'X' button to grab the object and move it freely around the scene, like we had it in our hand. The current implementation casts a ray to the object and registers the point of contact, then makes it child of the *Wand* controller object, so that they move along in the same way by maintaing the distance defined when the 'X' button was pressed. This way, it is possible to move the selected object around the scene in the 3 dimensions with a single movement; at the same time, the user can rotate the object along the

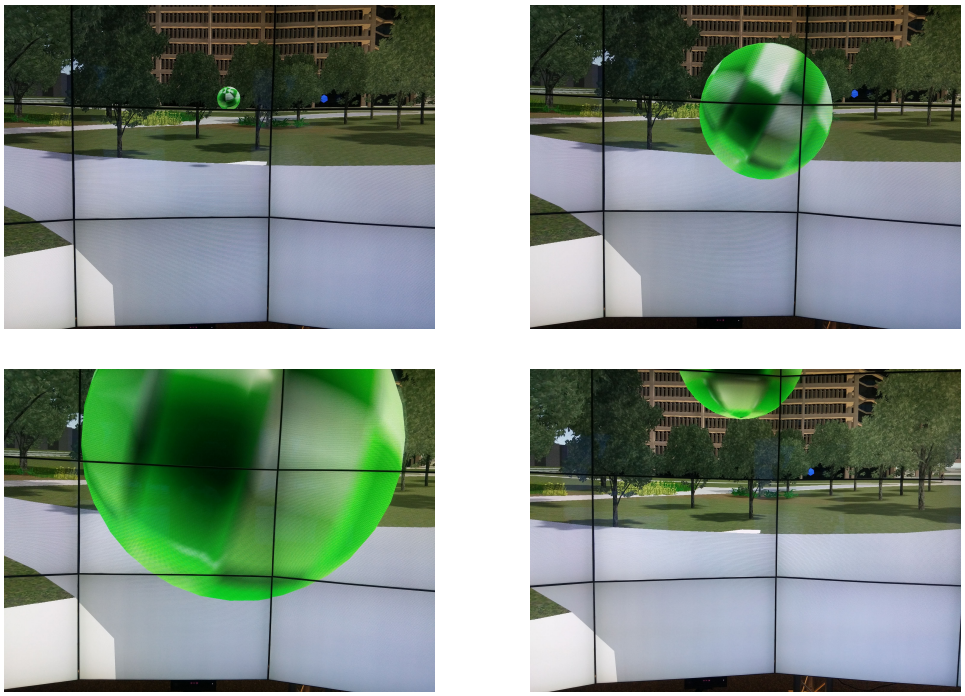

*Figure 4.7: The four basic editing operations are shown in the above images: in the first one, the object is translated along the forward direction; in the second one (rightmost, topmost) image, the object is rotated; in the third scaling is applied, increasing the size of the object; finally, the last one shows an increase in the height of the object.*

direction defined by the raycasting or in more complex ways by orientating the *Wand* in space. This possibility of moving almost with no constraints an object in 3D, allowing complex positioning with single-hand precision and in a highly natural and learnable way, proved to guarantee the best trade-off in terms of prototyping speed and accuracy. The main drawbacks of this method are two: moving an object feely with both translation and rotation enabled could cause unwanted modifications of the orientation of the object; the intrinsic way of moving the controller around the the CAVE2 environment makes it easy to move objects in the scene by maintaining their distance from the user, thus preferring circular translations. To address these two issues, we introduced the possibility to reset at anytime the default orientation of objects and the four editing modes mentioned before, that, despite maybe less intuitive, can be more effectively used for precisely moving elements in the desired way. Another limitation of the free movement of objects by dragging them around the scene is that, since space in CAVE2 is limited, it is not possible to directly walk with them in a specified position. However, by controlling the joypad of the controller with the other hand, it is possible to combine the movement of the dragged object with the movement of the camera, thus making it possible to move objects around the scene even at longer distances.

As we said before, these transformations can be applied both to virtual objects and to fiducials. Since in some cases their representation may be undistinguishable and since sometimes they could be not easily spotted inside the 3D reconstruction of the environment, we adopted two visualization techniques in order to allow the user to easily identify them:

- By pressing button 'L2', all virtual content is highlighted with an outline effect, whose color depends on the type of content (i.e. virtual objects are highlighted in white, fiducials in red - Figure 4.8, right).
- We defined a shader that allows virtual elements to be rendered even if behind other meshes present in the scene, often belonging to the reconstruction of the 3D environment. This way, the user can always notice the presence of an object even if hidden behind a wall (Figure 4.8, left).

As a reminder, we remember to the reader that the peculiarity of our editor application does not consist in simply editing objects in a virtual scene, but in modifying their real-world appearance when multiple users will run the mobile application. So every change in the editor is reflected by a change in how users will see the content with respect to the live video stream from their mobile device.

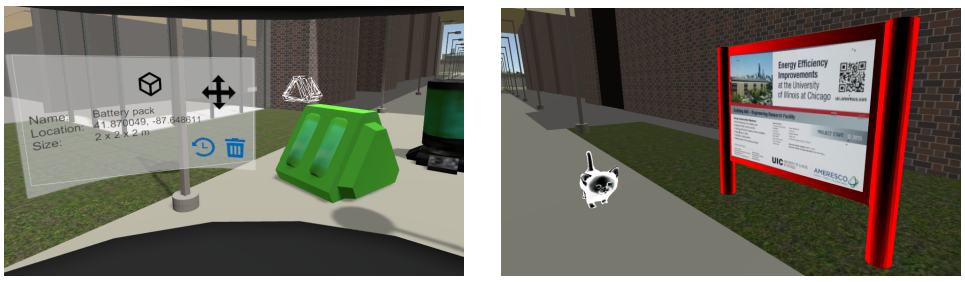


*Figure 4.8: On the left, a virtual object is selected and highlighted in green; a panel with the object details allows some additional editing features, while in the background a wireframe shader makes an object behind a wall visible to the designer. The figure on the right shows instead how content is categorized by pressing the 'L2' button: virtual content (not belonging to the environment) is temporarily highlighted in white, while fiducials are visualized in red.*

## 4.4 Real-time editing

Until now, we have dealt with an off-line approach to editing the elements characterizing the MR experience: the designer uses the authoring tool to insert and preview the desired virtual content, a configuration file is created (according to the protocol defined in *Section 4.2.2*) and mobile clients are able to visualize the content by loading this specific file. As normally happens with this type of applications, after the MR scene has been loaded on the mobile device the designer doesn't have anymore control over the single instances of the application, which operate independently. In some cases it may be useful instead to allow the designer to edit the MR experience at run-time and to keep track of the running mobile instances: this would allow the development of dynamic MR applications able to change over time and to adapt to concrete user behaviors. For example, we could improve at run-time the position of an object that turns out to be hardly reachable due to environmental contraints, limiting the accuracy of mobile devices in a particular area; otherwise we could simply change the disposition of elements when organizing collaborative MR live events, so that users will see different content based on external variables such as time, weather and behavior of the other users. On the other hand, a real-time approach would also open to new interaction possibilities from the client's perspective, which include allowing mobile users to modify the virtual content directly from their device. In chapter *Case studies* we will show some other concrete examples of how real-time editing can be used to improve an MR experience. However, the system architecture presented earlier does not allow a smooth transition to a real-time approach. For instance, we could update through the editor

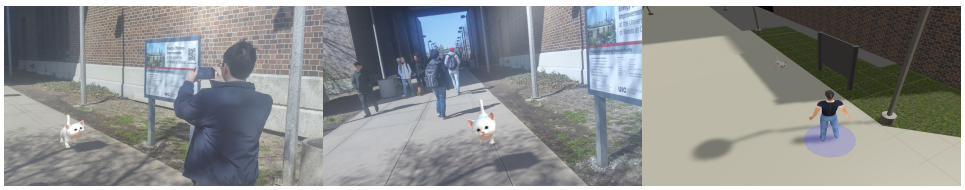

*Figure 4.9: In the above figure, the left image shows a user with one virtual content that has been added to the the real world, while the middle one represents how that user is able to see the AR overlay through his mobile phone; in the the image on the right it is instead possible to observe how the user and the virtual elements are represented at real-time in our authoring tool, together with a partial reconstruction of the environment.*

the configuration stored on the central server and then make each mobile application do a request every few seconds in order to check if something has been modified in the meantime. This method, with a larger number of users, would create a massive number of requests to the server with a big useless overhead; in addition to this, it would not be able to render movements smoothly ad in some cases, if modifications happen very quickly, it wouldn't even be able to show transitions at all. Some small improvements could be done by timestamping modifications to the single elements and updating clients only when a modification is performed, but a dedicated networking architecture is required in order to guarantee solid performances. We note that, while we previously defined a way for mobile devices to operate off-line, by enabling real-time features mobile connectivity becomes required on all devices.

### 4.4.1 System architecture

The necessity of having a networking system able to operate real-time modifications to the MR experience involves configuring a lower level infrastructure for serializing data, sending and receiving network messages, commands and events - all by maintaining an acceptable delay between the operations on the virtual content and their modified rendering on all other devices. To implement these features, we relied on the HLAPI [93] service offered by Unity, which greatly simplifies communication between devices, taking care of all basic networking services.

In the new system architecture we propose, we do not use anymore an external server for storing the MR configuration, but we leverage directly the master node of CAVE2 as both editor and server. All client mobile applications, at startup, will have to connect to the master node through a

dedicated IP address and port. While mobile devices, depending on the type of connectivity (e.g. Wi-Fi or cellular) and eventually the phone carrier, may result connected to private networks, CAVE2 has its own public IP address and thus does not require NAT traversal or proxy servers for establishing and maintaining internet connectivity: this simplifies noticeably incoming connections and makes CAVE2 a good candidate for acting as a server.

One first main difference with the previous off-line approach, is that the information about the MR scene is not stored anymore in a configuration file that clients need to download and interpret. Now every virtual content defined in the editor and meant to appear on the mobile application is characterized by a network identity. When a client connects to the server, this instantiates automatically on the mobile device all elements having a network identity, placing them with the correct position and orientation in space. The virtual scene built inside the editor acts indeed as a reference for all other instances of the same MR experience. By leveraging Unity's *network transform* component and the use of *syncronized variables*, when a content is modified inside the editor all changes are propagated to the clients, which will be able to see them with very low latency (¡ 40 ms in all our tests, with both WiFi and mobile connectivity).

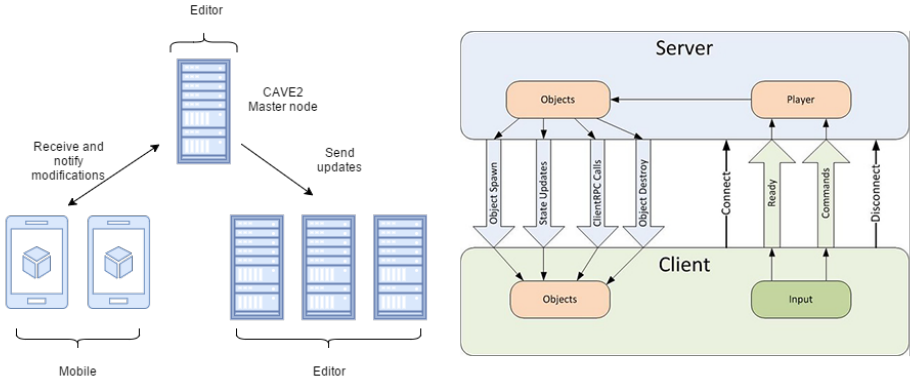


*Figure 4.10: Second and final version of our system architecture: the master node of CAVE2 now acts as a server and maintains the networked state of the MR application; both mobile devices and the other CAVE2 nodes act as clients of the master node. In the Unity networking diagram on the right [94], it is possible to observe how commands are used by the clients to update the state of the server, while techniques such as spawning, RPC calls and synchronized variables are used in order to keep consistency on the clients.*

We want to note that the CAVE2 cluster is composed of 37 computers, of which exclusively one (the master node) acts as a server: otherwise every obile device would have to keep 36 open connections to all the nodes of the

cluster, which is inconceivable. This means that the nodes of the cluster do not run anymore the same instance of the editor application, since the slave nodes will not apply the server logic. From this point of view, these 36 nodes now act themselves as "clients" of the master node, which in some cases will need to update them manually through RPC calls in order to maintain consistency. Let's consider two cases: when some edits are operated from the editor and when eventual modifications are allowed to be performed by mobile clients. In the former case, the insertion, modification or deletion of content inside the editor scene is performed identically on all nodes, since the input of the wand controller is connected to all of them; however, only the master node will take care of sending updates to the mobile clients. In the latter case, a modification on a client will notify the server running on the master node, that will update the editor scene correspondingly and then forward these changes to the other instances of the mobile application; at this point, the state of the networked scene will not be consistent inside CAVE2, since only the master node will have the updated version of the content and thus the modifications will not be rendered on the 72 displays controlled by the remaining nodes of the cluster. To solve this issue, the master node needs to manually update the properties of networked items on all the other nodes: this can simply achieved by keeping a data structure with the list of virtual content and update it on all the cluster to RPC calls. In this sense, if modifications to the content are performed from external applications, the master node will now have to maintain consistency both on mobile clients and on local nodes of the cluster.

## 4.5 A portal between the two worlds

Real-time editing of a MR experience can be very useful to adjust content and to correct design choices, but leads also to many more interesting applications. Until now, in our editor we have defined three types of visible content: virtual objects, fiducials and the non-interactive 3D reconstruction of the environment, aimed at guiding the designer during content definition and modification. Trying to consider mixed reality from a different perspective, a very important type of content is still missing: mobile users. By enabling real-time capabilities of the editor application, we also opened the possibility to visualize and potentially interact with users while the run our mobile MR experience. This would allow many advantages for the designer, among which:

- Knowing the position of users at run-time, in order to study their

behavior and modify future design decisions accordingly.

- Correcting the displacement of content if users appear not to be able to reach it due to environmental contraints.
- Preview the user mixed reality perspective directly from the editor, without the need of observing phisically each single person during the use of the mobile application.
- Testing the MR experience before public deployment, with the possibility to collect a larger amount of user data and to debug selectively the entire MR solution.
- Experimenting new ways of interaction between the mobile users and the people staying in the CAVE2 environment, creating a sort of "portal" interconnecting the designers and the people in the outer world.

### 4.5.1 Representing users

We wil start our discussion by first considering how users can be represented inside the editor application. Thanks to the information gathered through our mobile aplication, we are able to know the geolocation (latitude and longitude), estimated horizontal accuracy and mobile device orientation in space of a user. On top of this, we are also able to know the field of view of the mobile camera and other statistical information about the device itself. Unfortunately, we are not able to determine other parameters such as the height or the shape of his body nor the position of the joints. Overall, by considering the horizontal position on the (X,Z) plane and the 3D orientation of the device, we have a total of 5 degrees of freedom. Since this is actually the information we need for our purposes and for now we are not interested in a realistic representation of users, a simply yet efficient way to render them in the editor application is to use avatars.

In our editor an avatar is instantiated each time a user connects automatically to the server when starting the mobile application. The avatar is set to have human-like proportions and default height of 170 centimenters. Its position in space is based on the geolocation estimated by the mobile application and is computed by using the same map projections explained in Chapter 3, section *Mapping the two worlds*. Since smoothing and filtering are already performed on the mobile application, in the editor we can simply use the already pre-computed values. Not modifying original values has also the objective of seeing exactly the same behavior of the mobile application, in order to notice eventual incoherences and eventually take the needed

countermeasures. Currently, the avatar is characterized by two animations: standing and walking, activated depending on the variation of position over time. It is very important to take into account the horizontal accuracy of the position of the avatar: is it probable that the user will not be in the same exact position indicated by the avatar, since, as we showed in *Chapter 3*, many factors may influence the camera pose estimation. To address this problem, we defined for each avatar an area which indicates what is the "probable" location of the user: a red circle between the ground and the feet of the avatar has a diameter corresponding to the estimated horizontal accuracy of the mobile device. This way, we can expect the real position of the user to be anywhere inside the highlighted area, in order to have a realistic and quantitative image of the accuracy of the overall MR experience for that specific user. Another very important aspect of the avatar representation resides in mapping the device orientation in space to the rotation of the head of the avatar. In the real world case, the device of the user can be both moved and rotated in space, passing from situations in which it stays in the pocket of the user to others in which it is kept in front of the user's eyes (when concretely using the application) or simply kept in hand along the side of his body (when the application is inactive or the person is resting). Since we are mostly interested in the moments in which the device is held for a continuous use in front of the user's eyes, with some approximation we decided to map its orientation to the head of the avatar, rotating it correspondingly. The hardware mobile camera is indeed represented in our application with a virtual camera placed orthogonally to the eyes of the avatar, with a field of view that matches the real one. While the rotation of the head is defined by the pose estimated by the mobile camera, a possible question involves the orientation along the vertical axis of the avatar itself: should it rotate togheter with the head, by keeping only the rotation on the Y axis, or rotate independently? In real life, the user may rotate the device up to a certain angle before actually rotating his body and moving his legs. Since our main interest is the rotation of the head of the avatar, this may appear as a secundary issue, especially considering we do not have a precise way to estimate the rotation of the body with respect to the head. However, in order to make the movements of the avatar realistic, we applied the unmodified estimated pose of the camera to the head, and rotated the body by applying a smoothing factor to the head rotation along the vertical axis: this way, the avatar will first move its head with real-time rotations and, after a certain amount of time, slowly rotate its body to match the orientation of the head.

As a final result, in the editor we will have one avatar for each person

currently using our mobile application, where the head of the avatar is placed and oriented according to the estimated camera pose of the mobile device. For instance, if a user moves the device up in order to see a far building, we will see the head of its corresponding avatar rotate up at real-time with the same angle; if a user spots an object to his left and tries to get closer to it, we will be able to observe its avatar turning left and walking towards the virtual content. By considering the accuracy area highlighted below the avatar, we are able to take into account which may be the possible real position of the user and at the same time know exactly the position according to which the content the user sees is rendered. For instance, even if we don't know very precisely the real location of a user, by seeing in the editor its estimated position we can eventually move content towards the user if we realize he is not able to reach it: in fact, the content is rendered by the mobile application based on the estimated pose of the camera, not the real one.

For what it concerns our networking implementation, the master node of CAVE2 takes care of handling the incoming connections from mobile users and creates an avatar for each of them, keeping a centralized list of connected users. Instead of using Unity's default *NetworkManager*, we preferred to implement ourselves a customized network solution on top of the HLAPI offered by the engine: this way, we are able to track with more precision each networked event, spawn avatars on client nodes and update their state (otherwise they would be shown only on the master node but not in CAVE2) and handle specific actions that go beyond the functionalities offered by Unity's ready-made solution.

### 4.5.2 Visualizing user data

Inside the editor application it is possible to select avatars with the *Wand* controller in the same way this operation is performed with normal virtual elements; however, since the size of avatars is fixed and their position is controlled by the devices running our mobile application, customization features are disabled for avatars. In place of those functionalities, it is possible to performs more actions that are specific for visualizing users' data and interacting with them.

When selecting an avatar, its shader is replaced in order to highlight it like with virtual content, but a wireframe camera frustum is also represented in front of the eyes of the avatar, allowing a more precise visualization of the orientation of the related user's device. The details window that appears on the graphical user interface shows the the geolocation of the user with latitude and longitude coordinates and some useful information information

involving the user's device:

- Its model, operative system and screen resolution.
- The field of view and resolution of the mobile camera.
- The estimated horizontal accuracy of the device.
- The framerate of the rendering and tracking threads running on the device processor.
- The types of tracking currently used for determining the position and orientation of the device.

By taking into account this information, the designer can collect statistical data and make considerations about the technology currently used by the people running the mobile application, eventually taking some runtime decisions on the displacement of content. For example, the designer can easily detect if users are trying to run the application with a non fully compatible device or if one of the tracking methods is not working properly.

### 4.5.3 Assuming the user's perspective

By knowing the absolute position and orientation of the user's device in the real world and by leveraging our recontruction of the environment, we are able to preview what a user is seeing according to the tracking performed by his smartphone. The idea is that we simply put a virtual camera in correspondence of the eyes of an avatar and we make its parameters (i.e. the field of view) correspond to the ones of the real camera. This way, without any difficult computation, we easily obtain a dynamic 2D texture containing what we expect the user to see on his mobile display according to the best of our knowledge. In particular, this texture will contain only the elements visible inside the editor application, which means that only the augmented content and the reconstruction of the environment will be shown, not the actual video stream from the camera. This feature allows us to know real-time what the user is looking at, but is dependent on the tracking performed on the mobile device and is accurate in relation to the quality of our environment reconstruction. For instance, an unstable horizontal accuracy of the GPS signal could cause our preview to be completely inaccurate, whereas environmental modifications with respect to the model in the editor could never be represented (e.g. weather conditions, a new building, the passing of vehicles, the presence of unexpected obstacles in the real world). For this reason, in order to provide a visual comparison between what the player

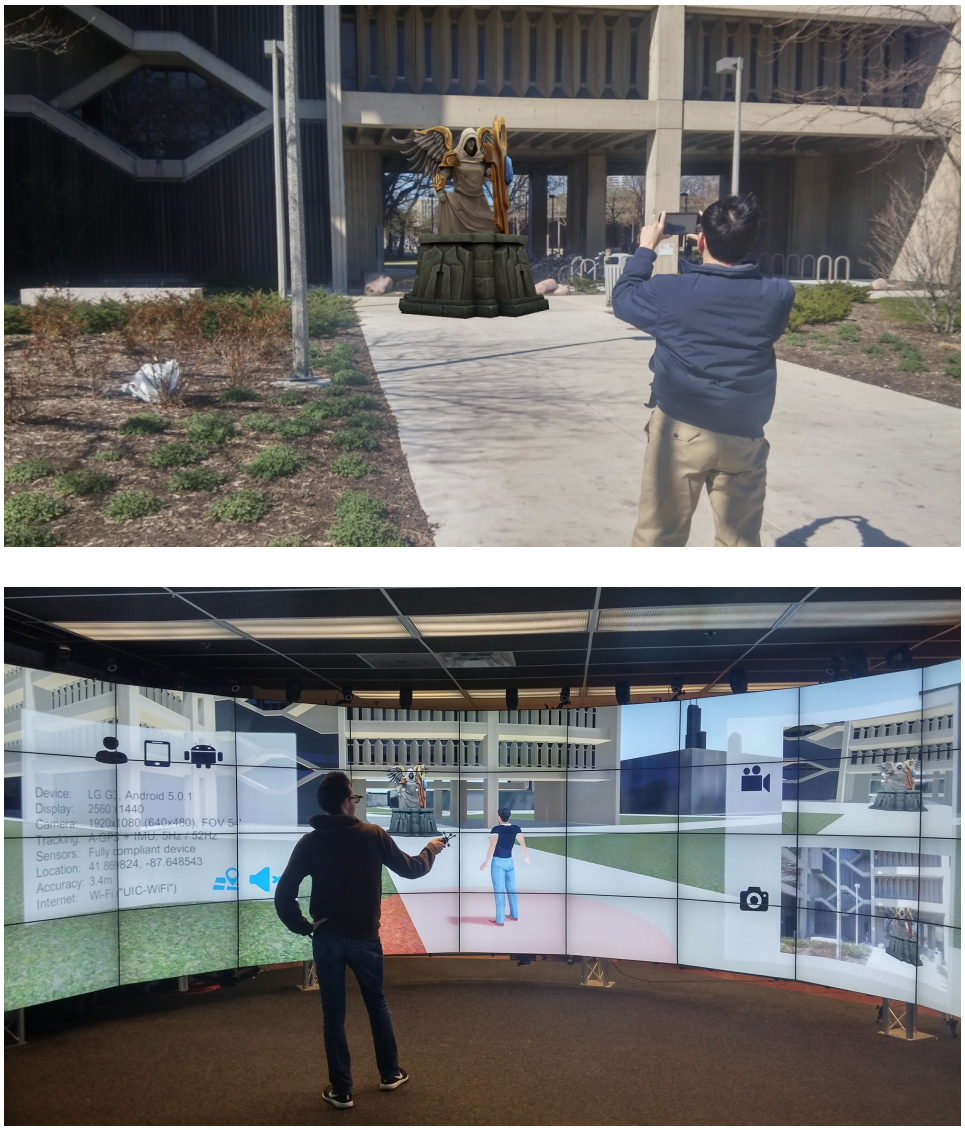


*Figure 4.11: In the upper image, the user in the real world is seeing the augmented content through his mobile device; in the lower image, the designer is able to observe in real-time from the editor application what that specific user is seeing. In particular, the red area below the avatar represents the estimated horizontal accuracy of the user, while in the panel on the left some information about his mobile device are displayed. The right panel shows instead a comparison between two views: the upper one is the expected perspective of the user, while the lower one is represented by a video streaming of what he is actually seeing.*

sees and what we expect him or her to see, we provide also a simple form of image streaming between the mobile devices and our editor application.

When an avatar is selected, a window shows in two separate views both what the user is seeing according to the position of his avatar in the editor application and, on request, the live video stream from his or her mobile device, including the augmented content from his or her perspective. Since, due to accuracy errors, content is rendered on the device according to the estimated position and orientation of a user, comparing these two views allows the designer to immediately understand if the user is perceiving the virtual content in the correct way and to eventually adjust the location of objects inside the scene. Having the possibility to activate the live video feed from the mobile camera makes it possible to identify external elements that are not modeled inside the virtual environment, such as weather conditions, passing of people or cars, environment modifications— elements that may significantly affect the AR experience. In Figure 4.11 (right), it is possible to observe how the two views are located inside the GUI: the upper one renders how the user is expected two see a statue considering the estimated position of his device, while the lower one reflects how the user concretely sees the augmented content. Video streaming has currently been tested only on a LG G3 device with a framerate of 10 FPS, wich we consider a good compromise in order not to use too many additional resources on the mobile device - especially regarding the data network; video streaming is in fact disabled by default and should be activated only when required.

In order to enable more accurate debugging of the overall AR experience, once an avatar is selected, it is also possible to activate what we called "user perspective" mode: the camera rendering the scene inside the editor application is placed in the position of the avatar and follows its movements, in order to simulate a first person view of the mobile application from the user's perspective. Since the CAVE2 is characterized by a cylindrical shape, it is not so straightforward to represent the limited field of view of a mobile camera. Aiming at addressing this problem, we propose the solution of creating a "mask" whose size is proportionally related to the field of view of the device (Figure 4.12). All the content outside this mask is slightly darkened in order highlight only what the player is currently seeing.

Another decision involves how to render the mask when a user is rotating his or her device. A first option is keep the mask fixed in the central screens of CAVE2 and rotate the surrounding scene according to the orientation in space of the device. An alternative is to leave the scene as it is and move the mask on different screens. The former solution has the advantage of leaving the point of interest fixed in front of the designer, but can possibly

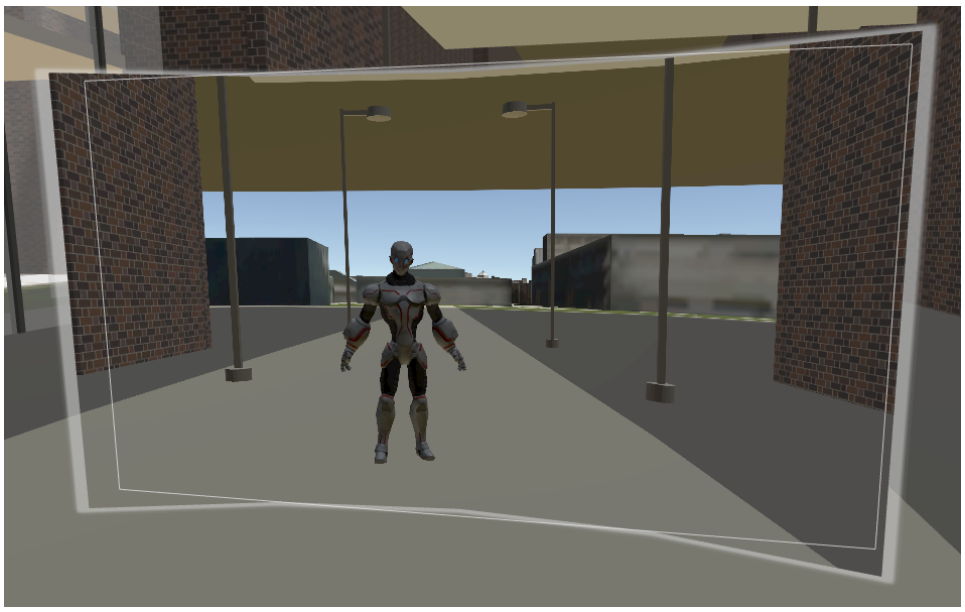

*Figure 4.12: By defining a mask proportional to the field of view of the mobile device camera of a user, it is possible to simulate inside CAVE2 a first-person perspective of what he sees.*

create unpleasant rotations of the content rendered on all the other screens, especially in case of fast orientation changes; the latter requires instead the designer to follow the movements of the mask with his head, but has the advantage of not modifying the content rendered on the screens and could potentially by extended to the selection of multiple users. Overall, for our implementation in CAVE2, we preferred to keep the former option as we consider its "first-person" perspective a more natural way to impersonate a user. A last possible solution is based on implementing the editor as an HMD application, leveraging the inherently first-person view of this technology: we will briefly explain this alternative implementation at the end of this chapter.

### 4.5.4 Physical participation of the designer

In the previous subsection we explored many possibilities for putting the designer in the perspective of the users of the mobile application and we focused on how to visualize inside our editor application elements belonging to the outer world. An interesting possibility is to consider also the inverse relationship: what if elements inside CAVE2 could be rendered in the real world? In particular, the camera tracking system and the Kinect v2 sensor

of CAVE2 allow us to track the movements of the people inside the environement, allowing to recreate virtually their movements through motion capture. With an accuracy slightly dependent on the implementation technology (camera system vs Kinect), we are able to digitalize the designer and the eventual other people as animated avatars, that can be inserted into the mixed reality scene as virtual content. For instance, by using the Kinect sensor, we can track up to 6 people inside CAVE2, determining at real-time the position and rotation in space of a maximum of 29 body joints for each person. While the camera system of CAVE2 allows a much more precise tracking, it requires people inside the environment to wear markers on the joints of their body, which is often less practical unless there are specific requirements of accuracy. By allowing the designer to place these avatars at a specific location in the editor, these people appear at the corresponding real-world position on the screen of mobile users. For instance, the designer can insert himself next to a user and give him physical indications on how to reach a nearby virtual object; another example, involving *Chicago 0,0*, could be related to enabling remote touristic guidance to users in the city, without needing a guide to go phisically onsite for this task. In brief, it is possible to project a person into whichever desired part of the world and make him appear as a contextualized avatar onto the mobile MR application of the users in that location. While the position of the avatar is determined by the position of the camera rendering the editor and the pose of the avatar joints is given by the tracking performed by the Kinect, the audio is selectively streamed to nearby users from the Kinect's built-in microphone. The current state of this feature is still under development and will not be evaluated in the case studies presented in *Chapter 5*, however our current implementation already demonstrated its feasibility by projecting four people inside CAVE2 on the screen of a mobile device located outdoors in front of the EVL laboratory. In this case we preferred the use of the Kinect due to the above mentioned reasons: an average of 20 + 20 joint rotations and positions per tracked person is collected at 30 FPS by the master node of CAVE2, allowing the computation of the required inverse kinematics and a smooth representation of the avatars within the editor; the refresh rate with which avatar movements are propagated to the client application is currently 15 FPS.

## 4.6 Alternative implementations

Since not everybody can afford to develop an editor application in a CAVE2 environment, we also tried to explore more scalable implementations of our

authoring tool with slightly different features. We will now list some of them, discussing the pros and cons of which implementation with respect to the previously presented one.

### 4.6.1 Head Mounted Displays

Basically, this implementation uses the same virtual scene that we presented for CAVE2, but relies on slightly different user interfaces. The designer, instead of walking inside a hybrid environment, wears an HMD display (in our case, we used an HTC Vive [40]) and is completely immersed inside the 3D representation of the real world, to which virtual objects are added. Both the features available and the user interface are the same, since, similarly to the shape of CAVE2, the head rotation is performed along a circular path. One first main difference is that in this case the user of the editor application needs to stand or seat in a fixed position for eventually prolonged time, thus sometimes leading to fatigue and eye strain due to the excessive use of the HMD display. For what it concerns interfacing with the application, HTC Vive provides two controllers that are very similar to the *Wand* we used in CAVE2: with a simple mapping of the available features, it was very easy to port the activation of many actions from one type of controller to the other one; despite the new touchpad can naturally substitute the joypad of the Wand, HTC Vive's input devices present less buttons with respect to the Playstation Move controller, so we preferred to deal with this limitation by enabling the use of both the controllers available with the VIVE (i.e. one controller is mostly related to moving the camera, the other one is used for customizing content). Instead, a great advantage of this implementation is that, differently from CAVE2, it allows to have a full spherical environment, thus including also the floor and the sky; on top of this, when enabling the perspective of a mobile application user, the HMD is more suited to simulate his first-person point of view.

### 4.6.2 Unity Editor extension

This is probably the simplest and cheapest implementation, that we used in the early phases of the development of the two applications *Chicago 0,0* and *DigitalQuest*. It leverages the use of Google Maps [34] and can be performed directly inside Unity Editor, without requiring a complete 3D reconstruction of the environment 5.2. In this implementation we defined two different views: a *User mode*, a 1:1 scaled simulation where we can preview what the user would be able to see from a particular perspective, and a *Map mode*, a 1:100 scaled map representation with a 3D perspective top-view of the

previous mode. A more detailed description of this environment will be provided in *Section 5.2.2* in relation to the development of *Chicago 0,0*.

### 4.6.3 Mobile editor

This prototypal implementation is complimentary to the Unity Editor one and provides the possibility to adjust the positioning of content by going onsite and testing overlays directly on top of a live camera feed. In brief, it consists in a modified version of the client application we normally used for viewing augmented content: the idea is that, after defining the position of virtual content through one of the other editor implementations, the designer can go onsite to check if content appears in the correct locations. With the help of an auxiliary map and some additional controls, the designer can modify with simple gestures the position, orientation and scale of each virtual object, trying to align them with the current views of the real world. The main difference with the other implementations is in fact that the Mobile editor is conceived as a helper tool, aimed at "correcting" onsite eventually imprecise content definitions previously made with the remote editor; this feature is mostly achieved thanks to the video see-through capability of smarthphone devices, allowing us to match real video footage and synthetic virtual elements. When a modification is performed, the Mobile editor simply updates the configuration file related to the MR experience. However, as we explained in *Section 2.7*, "editing MR from within MR" may suffer from imprecise tracking, so that testing the application in a different situation or with a different device may not guarantee the same results.

### 4.6.4 Web-based editor

An alternative idea aimed at creating a very scalable tool is to implement it as a web application, so that it could be used remotely and be accessible by everyone through a simple web browser. The key concept of this prototypal editor is to leverage the huge imagery dataset provided by *Google Street View* [36] in order to build a tool that allows the placement of virtual content in 3D inside widely available spherical panoramas, combined with a partial reconstruction of the environment based on depth maps.

The implementation is based on the WebGL [64] wrapper THREE.js [89], used in combination with Google's imagery to avoid the necessity to obtain 3D models of the environment. Currently, the designer is prompted to input an address or to select a location from Google Maps, and then the editor loads the closest Google Street View panoramic image, if available. By leveraging WebGL and Google APIs, the imagery is projected onto a sphere

*Figure 4.13: Implementing a web-based editor by leveraging Google Street View imagery. In the figure above it is possible to observe a sample panoramic image (top) and its relative computed depth map (middle). The depth information is used to allow a more precise positioning of the virtual objects in the real world (bottom), which can simply be dragged into the scene and then moved, scaled or rotated.*

in order to obtain a spherical panorama viewer. The user can then drag and drop the virtual object into the scene in order to preview how the augmentation will look like when running on the mobile application. In order to place objects inside the spherical projection, we need to retrieve the depth map corresponding to the current panorama in order to determine distances and permitted positions. The Google Maps REST API provides functionality to download a compressed JSON representation of the depth image, which contains the distance from the camera to the nearest surface at each pixel of the panorama. We obtain a grid of pixels after decoding the image from Base64 and converting it to a more suitable data structure. Each of the pixels in this grid corresponds to one of several planes, defined by its normal vector and its distance to the camera. Therefore, in order to calculate the depth at each pixel we have to determine the intersection point of a ray starting at the center of the camera and the plane corresponding to the pixel. Iterating for all the planes, we can then populate our depth map which will have a representation similar to the one in Figure 4.13. By considering the correspondence between the points in the 2D texture and the points projected on the sphere, we can calculate the 3D position of the objects (e.g. the walls of the buildings) that are present in the panoramic image. We can then restrict the positions that are available for the virtual object to placed at. By mapping the 3D scene distances with the real world ones in a similar way to the one we presented for our CAVE2 editor, we can obtain the absolute coordinates, orientation, and dimensions of the object as if it were a real object. This information can then be stored on the server and loaded by users on the mobile application.

## 4.7 Final considerations

In this section we have showed how our authoring tool can be used to create and customize an AR experience in an easy and precise way and we have then introduced how adding real-time features to the customization of content allows us many additional interaction possibilities, especially with respect to the users. In particular, we believe that many of the advantages involving real-time applications lie around the possibilities of interaction of users among themselves and with the people eventually coordinating the AR experience from CAVE2. For instance we can identify the following use cases:

- It is very common for virtual content not to be rendered as the designer originally expected due to many reasons, mostly related to the low

accuracy of mobile sensors and to changes in the environment. In this case, users can notify this problem and at the same time the designer can verify the possible motivations by analyzing what the user is seeing and the statistics gathered from his device. This way, the designer can operate adjustments to the location of virtual content taking in consideration the current environmental situation and the state of the mobile device; alternatively, by considering the live stream video, he can provide guidance and assistance to the user in completing a task or visualizing content correctly.

- A real-time implementation allows even users themselves to customize the environment, by adding and moving their own content and make this changes reflect on the instance of the application run by other users. Some examples could involve picking up virtual items at a certain location or replacing them, trading objects, activating events for collaborative tasks or implementing videogame-based features like shooting virtual bullets.

- Interesting possible applications involve the monitoring of users' behaviour and the centralized control of activities. For example, we may consider directing from CAVE2 independent or collaborative tasks, that are assigned to multiple teams of people. The possibility to control all users at the same time from a single location makes it possible to direct and study how they "perform specific missions".

- In *Chapter 5* we will briefly discuss the possibility of of providing remotely onsite assistance by rendering the body of the designer directly on the users' mobile device as augmented content; similar applications could include the introduction of intelligent "helping avatars", controlled by AI and aimed at helping people during their tasks.

- Ultimately, a real-time implementation allows the presence of dynamic content and animations: virtual content can be moved under certain narrative circumstances or to force users to follow its movements; elements such as NPCs, animals and virtual creatures can be added and act with their own logic, moving closer to users or interacting with them, while keeping consistency on all devices at the same time.

# Chapter 5

# Case Studies

This chapter is aimed at presenting to the reader some real-world examples to which our approach can be applied, in order to show more clearly its advantages with respect to traditional methods. In addition to this, we will show how our editor application can be used to greatly improve the design and creation of MR experiences and how our protocol can be extended in order to suit different needs. We will ultimately provide quantitative data and discuss the feasability of building accurate MR applications with the currently available technologies. In particular, we will present two different mobile applications developed with our approach, *Chicago 0,0* and *DigitalQuest*. Despite based on the same method, these two applications have very different characteristics:

- *Chicago 0,0* is aimed at presenting historical photographs of the city of Chicago in an urban environment, characterized by low sensors accuracy and thus requiring the use of fiducials. The virtual content that need to be shown is composed of bidimensional transparencies, that need to be located in 3D space in order to make them overlap with particular views of the city.
- *DigitalQuest* is a collaborative videogame intended for organizing events, providing a digital version of the classical scavenger hunts. The virtual content associated to this application involves mostly 3D content, which does not need to have an extremely precise registration in space but requires the definition of custom behaviors associated to it, according to different in-game variables that need to be taken into account. Differently from *Chicago 0,0*, *DigitalQuest* is mostly intended for outdoor open spaces where mobile sensors are more accurate.

For each of these two applications, we will provide a brief background,

presenting eventual previous works and stating what are the specific advantages of applying our approach to that context. In particular, we will highlight the significant implementation details that describe how our approach has been adapted to the different situations. We will also provide some data gathered during sample executions of both applications, in order to allow space for a short discussion on the advantages and limitations of our method. Specifically, *Sections 5.2* and *5.3* will be focused on our standard dual-camera approach, while *Section 5.4* will explore the additional advantages achievable through the use of the SLAMCamera. Before starting with the presentation of *Chicago 0,0*, in the next section we will quickly describe how we approached the measurement and analisys of the quantitative data we used for our discussion.

## 5.1 Measurements and feasability

In order to being able to produce useful data to be used in our evalutations, we slightly modified the normal implementation of our two mobile applications, allowing the debug of raw values gathered from mobile sensors and at the same time storing the values computed by our algorithm. By adding the possibility to see on an optional map the position and orientation in space of helper cameras and virtual content, we are also able to see directly, during the execution of the application, how well our approach is performing; with a simple in-app tool we created, we can ultimately compute real-world distances between elements inside the scene, specifying manually positions and rotations on the map in order to make the difference between actual and estimated values explicit. Most of our measurements are expressed in meters, with a resolution of 0.1m, or in degrees, with a resolution of 1°; despite we could potentially use more precise values, we believe resolutions lower to the ones we provide would be meaningless with respect to the accuracy of our measurement tools.

For what concerns the mobile application, we gathered the following main types of data, which we considered relevant in order to discuss the performance of our approach:

- Estimated GPS error ($e_{gps_{est}}$), calculated as half of the horizontal accuracy (in meters) estimated natively by the device, based on the combination of GPS, cellular and Wi-Fi signals. In our experiments, the device was set in order to request the best accuracy achievable through these technologies, given the environmental limitations.

- Real GPS error ($e_{gps_{real}}$), computed as the distance in meters between

the GPS position estimated by the device and the real position of the device, manually specified through an interactive map we introduced in our application for debug purposes.

- Location error ($e_{loc}$), representing the distance in meters between the position computed with our apporach and the real position of the device.
- Compass orientation error ($e_{comp}$), providing the angular difference on the vertical axis between the orientation provided by the compass sensor and the vector connecting the device and the position of the current virtual object. Optimally, it should tend to zero when the device is perfectly aligned towards the center of the virtual content.
- Orientation error ($e_{or}$), very similar to the compass orientation error, but referred to the direction computed through our approach and not by the compass.
- Fiducial, stating if pattern-based tracking was performed in order to compute the values of $e_{loc}$ and $e_{or}$ and, in case, with which type of fiducial.
- SLAM, stating if the SLAMCamera was added to our standard dual-camera approach, affecting the estimation of the values $e_{loc}$ and $e_{or}$.

During the estimation of the position and orientation of the device camera, our goal is to minimize the two errors $e_{loc}$ and $e_{or}$, based on the values computed by our approach. Since we cannot improve the available hardware technology, it is impossible to bring the estimation error to zero, but we can still use the raw values computed by the single mobile sensors as a rough comparison. On top of this, it is fundamental to consider that our approach does not aim at improving already existing localization or tracking technologies, but tries to combine them in novel ways to enable a wider range of possibilities both for the user and the designer. So, the experiments proposed in this chapter will represent more a feasibility study than an attempt to enhance other techniques, and will be accompanied by substantial discussion parts.

A different method was used to evaluate the usefulness of our editor application. In this case, taking into account accuracy measurements related to a specific mobile device would be meaningless, so we decided to apply more abstract criterias to our authoring tool. In particular, we focused on the task of inserting and placing virtual content through the following definitions:

- *Insertion time*, referring to the time (in seconds) that was required to the user to insert and position a virtual object in a desired position.

- *Acceptable overlay* is represented by a binary value ('yes' or 'no') indicating if, excluding accuracy errors related to the mobile device, the virtual content concretely appeared in the specified real-world location.

- *Should be improved* is a binary value related to the optimal position in space that could be chosen for placing virtual content, considering the acceptable locations around the originally conceived location. In brief, this measure indicates if, through the editor, it was possible to "improve" the position of an object beyond what the user decided in mind at first: a "yes" value means that the real-world location obtained is not satisfactory enough, while a "no" shows that the location could not be improved more than this.

- *Type of content*, indicating the nature of the content to be inserted (e.g. bidimensional image, 3D mesh, ...), since there may be cases in which positioning particular virtual objects requires more effort than dealing with other ones.

Regarding the above definitions, unfortunately there are not authoring tools that could be directly compared to our editor application. For this reason, we will have to compare it to what we define "manual insertion" of virtual content, a concept that slighty changes depending on the context of the application and that we will have to redefine each time before presenting the results of our experiments.

We would like to precise that the author of this document, who can be considered an experienced user of the the system, is the only person who has been tested in order to collect performance data about the Editor application. He is also the one in charge of deciding if a virtual content is perceived in the correct position, according to how much that specific content is aligned with the established narrative (e.g. some objects may require to be in very precise positions, while the location accuracy of others is meaningless to the MR experience). Due to the complexity and subjectivity of the evaluation method for the Editor application, motivated by the novelty of our tool and by the absence of similar systems, we reserve ourselves right to modify this approach in the future.

Ultimately, we want to note that the values defined in this section may be possibly measured in a slightly different way in the two applications: where

necessary, further details will be specified to explain our choices. Other significant measures, like the network performance for the real-time editing feature, won't be analyzed in this chapter: since they are specific of our approach and do not change in the two applications here proposed, they have already been discussed in their respective sections in *Chapters 3* and *4*.

## 5.2 Chicago 0,0

The *Chicago 0,0 Riverwalk* AR experience provides a novel, interactive way for users to explore historical photographs sourced from museum archives. As users walk along the Chicago River, they are instructed to use their smartphone or tablet to view these photographs alongside he current views of the city. By matching location and view orientation the *Riverwalk* application creates an illusion of "then and now" co-extant. This superimposition of the historical photographer's view and the user view is the basis of educational engagement for the user and a key factor in curating the images and the narrative surrounding them, facilitating a meaningful museum experience in a public, outdoor context.

As such, creating the AR experience involves a complex back-and-forth between 3D and 2D experiences of locations: the historical images are 2D, taken from specific locations through specific optics and views; the user, present at the real-world 3D location is orienting their own camera and 2D phone video screen in space in order to achieve the AR experience and illusion; and the ultimate experience of superimposition is one of seeing two integrated views simultaneously — the stream of data taken from a smartphone's camera and the historic image. The site-specific nature of the publication makes it necessary to utilize a virtual 3D environment in which to place 2D augmented content that enhances the experience. This content needs to be placed in such a way so that the designer can accurately visualize what the user will see on screen from a particular location.

The first episode of the *Riverwalk* AR experience focuses on a single block between N. LaSalle and Clark Streets; the site of the Eastland Disaster in 1915. The site was selected because of the importance of this historical event — the sinking of the Eastland cruise ship 100 years ago was the largest single loss of life in Chicago's history — and because of the abundant media available in the archive including extensive photographic documentation, newspapers and film reels. Moreover, the site potentially offers natural wayfinding characteristics that are amenable to developing our project, including a newly built pedestrian walkway along the river with


*Figure 5.1: Chicago 0,0 is a mobile application aimed at presenting 2D media content from archives of historical media through the use of augmented reality.*

viewing platforms, views from pedestrian walks along the bridges, and historical markers. Additionally, the urban environment offers many features providing sufficient quality for robustly tracking user location based on their cameras from a variety of views and positions. This is a typical scenario in the creation of public outdoor AR experiences; the quality of content and narrative are the primary motivations for their creation.

There is only one major architectural site that is available from all vantage points along the Riverwalk, the Reid, Murdoch & Company building on the north side of the river (a historic building that is also featured in the archival photographs of the disaster). However, there are also smaller tracking-adequate features along the site — mainly signage as well as more distant buildings that could be used for tracking from a more limited number of views. Inversely, the archival photography was captured from a variety of angles: from the riverside, from the bridges, from boats on the river. In order to match rich historical imagery with a user's current orientation, a form of "extended tracking" is necessary and is part of both the design of the primary Riverwalk experience as well as the in-app user guidance in which users are directed to obtain and calibrate tracking by pointing their cameras at the Reid, Murdoch & Company building.

Tracking performed by the ARCamera is extended beyond views of this

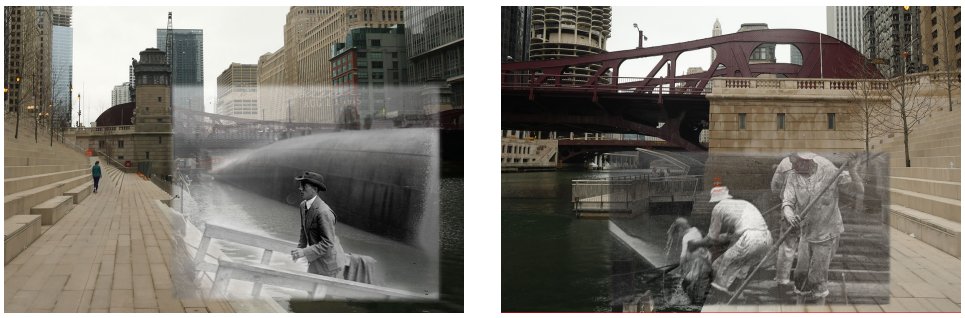

*Figure 5.2: Two sample photos from the historical archive superimposed on the live camera stream. In order to create an appealing augmented reality effect, the two photos need to appear aligned with specific environmental features (in this example, the edge of the river) and need to be seen by the user from a particular point of view. On the left, the half-sunken ship, the Eastland, can be seen placed accurately in the river the exact location it sunk 100 years ago.*

one building by exploiting the SensorCamera introduced by our approach. The two virtual cameras need to calibrate and intelligently turn on and off depending on the users situation and the content available for augmentation. Though existing platforms, like Vuforia, include extended tracking behavior, our method is able to create a more robust extended tracking technique that can be used in an "always on", mode suitable for public outdoor AR situation. In addition, because of its real-world application for a specific site and experience, we want to enable customization of the tracking behavior based on known variables and constraints for each specific site.

Below we will present how we applied our approach to developing a mobile augmented reality system that incorporates 2D historical photographs, analyzing how domain-specific considerations can be used to improve both the tracking and the user experience. We will also show how our authoring tool is fundamental for easily geolocating content in this type of public outdoor projects.

### 5.2.1 Background and contribution

Available public platforms for image-based AR — such as Layar [47], Daqri [20], Aurasma [10], Vuforia [96], and ARToolkit [8] — focus on robust image tracking for 2D features, and generally work well for print publishing and advertising campaigns that incorporate AR components. Research in natural feature detection for 3D space, and particularly architecture, has focused on surveillance and military applications [41]. Though museums, city arts councils, and tourist boards are interested in creating public AR exhibitions

within the urban landscape, there does not yet exist a platform optimized for the challenges that are presented by public outdoor contexts in urban environments.

Despite previous works that utilize location-based augmented reality with the support of mobile sensors (such as the Andy Warhol Museum's *Geo Layer* and the Museum of London's *Street Museum*, both of which highlight geolocated media archives within the urban community), in our case the need for accuracy prevents us from relying solely on GPS and mobile sensors for positioning content. The desired illusion is inherently about matching and alignment rough, "floaty" approximation would defeat the goal. In fact, while it has sufficient accuracy in open spaces, its performance degrades significantly in urban environments [75], since shadowing from buildings and signal reflections greatly reduce its availability. At the same time, inertial sensors are often prone to drift, while local magnetic fields encountered in urban environments may disturb magnetic sensors on mobile devices. Other approaches rely instead only on markerless augmented reality techniques. For instance, *Tidy City* [98] and *City-Wide Gaming Framework* [103] generate urban scavenger hunts by leveraging existing platforms to detect image patterns corresponding to the facades of the desired buildings. However, this general-purpose approach has drawbacks related to the recognition of 3D landscape features from multiple viewpoints, the recognition of features that are changeable by lighting, weather conditions, and man-made interventions. In the latter case, for instance, oversampling can be used for storing multiple tracking images for each building in order to overcome environmental changes. At the same time, many architectural features include flat surfaces, repetitive patterns, and shiny materials, properties that require more sophisticated detection algorithms.

In order to enable accurate, real-time overlays for a handheld device in urban environments, other works have tried to combine different approaches. Art et al. [7], for instance, propose a method for estimating the 3D pose for the camera using only untextured 2D+height maps of the environment, to refine a first estimate of the pose provided by the device's sensors. In order to obtain this result, the leveraged image processing for detecting straight line segments aligned the 2D map with a semantic segmentation of the input image. Ct et al. [16] note that augmented reality fails to provide the level of accuracy and robustness required for engineering and construction purposes and present a live mobile augmentation method based on panoramic video. In their system the user manually aligns the 3D model of the environment with the panoramic stream, thus avoiding sensor calibration issues. Takacs et al. [85] approach the problem by using an adaptive image retrieval algo-

rithm, aimed at creating a database of significant features that are continuously updated over time, in order to reflect changes in the environment and to prune features that may be seen as outliers. Their system relies on geotagged data collected by many people from different locations, at disparate times of the year and day. The tracking method proposed by Reitmayr and Drummond [75] combines several well-known approaches, providing an edge-based tracker for accurate localization, gyroscope measurements to mitigate problems arising from fast motion, drift compensation by measuring gravity and magnetic field, and automatic re-initialization after dynamic occlusions or failures. However, it does not take advantage of geolocated content and does not address the way in which this content is provided to users in real-world contexts.

### 5.2.2 Implementation

Considering all the environmental problematics we just mentioned, our approach concretizes in *Chicago 0,0* with giving more priority to the ARCamera, with the decision of not trusting solely the device sensors for te display of virtual content. The general flow of the application consists in roughly guiding a user towards a desired point of view through GPS and then suggesting him which content is available around him, explicitly telling him how to point his device camera at fiducials; once tracking is established, the ARCamera estimates the device position and orientation without the need for GPS and, through some guidance provided by the GUI, the user is directed towards content by leveraging the rotations estimated by the SensorCamera. Due to the intrinsic nature of 2D content positioned in 3D space, archive photos can be correctly seen only from a specific perspective and angle, so we defined in our configuration a way to limit the appearance of content only to realistic viewpoints. In the following paragraphs we will deal with the project-specific features and issues related to *Chicago 0,0*, which mostly involve site-specific considerations, the definition of a smart graphical user interface to guide tourists towards content and the need of providing multiple copies of the same fiducial to mitigate lighting and weather conditions.

#### Site-specific and user-based considerations

In real-world applications, an AR system does not need to perform in all areas without any user input. Instead, it is typically advantageous for the system to be limited to only function at specific sites, under specific user-generated conditions that result in the best experience. These limitations are complementary to natural limitations in the environment (naturally occur-

ring wayfinding characteristics on site) and available content; they become part of the user experience design as well as curatorial narrative. Because different aspects of the AR experience are created by different team members—graphic designers, photographers, writers, curators, programmers, etc— it is highly useful for the AR behaviors to be customizable by site, so that the application design can incorporate existing knowledge about the sites and desired AR experiences on a project by project, site by site, augment by augment basis, giving this ability to all primary team members (not just programmers).

Though in practice the jobs of creating an AR experience are frequently segmented, we introduce a more holistic approach to AR experience design, recognizing that the archival research, the narrative writing, the experience design, the onsite photography, and the creation of an AR platform and architecture share challenges and goals that are motivated by both the desired content and real-world sites. This is true in designing the experience, where technical considerations of bad tracking can be countered by designing in to the application user direction and onsite wayfinding, as well as in the graphic design and fabrication of AR illusions and images, which are based on the same spatial, architectural, 2D photographic overlays of 3D space that must be represented in the AR design platform. The use of actual coordinates, simple interfaces, and sharing of site photography and naming conventions allow a back and forth communication of information between programmers and designers.

### Navigation and information browsing

Our user-centered approach to AR leverages all available and site-specific information to guarantee an intuitive user experience. This reflects mostly in the graphical interface with which the user interacts during the AR experience, and which relies on the new possibilities offered by our approach.

We opted for a minimal user interface in order to dedicate more screen space to the AR experience and make the user feel more engaged. with a simple swipe, a slider containing information about nearby virtual content may be activated on the right side of the screen: points of interest are color-coded and grouped based on their location or historical relationship, allowing the user to explore all related content before moving to a different area. The sequencing of the content mirrors the linear movement along the river. When the user selects a point of interest, a popup shows where he has to aim his mobile camera at in order to enable tracking, as shown in Figure 5.3. The AR experience relies on directing the user to a location and

view orientation that matches one by a photographer decades ago; the app UI seeks to communicate this positioning, and the number and breadth of locations available, in a highly visible way. A minimap at the bottom of the slider may be toggled on and off, showing points and areas of interest, respecting the color convention mentioned above.

When the user has reached a specific area or has focused his camera on a fiducial, different types of interaction may be enabled. For instance, the fiducial itself can be a piece of architecture, onto which an overlay with related historical imagery is superimposed. Narrative textual content may be added to the experience through annotations, historical descriptions or pre-recorded audio. Alternatively, directing the user aim at a fiducial can even have the purpose of tracking his position with a higher accuracy, in order to show him content that is not necessarily in the direction in which he is looking. For example, we can make our users look at the facade of the Reid, Murdoch & Company building because of its robust tracking, and then rely on relative rotations in order to display overlays 90 degrees to the left, where the bridge and the skyscrapers in the background would not allow an acceptable tracking. An in-app UI can direct the user to this view. We can rely on narrative audio, visual and textual annotations to guide the user towards the desired content, indicating for instance to turn his device to the left until he reaches the desired orientation. By leveraging the absolute rotation in space of the device, we can also know when the user suddenly puts down his mobile device and as a consequence we can dismiss eventual indications. If an incoherency from the mobile sensors is detected, some instructions guide the user back to the fiducial, in order to re-establish a robust tracking; in particular cases of sensors unreliability (e.g. in presence of relatively strong geomagnetic fields), some suggestions will be displayed on screen explaining how the user can perform a calibration procedure by moving its device.

When multiple contents are closely available from a particular point of view, an orientation threshold is used for determining how to update eventual annotations when a user is moving his attention from the previous overlay to the next one. In particular, there are many cases in which two or more overlays may appear overlapped one to each other. This eventuality is dealt by considering the angle at which the contents overlap and showing a colored dot that, when pressed, activates a transition between one overlay to an adjacent one, changing their opacity accordingly. A similar very common case happens when from a specific point of view has multiple overlays that need to be displayed in the same position, so that they completely hide each other. Here we decided to allow the user to see one of them at a time,

displaying the availability of multiple content and creating a transition to the next overlay when the user touches the screen.

Additionally, we consider a feature that allows users to correct for themselves their position or the camera orientation is case of poor tracking or misaligned content. Asking explicitly to the user to manually improve tracking could represent an interesting innovation for AR applications for the general public, but we reserve to study this behavior more accurately in the future and in user testing.

Ultimately, we consider fundamental the presence of a static user interface for letting the user visualize the content if the environmental conditions do not allow tracking or a realistic AR experience. Weather conditions like fog or rain or simply the absence of light may cause the tracking not to work properly or the overlays to appear inconsistent with the real scene. More importantly, non-AR based methods to display the content created for the experience a broad range of images, overlay illusions, site photography, historical annotation, audio and textual narrative allow this content to be viewed off site, in other cities, countries, or simply at the user's home. In these cases, we allow the user to activate a more classic static user interface where he can access the historical imagery and the available narrative content. For instance, Figure 5.4 shows two sample interfaces through which the user can select available content, organized according to the site geography.

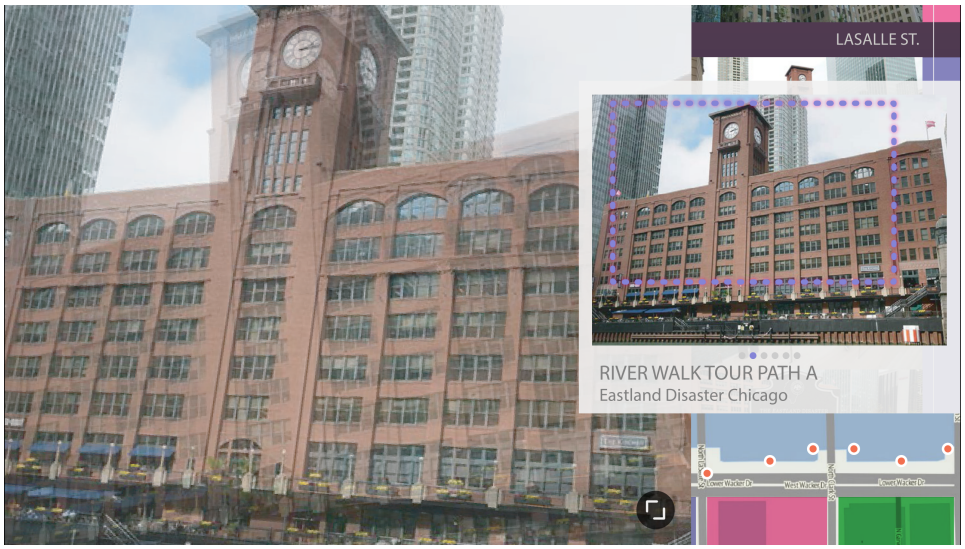


*Figure 5.3: Our approach involves also leveraging as much as possible a simple yet smart user interface for allowing the user to better experience AR augmentations. In the picture above, a popop indicates where the user should aim his device in order to activate an augment.*

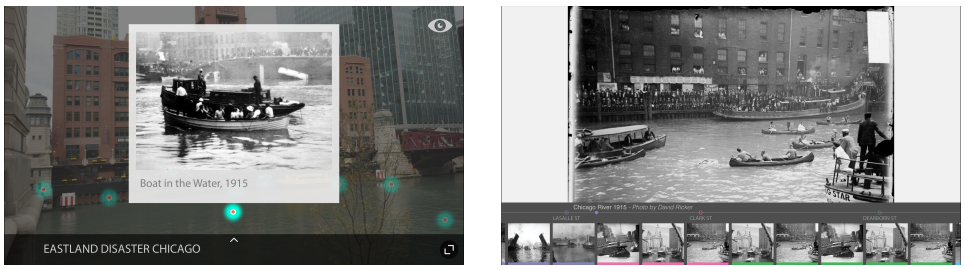


*Figure 5.4: On top of the augmented reality features, we added support for a linear reading experience, that can be activated within the same app. The image on left allows the user to select manually among different photos overlayed onto a single site view, while the image on the right shows a slider navigation for a browsable set of spreads of images and text with color-coded photos related to locations.*

### The problem of oversampling

As already mentioned, the low accuracy of sensors that characterizes the urban environment in downtown Chicago forces us not to excessively rely on the SensorCamera, creating the need to find good features to track around the overlays that we want to show. The big size, unchanging position and availability from many different views of some buildings (the Reid, Murdoch & Company building in particular) seem to make them ideal fiducials for our purpose. However, if we recall the principles for good tracking that we introduced in *Paragraph 3.2.1.1.1*, we soon realize they often do not respect properties such as high texturization, good local contrast nd even feature distribution; sometimes skyscrapers also present reflective surfaces and the displacement of windows creates many repetitive patterns, that can confuse the tracking implementation. Additionally, in order to be used as pattern images, buildings need to be photographed as flat surfaces, requiring some effort to produce orthographically aligned patterns, without perspective distortion.

Despite all these issues, surrounding buildings still represent a good feature for tracking, mostly because the intrinsic displacement of virtual elements along the river walk does not provide us close trackable features: if they are too far, they need to be of a certain dimension, otherwise the tracking algorithm would not be able to detect them. However, problems related to tracking to not end here: most of the key points detected by many algorithms reside within the windows of the buildings, whose state may be changed during the day from open to close and viceversa, thus changing the set of feaures of the building; secundarily, the changing of the lighting conditions create big moving shadows along the surfaces of buildings, misleading

the tracking process. Finally, buildings appearance is significantly affected by weather conditions such as rain and snow.

Similarly to other approaches we presented in *Section 4.2.1*, we tried to collect features from the buildings in more than one circumstance, in order to have more sets of keypoints to be searched for correspondences in the live camera stream of the mobile application. Our experiments showed that, by using Vuforia, at least 7 different image patterns, recorder at different times of the day and in different environemntal conditions, need to be provided to the tracking algorithm in order to always recognize the building (excluding night time). Kudan AR demonstrated instead to accomplish this task with only 3 to 5 samples, however with a lower precision with respect to Vuforia: it seems that Kudan AR has a threshold for recognition involving less features.

Unfortunately, the tracking algorithm provided by these pre-packaged libraries cannot be modified and even pre-processing the image given as fiducial would not work (e.g. removing uninteresting parts or elements affected by daily changes, restricting the number of features), since then the algorithm would search for different features when processing the raw camera stream. So for now we have temporarily resolved this issue with oversampling fiducials and feeding the tracking algorithms with more images of the same building, however we are currently working on two better solutions:

- Creating our own algorithm for tracking buildings, to be implemented together with the OpenCV algorithm we proposed in *Paragraph 3.2.1.1*. With this method, we could freely do pre-processing both on the pattern images and mostly on the live camera input, thus feeding the algorithm with a simplified representation of buildings, aimed at keeping only invariant features. Among the many features of a building, we could consider for example its outer shape, number of windows or alternatively directly eliminate problematic elements like the glass inside windows.

- Leveraging the additional type of tracking presented in *Section 3.6*, based on features calculated at real-time through SLAM, thus completely eliminating the problem of changes over time. This approach will be explained with greater detail in *Section 5.4*.

### 5.2.3 Discussion

This section is aimed at showing and analyzing some of the data we collected during our onsite tests of the application and during its design through our authoring tool.

### Mobile application

In this experiment proposed below, we consider the use of our *Chicago 0,0* application for the visualization of 10 overlay images differently placed along Chicago Riverwalk. In 5.1 it is possible to observe the data we gathered during one sample execution on a LG G3 smartphone device, within a single session of approximately 42 minutes.

| Content / Accuracy | $e_{gps_{est}}$ | $e_{gps_{real}}$ | $e_{loc}$ | $e_{comp}$ | $e_{or}$ | Fiducial | SLAM |
|---|---|---|---|---|---|---|---|
| 1 | 8.3 m | 5.4 m | 1.3 m | 23° | 4° | Building | No |
| 2 | 7.7 m | 4.2 m | 2.0 m | 9° | 8° | Building | No |
| 3 | 9.2 m | 6.7 m | 2.2 m | 33° | 16° | Building | No |
| 4 | 8.9 m | 4.8 m | 1.7 m | 14° | 6° | Building | No |
| 5 | 10.4 m | 7.2 m | 1.9 m | 126° | 14° | Building | No |
| 6 | 8.6 m | 4.3 m | 2.4 m | 21° | 11° | Building | No |
| 7 | 9.1 m | 5.3 m | 3.4 m | 93° | 5° | Building | No |
| 8 | 8.4 m | 5.8 m | 1.5 m | 67° | 8° | Building | No |
| 9 | 9.0 m | 6.0 m | 0.2 m | 23° | 2° | Panel | No |
| 10 | 8.9 m | 6.2 m | 0.3 m | 42° | 7° | Panel | No |

*Table 5.1: Estimated and actual accuracy with which each virtual content has been rendered while running the Chicago 0,0 application. The meaning of each symbol is explained in detail in Section 4.1.*

It is possible to see from the second to last column of the table how the content that needs to be rendered is bidimensional and is characterized by the problem of showing it in 3D space, possibly overlapping with its respective real-view of the city. The two photographs shown in 5.2, for instance, are situated right next to river and need to be aligned to its edge in order to make them appear realistic form a specific point of view. To achieve this effect, we would need a tracking that is as accurate as possible, since an unprecisely positioned overlay would not respect the correspondences between the photograph and the live camera stream. In certain cases, some photographs need to be positioned on still existing buildings that can be used as fiducials, providing an acceptable tracking. However, most archive photos are related to environmental elements that nowadays do not exist anymore or need to be placed in a location where there are not many trackable features available.

As we can observe by the $e_{gps_{est}}$ column of 5.1, it is unfeasible for us to rely on the position given by the SensorCamera alone: with an estimated horizontal positioning error of almost 10 meters, the overlay could appear positioned in a completely different place with respect to the desired one.

This is mostly caused by the presence of many skyscrapers and tall buildings that obstruct the GPS signal and cause its reflection. On top of this issue, column $e_{comp}$ shows how unreliable is the orientation natively provided by the device, that literally goes crazy due to the high geomagnetic field perceived in this area and leads in one case to an incredible error of 126°: it is thus inconceivable to use the absolute pose provided by mobile sensors, we necessarily need tracking.

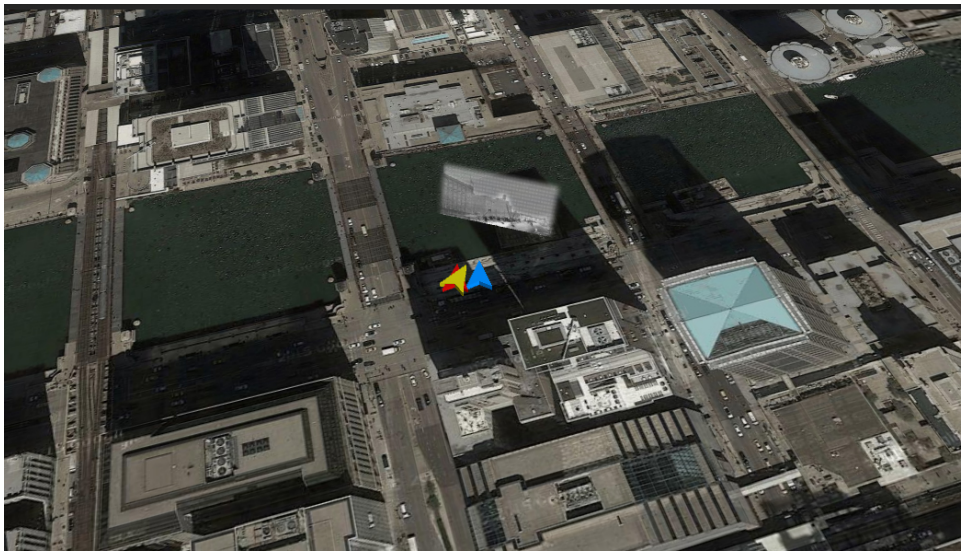

*Figure 5.5: In the above sample scenario, the user is situated along Chicago Riverwalk and the only available fiducial is the façade of the Reid, Murdoch & Company building; the application has to show an overlay with a sinking boat, appearing on the surface of the river. In the scene three arrows are represented: the blue arrow represents the pose of the SensorCamera, the red one is associated to the ARCamera and the yellow one shows the MainCamera estimated pose. As the user aims at the façade of the building, the ARCamera becomes enabled and estimates the position of the user, possibly more accurate than the one obtained from the SensorCamera. When the user turns right to see the augmented content, the tracking is lost and the ARCamera is disabled, however the computation of the $\triangle r$ matrix (in combination with the current SensorCamera pose) allows us to extend the tracking and render the content anyway the estimated pose of the MainCamera.*

As far as good tracking features, this particular urban environment has unfortunately many issues: differently from the properties a good fiducial shoud have (that we presented in *Paragraph 3.2.1.1.1*, the area is mostly characterized by buildings with repetitive patterns (i.e. many identical windows) and sometimes reflective surfaces; another component of the environment is the river, that however cannot be used as fiducial since its dynamic,

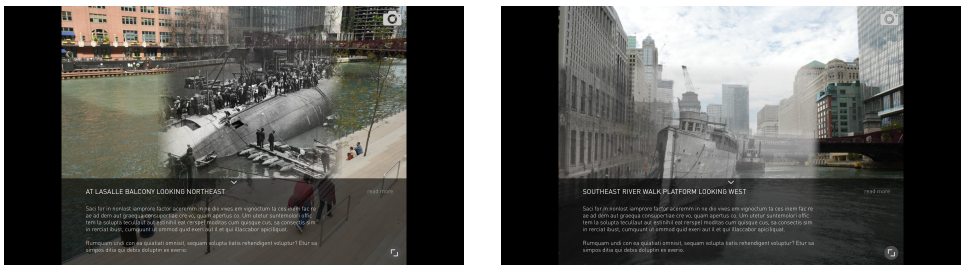


*Figure 5.6: In the above figure, two sample overlays are rendered on top of the live camera video thanks to our dual camera approach. On the left if is possible to see the the result of the scenario already described in Figure 5.5, while on the right a different boat is visible from a viewpoint located on a bridge.*

too uniform and has little contrast. Most of the minor elements in the scene are generally too small to be detected and thus cannot be used either; ultimately, candidate fiducials should be considered also in relation to the passing of people, which may occlude their visibility and make tracking unpractical. Through the method we explained in *Section 4.2.2*, we can rely anyway on few sgnificant buildings, which are fortunately available from many different views. For this reason, our goal becomes to leverage the graphical interface we introduced in *Section 4.2.2* in order to direct the user towards one of this buildings and then use them as fiducial to estimate his position: according to column $e_{loc}$ of 5.1, our approach is able to reduce the horizontal accuracy to few meters, even without using the GPS. For content that needs to be overlayed on top of the buildings used as fiducials, rendering is pretty straightforward and can easily rely on the pose estimated by the ARCamera. Virtual content belonging to locations without trackable features needs instead to merge the information from the two helper cameras: after the mobile phone has detected the fiducial, the user interface suggests to rotate the device towards the target content, aware that tracking will be lost; when this event happens, the delta rotation registered by the mobile sensors is used to continue rendering even with no tracking. So, the value present in column $e_{or}$ can be referred directly to the rotation error of the ARCamera (if a fiducial is still visible) or to the rotation error caused by mobile sensors after the fiducial has been lost. The values obtained thanks to our approach are significantly smaller than the ones shown in column $e_{comp}$, but not all are very close to zero yet due to many hardware and possibly software reasons. However, considering we are dealing mainly with big floating billboards, occupying a good part of the device screen and situated at distances from the user that in most cases range from 15 to 60 meters, such an orientation error is slightly perceivable by the user and the overall

mixed reality effect appears to be realistic.

A side note needs to be made with respect to the fiducials used in *Chicago 0,0*: since the content needs to be rendered at a certain distance from the user and generally there are no close features to be tracked (most of them are on the other side of the river), the fiducials need to be very big in order to be tracked by common libraries su as Vuforia or Kudan AR. If we consider that the live video texture resolution is often reduced to 640x480 pixels before applying the image processing algorithms, it is easy to understand that a small object on screen won't have enough pixels to be recognized as a tracked pattern image. This is why the use of buildings as fiducials was necessary. However having so distant trackable features has two drawbacks: the accuracy of the pose estimated by our algorithm decreases and small fluctuations of the ARCamera, despite being of few degrees, can make virtual content move by many meters if its very distant from where the fiducial is positioned. For instance, we can see how the values of $e_{loc}$ and $e_{or}$ are much smaller in rows 9 and 10 of 5.1: this is mostly due to the fact for those two overlays we relied on tracking two signage panels along the river walk, with a size that is of about one meter compared to the 90x63 m that characterized the Reid, Murdoch & Company building fiducial. Luckily, our smoothing function takes into account distances and provides an easy solution to stabilize these small erroneous movements done by the ARCamera.

### Editor application

Building the environment for *Chicago 0,0* was not an easy task, because of two strong requirements related to the accuracy and realism of overlays: archive photographs need to be precisely oriented in 3D space in order to appear in a certain way from a user perspective and fiducials need to be properly sized in order to guarantee an accurate position estimation, especially considering the problem of long distances mentioned above. Actually, it is even very difficult to conceive this situation without defining an approach like ours with a dedicated authoring tool. The common way to handle the problem of overlapping images to real-world views involves associating those photos directly with the pattern images generated from those views, that are then used for tracking: however, this type of approach would not allow visualizing content in the absence of fiducials, like our approach does; on top of this, it would not allow a comprehensible spatial representation. For instance, 5.7 shows how our entire application would appear in Unity Editor with the usual Image Target approach provided by Vuforia: the designer would place the fiducials inside the Unity scene, where their

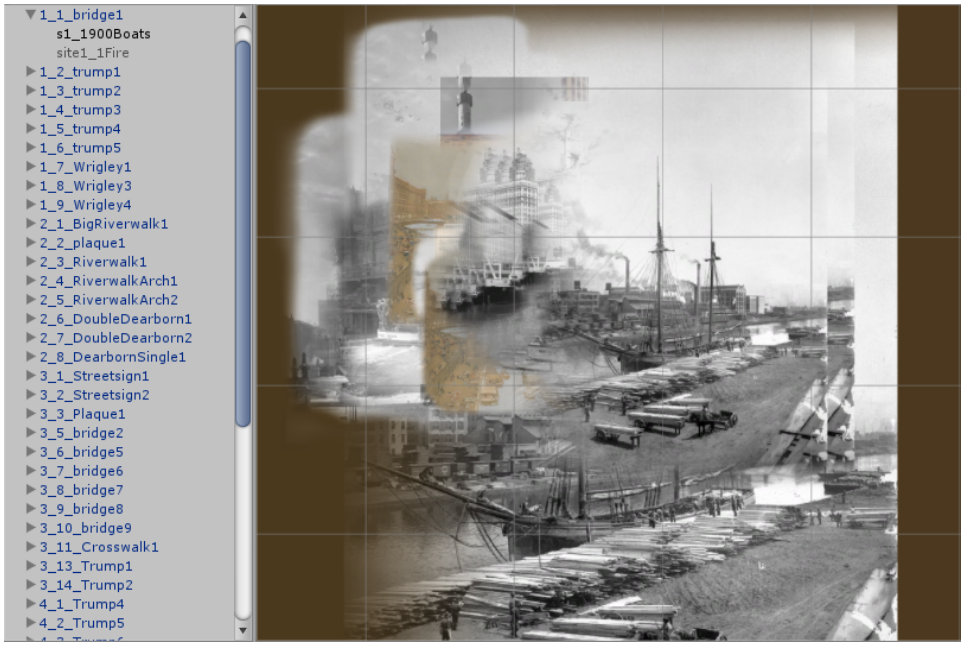


*Figure 5.7: A screenshot of how the Unity Editor would appear with the traditional conception of Image Targets: on the left side, it is possible to see a list of 32 tracking images, that are represented in 2D on the right inside the Unity scene, where the designer tryes to make them overlay correctly with their respective fiducial. Since the camera pose estimation is only relative and the content is selectively switched on and off, their position in space is completely irrelevant and for this reason the lack of conscientiousness leads to a very confusonary editing environment.*

position is irrelevant, and would then try to overlap in a 2D environment the archive photos to their respective pattern image. It is easy to notice how the editing environment can become very cluttered when many images are added and the absence of the notion of space makes the content shown apparently meaningless to the overall MR experience. Additionally, as we already mentioned, this method would only allow to position overlays on top of a fiducial, which is very limiting in our case.

In the early phase of the development of *Chicago 0,0*, we greatly made use of one of the alternative implementations of our authoring tool: the one that in *Section 3.3.8* we defined as "Unity Editor implementation", of which it is possibile to see two screenshots in 5.8. This decision was originally taken due to the early stage of the development of our CAVE2 editor and to the need to make the tool remotely accessible to other designers, for whom the availability of a personal CAVE2 system is simply inconceivable. We will briefly describe below this first editor implementation and then compare it

with the current one.

In the in-Unity implementation of our authoring tool, we defined two different views: a *User mode*, a 1:1 scaled simulation where we can preview what the user would be able to see from a particular perspective, and a *Map mode*, a 1:100 scaled map representation with a 3D perspective top-view of the previous mode. In both modes, the designer is able to move a virtual camera representing the mobile camera of a user running our application: we can move and rotate the camera around the virtual scene as a user would walking on the Chicago Riverwalk and rotating the device to see points of interest. This way, we can explore the scene from many perspectives and preview how overlays will appear in a user's mobile device. In both modes, both fiducials and overlays are located in space, in correspondence to a specific geolocation: this allows us to position accurately our overlays with the help of a reference map but also to define the correct position in space of eventual fiducials, whose accuracy is fundamental for the *ARCamera* to estimate the pose of the camera in absolute coordinates. By switching to *Map mode*, we are able to see the position and orientation of the virtual camera and position exactly the points of interests needed to build a correct perspective of the overlays. We decided to adopt a slightly 3D perspective even in the *Map mode*, since otherwise it would be hard to distinguish 2D vertical overlays from a pure top-view. In both modes, overlays and fiducials can be moved in space, scaled and rotated; changes in one mode are reflected in the other. At the end of the design process, each virtual object will be characterized by a geolocation, an orientation and a scale and the data representing the whole scene will be stored in a JSON file for future modifications.

In this simple implementation, overlays and fiducials are moved, rotated and scaled manually through Unity's built-in functionalities and one script automatically takes care of registering all modifications correctly. Despite is method is visually very helpful and conveys greatly the user perspective to test the correct overlapping of photographs, it has two main drawbacks: the precise scaling of fiducials requires a certain effort, since their height needs to be computed externally (i.e. if using a building as a tracking pattern, we need to first estimate its height in meters), and at least some other images should be added to the scene to "recreate" part of the environment. Overall, both these issues caused the horizontal accuracy of the estimated camera pose to be between 5 and 10 meters and, eventually, also slightly erroneous correspondences between the overlays and the real-world view. However, it proved to be efficient if used in combination with our mobile editor implementation, which allowed us to go onsite and correct manually

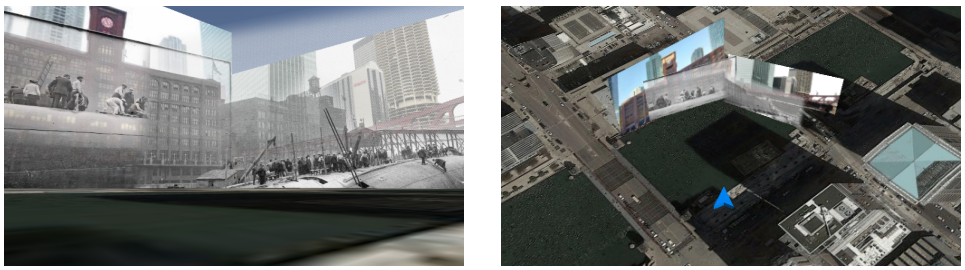

*Figure 5.8: Two sample screenshots of the early virtual environment that we created inside Unity in order to design our application Chicago 0,0. In User mode (left) we can move and rotate a virtual camera inside the environment and preview offline how a user would see the overlays from that perspective. At the same time, Map mode shows a pointer indicating our current position and orientation, allowing us to position ourselves in the desired positions. From both modes it is possible to move, scale and rotate the overlays to personalize their appearance from a particular perspective.*

from a mobile device the possible imperfections generated by our tool (see *Section 3.8.8* for more details about this implementation). Anyway, despite more precise, using two editor applications and the need of going onsite generally required a considerable amount of time to desgin and customize a MR experience.

Our CAVE2 editor implementation provides all the functionalities of the in-Unity implementation mentioned above, but has one key advantage: it already has a 3D model of the environment, correctly scaled in order to maintain 1-to-1 correspondences with the real world. Thanks to this, when an image is positioned in 3D space, the user can directly see how it is overlayed onto a view of the environment; on top of this, a feature allows and easy auto-positioning of fiducials onto other elements (e.g. buildings) and does not require to provide manually their size, since it is automatically computed from the 3D reconstruction of the environment. On top of this, the CAVE2 user interface allows a much faster customization of the virtual objects with respect to Unity Editor.

In 5.2, we present some information regarding the task of inserting 10 archive photographs in the environment at a specified location, comparing its "manual" execution and its execution with our CAVE2 editor application. In this case, for manual execution we intend the action of inserting manually the values for latitude, longitude, orientation and size for each virtual object, making use of Google Maps [34] tools for obtaining geocoordinates and distances.

It is possible to notice how the time to insert and position an image is much lower with our editor, mostly to the problem of trying to orient man-

| Manual vs Editor | **Insertion time** | **Acceptable overlay** | **Should be improved** | **Type of content** |
|---|---|---|---|---|
| 1 | 73.5s / 32.0s | No / Yes | Yes / No | 2D Image |
| 2 | 81.4s / 29.7s | No / Yes | Yes / No | 2D Image |
| 3 | 75.2s / 34.2s | No / Yes | Yes / Yes | 2D Image |
| 4 | 70.1s / 22.3s | No / No | Yes / Yes | 2D Image |
| 5 | 69.4s / 26.3s | No / Yes | Yes / No | 2D Image |
| 6 | 50.1s / 24.3s | No / Yes | Yes / No | 2D Image |
| 7 | 92.2s / 31.1s | No / Yes | Yes / No | 2D Image |
| 8 | 60.4s / 29.2s | No / Yes | Yes / Yes | 2D Image |
| 9 | 65.9s / 28.3s | No / Yes | Yes / No | 2D Image |
| 10 | 82.3s / 27.5s | No / Yes | Yes / Yes | 2D Video |

*Table 5.2: Comparison between manual insertion of virtual content with respect to the use of our editor for the application Chicago 0,0. The meaning of each column is explained in detail in Section 4.1.*

ually images in 3D space and compute their real-world dimensions without having the possibility to preview the surrounding environment: our tool instead extracts automatically size parameters for a fiducial and enables semi-automatic alignment to building surfaces. More importantly it is more reliable: without our method, the inserted image never overlayed correctly onto a specific real-world view. Our editor, in particular, created the correct overlay in all but one cases. The second-to-last column indicates that, according to a posterior consideration about the real-world displacement of content, the virtual objects positioned with our editor were often already positioned in the best conceivable location.

Overall, our editor demonstrates to be fundamental for the positioning and previewing overlays from the user perspective. In addition this this, features such as the auto-positioning and auto-sizing of fiducials greatly helped decreasing the insertion time for content.

## 5.3 DigitalQuest

*DigitalQuest* mobile application is a videogame involving futuristic "scavenger hunts", where multiple users search for virtual objects positioned in the real world, and where each object is related to a riddle or a challenge to be solved. Each player can compete with the other participants by finding virtual objects and solving puzzles, thereby unlocking additional challenges. Virtual objects are represented by animated 3D meshes locked to a determined position in the real world. The objects are activated when a player gets within a proximity threshold and then taps the object on the screen of his or her mobile phone. In our demonstration application, configurable virtual content appears, followed by a question that must be answered in


*Figure 5.9: A sample camera view of the mobile application showing the user arriving in proximity of a virtual object. The buttons in the upright corner allow the user to change to map view and to see the current active challenge; the buttons on the bottom-left corner bring up a options. The upper-left bar indicates instead the score for the current DigitalQuest and the current GPS accuracy.*

order to be pass to the next challenge. The displayed content may consist of images, video and audio streams, graphical effects, or a text message that provides hints on how to advance in the game. The editor makes it easy to create puzzles that can be solved by exploring the surroundings of the virtual object in order to discover clues and making use of location-specific knowledge. When a participant figures out the correct solution, he or she scores points related to the complexity of the challenge and also unlocks remaining puzzles that cause new virtual object to appear in the world. At the end of the event, the player with more points wins. Fig. 5.9 shows an example of a virtual object attached to a real-world location; in this case, a public sculpture on the East Campus of University of Illinois at Chicago.

*DigitalQuest* follows our approach for adding user-defined virtual content to the real world, allowing the personalization of a "mirror world" that could be explored by anyone with a GPS-enabled mobile device simply by using the application. Though these *Quest*s were originally meant for creative exploration and mixed reality scavenger hunts, our application could also potentially support a wide range of use cases, for example: team building and team-work enhancement events, tourism, virtual galleries, and cultural heritage. *DigitalQuest* can also be used to create augmented narratives, engaging users by telling an interactive story that makes use of real-world

architecture; we also believe that it could be used to provide context-aware opportunities for learning. *DigitalQuest* can additionally be leveraged for marketing purposes, allowing companies to organize events with customized content in order to make customers more attached to the brand.

### 5.3.1 Background and contribution

Scavenger hunts are an effective means to provide a continuous mixed reality engagement that mediates between a virtual environment and the real world [49]. Traditionally, virtual content is delivered through location-based applications, mostly used indoors, or via markers or image patterns, exploiting live input from a mobile phone camera. A recent example of a mixed reality scavenger hunts is *HUNT* [51], where users seek a set of objects and scan them with the built-in camera on a smartphone, enabling the display of related multimedia content that includes images and videos overlaid on the real world view. Some scavenger hunts utilize game-based learning environments. For example, Loiseau et al. [49] propose an MMORPG videogame approach applied to the archaeological domain aimed at raising awareness of cultural heritage by providing relevant information in the virtual environment. They leverage Game-Based Learning (GBL) by immersing learners in digital environments and rely on collaboration order to enhance learning and motivation. Considering that digitalization, for better or worse, is natural for children, Doong et al. [22] explore how educational environments can enable individuals or groups of users to embark on a "quest for knowledge". *ARLearn* [88] similarly adopts a learning-oriented approach.

Scavenger hunts have also been used to orient people to new environments. Rogers et al. [78] exploit the use of pervasive AR aimed at designing a serious game for improving navigational skills in public environments. The University of Illinois at Urbana-Champaign proposed a similar idea for a team-based orientation activity aimed at orienting new students in the Department of Computer Science, demonstrating how these types of game promote a sense of community and effectively help new students getting involved in university life [86].

Some works broaden the concept of augmented reality based scavenger hunts by introducing new elements. Focusing on the growing trend of Internet of Things, *TagHunt* [30] enables interaction between smartphones and daily objects by leveraging additional technologies such as NFC. This encourages the user to interact through "hyperlinking" with the surrounding environment, asking them to look for clues in the game. Gonzales et al. [31] analyze the creative collaboration environment in their *GISHWES*

(The Greatest Scavenger Hunt the World Has Ever Seen), a scavenger hunt where each item requires the user to insert an image or a video, with the goal of making players produce art and/or help each other while attempting complex, creative and time-limited collaborative tasks.

While most of the above works focus on enabling digital content through the use of classical augmented reality (e.g. marker detection or pattern-based recognition), our framework concentrates on outdoor location-based mixed reality, making use of GPS and mobile sensors and resolving issues related to accuracy. Similar projects include for instance *Tidy City* [99], where players have to explore their city and interpret clues in order to discover new places, or *LMAC* mobile application [71], where children are able to gather geo-referenced information to learn about the environment in a playful way. Zund et al. [104] similarly propose a city-wide gaming framework that renders interactive content on top of the existing architecture, gardens, and streets.

A main issue regarding location-based mixed reality is that it is difficult to ensure the accuracy of the position of virtual objects. Accuracy is greatly affected by the measurement errors generated by the sensors of mobile devices. Our approach leverages the improved technology in current mobile phone sensors, eventually mitigating poor accuracy by enabling sensor fusion techniques to minimize the positioning error. This allows us to conceive of scavenger hunts in a novel way and provides new creative opportunities. For instance, virtual objects can be represented as animated 3D meshes even without using computer vision techniques. A similar sensor-based approach is used in *MIPos* [26], which describes how to detect the pose of a user's device by taking advantage of sensor fusion and filters able compensate sensor noise, thus avoiding excessive image processing. Using the sensors available on smartphones allows us to define a virtual scene where all objects can be located at the same time, mirroring their position in the real world. This simplifies both content management (from the designer's perspective) and introduces new interaction possibilities (from the player's point of view).

Through a web-based dedicated interface we implemented, the protocol we proposed in *Chapter 3.3.3.2* can be easily extended for creating *Quest*s, advancing previous work proposed by Wetzel [99] and Pirker [71]. In particular, our version enables more sophisticated logic to control when and where virtual objects appear. It also provides additional types of triggers, enables the use of a 3D window for positioning objects in space, and provides a preview mode for viewing the final appearance of virtual objects in the mobile application.

### 5.3.2 Implementation

As we mentioned earlier, *DigitalQuest* is mostly aimed at open green spaces, usually characterized by a better sensor accuracy, especially in relation to GPS and magnetic field measurements. In many cases, these type of environments do not have many significant fiducials to be used, but since we do not need a precise positioning of objects we can often avoid visual tracking; differently from *Chicago 0,0*, content has to be always available and there are no specific viewpoints from which the user is expected to see virtual objects. For these reasons, the estimated pose of the camera is mostly based on the values computed by the SensorCamera. As we will show in *Section 5.4*, *DigitalQuest* is also the application we mostly used for testing our third helper camera, the *SLAMCamera*. Before presenting the results of our tests, we want to explain more in detail one of the application-specific characteristics, which involves the personalization of the behavior of virtual objects.

#### Extending the protocol with behaviors

Being a videogame aimed at providing entertainment to users, *DigitalQuest* cannot simply settle to display virtual content at certain locations in the world, but needs to define ways with which players can interact with objects, to which various riddles need to be associated. Unfortunately, it is impossible to implement an editor capable of creating any possible MR application that can be conceived; however, our protocol definition allows the designer to extend the configuration file in order to define additional attributes that can be associated to every single object.

Let's start with a more detailed definition of what Quests are. A Quest can be imagined as a directed graph structure where each node corresponds to a virtual object. Like in the standard definition of our protocol, each object is characterized by a size and by position and orientation in space; it can be represented by a 3D model, an image, a video or an audio content, which can be downloaded from a server and eventually cached if not present on the local storage of the mobile phone. Each virtual object is associated to a challenge: when a user approaches to an object, he can activate it by selecting it on his mobile device; after an optional animation, a window with some instructions is displayed on screen, indicating what task needs to be performed or simply giving some clues; if the text requires the user to solve a riddle or to perform actions which lead to a result, the user is asked to insert the answer inside a textbox; depending on th type of challenge, some additional multimedia content (e.g. images) can be shown on the GUI in order to help him solve the riddle. If the user inserts the correct


*Figure 5.10: On the left, an example of a simple puzzle appearing on screen after a team reaches a virtual object. On the right, a sample map view showing the user approaching his next challenge, represented by an animation on the map (which may differ from the virtual object that will be shown in the camera view); the light blue element on the upper-right corner is instead a virtual sound zone – a particular type of virtual object that emits a sound whose intensity is proportional to the its distance from the player.*

answer, the object and the window disappear with an animation and new instuctions are given, possibly unlocking new challenges. Regarding the dependencies between objects, each node of the graph becomes unlocked when a configurable logical expression involving its ancestors is satisfied. For instance, a fourth challenge could become available only if the player has completed puzzles one and two or puzzle three, but has not yet solved challenge number five.

Since a general purpose authoring tool could never be capable of defining such fine-grained behaviors, there is the necessity to extend the protocol we previously defined. Since the main addition involves the definition of challenges and is strictly related to the single virtual objects, their definition in the protocol can simply be extended by adding special parameters, that the mobile application has to interpret by itself. For instance, each object has an associated challenge, characterized by a numeric identifier, by a message to be displayed when the object is reached (that often corresponds to a riddle, by a set of possible correct answers and by a message to be displayed after the challenge is solved (usually a hint to the next object). Some of those attributes can also remain empty, thus enabling configurations where objects are used simply to represent clues, without any puzzle that be solved. In order to represent the graph structure of relationships between objects, each challenge also has to store the list of dependecies it requires in order to become active. By considering the previous requirements, we can extend our protocol as follows:

```
"data": {
        "title": "DigitalQuest Example",
```

```
        "last_edited": 1460851200,
        "start_time": 1460973600,
        "end_time": 1460977200,
        ...     //other event information
        "objects": [{
                ...
                },
                {
                id: 4,
                "asset": "DarkSkull", //content to load
                "type": 0, //3D model
                "position": {41.867272, -87.675434}
                "rotation": {-12, 76.4, 0},
                "scale": {2, 2, 2},
                "properties": {},
                "attributes": {
                        "name": "Rosenthal",
                        "precedences": "((1&2)|3)!4",
                        "pre": "Ancient stories tell that...",
                        "answer": ["sun", "planet"],
                        "post": "The sun is strictly related...",
                        "score": 250,
                        "type": 0, //no additional content
                        "content": "",
                        "distance": 4,
                        "map": "visible",
                        "icon": "self", //representation on the map
                        "animation": "SplashDefault",
                        ...
                }
        },
        ...
        ],
        "fiducials": [...]
}
```

Application-specific parameters are simply added as "attributes" to the configuration file and *DigitalQuest* itself takes care of handling them through the desired program logic. When a Quest is initialized on the mobile application, all objects without precedences are enabled and their position is

eventually shown on a map, as it is possible to observe in 5.10. When a user gets close to an active virtual object within the specified distance, interaction is allowed and the specified animation and challenge window are displayed. When the user solves the riddle by inserting one of the possible correct answers, his achievement is stored in a separate data structure and his score is increased. The game can then proceed with the next challenges, that are eventually enabled by iterating the remaining virtual objects and checking if their dependencies have been satisfied. This way, the application can seprately have its own logic and associated behaviors to virtual content, as shown by the different UI features present in 5.9.

### 5.3.3 Discussion

Similarly to what we did for *Chicago 0,0*, in this section we will analyze some of the data we gathered during the execution of our *DigitalQuest* application, discussing if our approach allows to accuracy and realism requirements and, the same time, enables an effective authoring of the MR experience.

#### Mobile application

Differently from *Chicago 0,0*, *DigitalQuest* does not require to precisely overlap virtual content onto real-world views, but it rather aims at placing objects inside a small area of acceptance, inside which that content is meaningful to the development of the Quest. For this reason, in many cases tracking could not have the significant importance it has in *Chicago 0,0*, thus allowing us to possibly rely more on generally not very accurate measurements, like the GPS. In this experiment we will refer to a Quest we organized on the campus of the University Of Illinois at Chicago, consisting in 12 virtual objects connected to the same amount of challenges. Data has been collected from a LG G3 device, which was used to complete all 12 challenges in a single session of about 34 minutes.

From 5.3 it is possible to observe how a different enviroment, distinguished by open green areas instead of the tall buildings characterizing downtown Chicago, can significantly improve the accuracy of the Sensor-Camera, requiring the use of fewer fiducials. For example, in this case the horizontal positioning estimated error ($e_{gps_{est}}$) has dropped below 5 meters in all cases and the compass orientation accuracy ($e_{comp}$) seems to be much more reliable, with a worst case error of 27°. In particular situations in which a slightly higher accuracy is required (especially with 2D content), simple fiducials such as panels have been used in order to have a more precise positioning of the virtual content.

| Content / Accuracy | $e_{gps_{est}}$ | $e_{gps_{real}}$ | $e_{loc}$ | $e_{comp}$ | $e_{or}$ | Fiducial | SLAM |
|---|---|---|---|---|---|---|---|
| 1 | 2.2 m | 1.9 m | 1.9 m | 9° | 9° | None | No |
| 2 | 3.1 m | 2.4 m | 2.4 m | 27° | 27° | None | No |
| 3 | 2.7 m | 2.1 m | 2.1 m | 12° | 12° | None | No |
| 4 | 3.8 m | 3.2 m | 1.4 m | 14° | 7° | Building | No |
| 5 | 2.6 m | 1.8 m | 1.8 m | 13° | 13° | None | No |
| 6 | 2.2 m | 2.0 m | 2.0 m | 8° | 8° | None | No |
| 7 | 2.3 m | 1.5 m | 1.5 m | 10° | 10° | None | No |
| 8 | 4.6 m | 4.4 m | 0.2 m | 19° | 4° | Panel | No |
| 9 | 4.9 m | 4.6 m | 0.4 m | 22° | 6° | Panel | No |
| 10 | 3.2 m | 2.3 m | 2.8 m | 17° | 24° | None | Yes |
| 11 | 3.5 m | 3.2 m | 2.4 m | 13° | 11° | None | Yes |
| 12 | 2.8 m | 2.2 m | 2.2 m | 18° | 5° | Panel | No |

*Table 5.3: Estimated and actual accuracy with which each virtual content has been rendered while running the DigitalQuest application. The meaning of each symbol is explained in detail in Section 4.1.*

### Editor application

In relation to the task of inserting and positioning 12 virtual objects of different nature, 5.4 shows a comparison between using our editor and inserting the required values manually in the configuration file. This comparison is meaningful especially in comparison with other location-based AR libraries like Metaio [55] and Wikitude [100], where the user has to input manually the geocoordinates of a virtual object without being able to preview its concrete positioning in space.

| Manual vs Editor | **Insertion time** | **Acceptable overlay** | **Should be improved** | **Type of content** |
|---|---|---|---|---|
| 1 | 20.7s / 18.2s | No / Yes | Yes / No | 3D model |
| 2 | 15.3s / 20.0s | Yes / Yes | No / No | 3D model |
| 3 | 13.7s / 19.7s | No / Yes | Yes / No | 3D model |
| 4 | 16.9s / 15.2s | No / Yes | Yes / No | 3D Model |
| 5 | 14.4s / 16.9s | No / Yes | Yes / No | 3D Model |
| 6 | 15.2s / 17.4s | No / Yes | Yes / Yes | 3D Audio |
| 7 | 12.7s / 8.4s | Yes / No | Yes / No | 3D Audio |
| 8 | 23.9s / 10.2s | Yes / Yes | No / No | 2D Image |
| 9 | 17.1s / 34.2s | No / Yes | Yes / No | 2D Image |
| 10 | 16.2s / 28.7s | Yes / Yes | No / No | 3D Model |
| 11 | 14.5s / 21.0s | No / Yes | Yes / No | 3D Model |
| 12 | 19.6s / 20.2s | Yes / Yes | Yes / No | 2D Video |

*Table 5.4: Comparison between manual insertion of virtual content with respect to the use of our editor for the application DigitalQuest. The meaning of each column is explained in detail in Section 4.1.*

As it is possible to observe from 5.4, differently from the results obtained

with *Chicago 0,0*, for what it concerns insertion time our editor does not decrease much the time to position an object. This is mostly related to the low expectations that a classical location-based application would require: a user would have to check on a map provider (e.g. Google Maps) the desired location coordinates for his virtual content and then simply write them inside a configuration file. Instead, our editor requires the user to move the camera towards the desired location, add the object to the scene and position it in to the desired location, eventally modifying its scale and rotation. However, time in this case is not sorelevant, since the manual approach would not allow previewing the customization of the virtual constent with respect to the surrounding environment: it would just place the object more or less in the desired position, without guaranteeing the correct orientation and scale. That is why the column *Expected location* for the manual approach does not reflect the intention of the user, while the values referred to our editor indicate that content was precizely positioned in desired way in the real-world. At the same time, the previewing function of our editor enabled discovering more appropriate locations for content, with respect to the originally coinceived ones: the final position of varius objects, for instance, was changed to a nearby location in which the content better fit into the environment. It is also possible to note from 5.4 how insertion time depends on the type of virtual object inserted, since some of them could have different positioning requirements (e.g. images sometimes need to be orientated more precisely).

*Figure 5.11: The above picture shows how the horizontal inaccuracy, represented by the red area under the avatar, can significantly affect the positioning of virtual content on the mobile device. In this example, the object is rendered in the middle of the street, made inaccessible by the passing of cars. Thanks to our editor, the designer can detect and adjust at real-time these kind of issues.*

Among the two applications proposed in this chapter, *DigitalQuest* is the one for which we mostly tested the real-time capabilities of our authoring tool. While running a simple quest event with the same 12 virtual objects

mentioned above, we observed at real-time the situation from CAVE2. Two first interesting cases happened due to inaccuracy of mobile devices, which caused their users to spend much more time than expected in reaching a virtual object. In the first case, a user tried to reach an object located at the edge of the road, but the estimated GPS accuracy of his device was much lower then expected: 7 meters of error caused the object to appear on the road, preventing the user to reach it due to the passing of cars 5.11. It is important to remember that the position of the corresponding avatar in the editor is given by the location retrieved from the mobile device, that is not the real position of the user but the position according to which content is rendered. So, by noticing the accuracy area in the editor, the position of the avatar and the time spent in the same position, it was possible to identify that the user had correctly reached the object position, but it was still rendered far from him (possibly due to a missing GPS update). By moving the object closer to the avatar in the editor, its rendering on the user mobile application got corrected and at the same time the users who came next to him didn't have the same problem, since now the content was positioned more inwards with respect to the edge of the road. The second example is very similar and involves another accuracy problem for which a user could not reach an object, because according to his mobile application it was situated inside a building, instead of being in the garden outside of it. By considering the horizontal accuracy of that user, it was possible to make some considerations on a new possible collocation for that content, so that even low accuracy situation users could be able to reach it. Another insteresting situation that happened during our demo quest involves the presence of too many users trying to interact with the same object at the same time, because of non-ideal configuration of the precedences which enabled challenges: on top of showing a possible game design improvement, the editor in this case allowed us to move the object away from the users, disperding them and making them follow different paths. Additionally, in order to leverage even more the real-time editing features of our editor, we are going to test in the near future the new prototypal functionalities we introduced in *DigitalQuest*, which are aimed at user-to-user interaction with the possibility for users to move and customize content by themselves.

## 5.4 SLAMCamera extension

While in the previous sections we mostly dealt with our dual-camera approach, this section is entirely dedicated to the addition of the third virtual camera. The ability to position digital content anywhere, without the need

for a graphic trigger, means that MR applications can be even more versatile, contextual and realistic. In particular, we will show the additional tests we performed on the two applications *Chicago 0,0* and *DigitalQuest*: in the former case, we reused the same system to present few urban overlays in Manhattan, New York (USA), while in the latter we tested some closer-range interactions in the city center of Munich, Germany. Please note that, since our method allows to decouple tracking techniques and content definition, the usage of our authoring tool is not affected by the addition of a new virtual camera and all the discussion of the previous section is still valid. All the accuracy data presented in this section has been gathered with the same procedure we described in *Section 5.1* and with the same mobile device we used in the previous sections.

### 5.4.1 Chicago 0,0

Two of the main issues we encountered in *Chicago 0,0* were low reliability of sensors and the difficulty of tracking image patterns in an environment in which weather and lighting are critical. Despite the SLAMCamera has great capabilities of tracking real-time features, without the need of knowing in advance lighting conditions, it is unfortunately not suited for recognition. This means that once we have started tracking with the SLAMCamera, tracking would probably work smoothly, but the problem is that we simply don't know reliably where the tracking should start, for instance without detecting the interested building onto which to place the overlay. So, the only options in this case are two: 1) if horizontal and orientation accuracy allows it, try to rely only on the SensorCamera for placing an overlay, and then enable the SLAMCamera; 2) still use the ARCamera and then add the SLAM-based tracking for increased robustness.

Due to external necessities, we weren't able to perform the tests in downtown Chicago, so we decided to run our application in Manhattan, New York (USA) - a location that can be considered very similar under many aspects. We decided to place only 4 overlays in the city, measuring for each content the results in both cases 1 and 2, for a total of 4 measurements. Also, since we did not have an archive of historical photos like the ones provided by the *History Museum of Chicago*, we had to use fictitious overlays properly sized in order to fit the façades of buildings.

As it is possible to see from the two $e_{gps}$ columns, GPS accuracy in *Manhattan* appears to be very similar to the one in Chicago (about 10 meters) due to the concentration of skyscrapers, even if more open spaces close to *Central Park* revealed a better estimation (e.g. case 4 in *Table*

| Content / Accuracy | $e_{gps_{est}}$ | $e_{gps_{real}}$ | $e_{loc}$ | $e_{comp}$ | $e_{or}$ | Fiducial | SLAM |
|---|---|---|---|---|---|---|---|
| 1 | 9.4 m | 5.4 m | 3.3 m | 31° | 10° | Building | Yes |
| 1 | 10.5 m | 5.4 m | 5.4 m | 26° | 27° | None | Yes |
| 2 | 9.0 m | 4.5 m | 3.0 m | 22° | 11° | Panel | Yes |
| 2 | 8.6 m | 4.5 m | 4.5 m | 17° | 19° | None | Yes |
| 3 | 10.7 m | 8.4 m | 4.3 m | 18° | 7° | Building | Yes |
| 3 | 10.7 m | 8.4 m | 8.4 m | 23° | 26° | None | Yes |
| 4 | 5.2 m | 3.2 m | 2.1 m | 15° | 5° | Statue | Yes |
| 4 | 5.3 m | 3.2 m | 3.2 m | 13° | 17° | None | Yes |

*Table 5.5: Estimated and actual accuracy with which each virtual content has been rendered while running the Chicago 0,0 application with the triple camera approach. The meaning of each symbol is explained in detail in Section 4.1.*

*5.5*). As far as compass orientation, the average error seems to be lower with respect to downtown Chicago (and, in general, rotational tracking with the SensorCamera appears to be more stable), probably due to a smaller influence of electromagnetic fields.

While in *Section 5.2.4* we dealt with combining the ARCamera and the SensorCamera in order to estimate the pose of the MainCamera, the 4 situations listed in *Table 5.5* are used to test the two alternative solutions mentioned above: for each overlay we considered first the combination of the ARCamera with the SLAMCamera and then the combination of the SensorCamera with the SLAMCamera. In the first case for each overlayed content, an easily identifiable trackable (for instance a building well visible from different view points) was used to align the ARCamera - while the SLAMCamera was activated once the tracking of the ARCamera was lost or became unstable. This is the same idea we used in the dual camera approach, by using the values of the SLAMCamera instead of the ones of the SensorCamera in order to estimate the pose of the MainCamera. The improvement from column $e_{comp}$ to column $e_{or}$ of *Table 5.5* is very similar to the one already performed in the dual camera approach by combining the ARCamera with the SensorCamera. The SLAMCamera proved on average to be much less affected by environmental issues (e.g. magnetic fields) and less prone to drift; however, its tracking is also computationally more expensive with respect to the one of the SensorCamera and, since overlays are placed quite far in space, does not take advantage of the greater accuracy in tracking smaller movements of the user. Overall, except in situations in which we could have a particularly unstable SensorCamera orientation, the addition of the SLAMCamera in this case doesn't seem to bring many significant advantages - if we are considering distant virtual objects.

*Figure 5.12: In the above sample scenario, SensorCamera (dual camera approach, left) and SLAMCamera (triple camera approach, right) extended tracking methods are compared in a sample situation in which the user has lost a tracked fiducial (a statue - not visible in the figure) and then rotated his mobile phone clockwise along the vertical axis by about* $80°$*. In the left image, we can notice how larger rotations in an urban environment can be affected by sensor unreliability. Despite in Chicago 0,0 the addition of the SLAMCamera appears to slightly increase the robustness of the tracking, eliminating the magnetic interferences that could limit the accuracy of the SensorCamera, it cannot however replace the ARCamera: in order to obtain a reasonable image overlay, we still need to leverage the presence of a geolocated fiducial.*

The second test case for each overlay tries instead to get rid of the ARCamera and just use the SensorCamera in combination with the SLAMCamera, without relying on the presence of fiducials. Despite this apparent advantage, the initial horizontal and rotational error given by the SensorCamera in the urban environment cannot be mitigated by the SLAMCamera, which simply considers relative movements with respect to the target location. So, even in this case, the addition of the SLAMCamera doesn't bring significant advantages to the user experience of *Chicago 0,0* - if we are aiming at a precise overlay of 2D photography.

Overall, despite its capabilities of lighting-indepentent tracking and high sensibility movement detection, the SLAMCamera did not contribute its best to the *Chicago 0,0* application: it seems that for achieving a good initial pose for placing an overlay, a fiducial is needed; even after tracking is lost, the SLAMCamera just represents a slightly more robust form of extended tracking with respect to the SensorCamera, since the distance of overlays does not allow to appreciate its accuracy on small movements.

In the next section, we will instead see how the SLAMCamera is perfectly suited for an application like *DigitalQuest*, where close-range interaction with 3D models is required and accurate alignment of overlays is not necessary.

### 5.4.2 DigitalQuest

In the case of DigitalQuest, we really wanted to push the boundaries of the dual-camera aaproach, in particular with respect to realism and interaction. Keeping the same requirements we described in *Section 5.3*, extreme location accuracy is definitely not an issue, since in this case we do not need content to perfectly overlap onto views of the city. A relevant limitation of the dual-camera approach for DigitalQuest was instead related to small movements around the virtual content: depending on the GPS accuracy and refresh rate, in many situations it was good practice to define abstract or floating objects - with which to interact possibly from a certain distance, in order to make less evident to the human eye eventual tracking inaccuracies of the SensorCamera. In particular, despite smoothing and interpolation offered an acceptable compromise, the act of walking closely to an object was often unrealistic for two reasons: sudden position or rotation changes could make the user perceive the content was not fully anchored to one real-world location; movements of the user towards the object were definitely not real-time (and not even perceivable when too close to the content), giving an overall effect of low framerate. Our triple camera approach completely solves this two issue thanks to ability of the SLAMCamera to make objects stick solidly to one real-world arbitrary location, achieving an accuracy for small movements that could be even considered better than the one provided by the ARCamera.

| Content / Accuracy | $e_{gps_{est}}$ | $e_{gps_{real}}$ | $e_{loc}$ | $e_{comp}$ | $e_{or}$ | Fiducial | SLAM |
|---|---|---|---|---|---|---|---|
| 1 | 6.7 m | 4.3 m | 4.3 m | 11° | 11° | None | Yes |
| 2 | 5.5 m | 3.7 m | 3.7 m | 16° | 16° | None | Yes |
| 3 | 4.8 m | 2.6 m | 2.6 m | 8° | 8° | None | Yes |
| 4 | 5.1 m | 2.2 m | 2.2 m | 17° | 17° | None | Yes |
| 5 | 8.6 m | 5.2 m | 5.2 m | 46° | 46° | None | Yes |
| 6 | 4.3 m | 1.7 m | 1.7 m | 12° | 12° | None | Yes |
| 7 | 6.6 m | 3.5 m | 3.5 m | 20° | 20° | None | Yes |
| 8 | 4.3 m | 1.1 m | 1.1 m | 13° | 13° | None | Yes |

*Table 5.6: Estimated and actual accuracy with which each virtual content has been rendered while running the DigitalQuest application with the triple camera approach. The meaning of each symbol is explained in detail in Section 4.1.*

In particular, we placed 8 animated 3D models around the city center of Munich (Germany) and performed the same type of test we presented in *Section 5.3*. In this case we preferred to consider only 3D meshes, preferably animated and randomly moving around the environment, leveraging the


*Figure 5.13: On the left, a close-up image of a butterfly landing on a flower: despite tracking started meters away, getting closer to the virtual object preserved a certain realism with respect to tracking and perspective. In the right image, an example of content placed up in the air without the need of pattern-based tracking.*

possibility to easily define areas of movement given by our authoring tool.

With respect the results previously obtained from DigitalQuest, the horizontal accuracy has decreased, but we need to consider that this this time we are not running the application in a green area but in an urban environment. A more significant comparison would be with the values obtained in downtown Chicago, which are definitely worse than the ones obtained in Munich. This is probably due to the lower height of the buildings in the German city, which causes less signal reflection and allows cleaner GPS readings. While relying only on the SensorCamera would not be feasible in downtown Chicago, this data shows that running DigitalQuest in a city like Munich is completely acceptable, considering an error range of few meters.

As we may also notice from the table, the addition of the SLAMCamera is not correlated to the horizontal accuracy: the position where the content appears is determined by the SensorCamera and, when the SLAMCamera is enabled, it starts already shifted by the horizontal error given by the GPS. However, as we already explained, this is irrelevant to our case: the advantages of the third virtual camera consist instead in the improved reliability (no unexpected movements, at least in a static environment), sensibility (few centimeters insted of few meters) and frequency (20-30Hz instead of 2-5Hz) of the SLAM-based tracking with respect to using solely the SensorCamera. In brief, we get the same results of the ARCamera, with the disadvantage of not having a very accurate initial positioning in the real world, but with the advantage of not needing a pattern image to detect in predefined conditions in order to start rendering the content.

When placing objects in a static context, with fixed environmental features, the result we obtained were quite promising. In Figure 5.13 (left) it is possible to see the addition of a butterfly, moving along a regular pattern, on a set of flowers: despite the tracking started about 5 meters away from

*Figure 5.14: Sample test of a virtual object inserted into a scene with continuously changing environmental features. In this case, the tracking of the SLAMCamera alone becomes very unreliable due to the passing of people, making the animated zombie in the left image appear, few seconds later, in a completely different setting (right image). This can be avoided by combining the SLAMCamera with the SensorCamera.*

the target, it was possible to get close enough and see the butterfly landing on the same flower, with the correct scaling. We noticed that is even easier to have realistic objects flying at a certain height (Figure 5.13, right): to have precise tracking, we don't need anymore to make the user lock to an environmental feature (e.g. a building) and then turn to the interested area, but now it is possible to directly start tracking without patterns, as long as the scene doesn't have a completely clear sky.

In a second case, we tried to insert into *Marienplatz* square in Munich an animated zombie, for which we defined a height of 1.60m. We verified that the SLAMCamera was able to track with the correct perspective up to a distance of about 30m: all along this range, the animated content seemed to keep the correct proportions according perspective, appearing the same height as people in the scene at the same distance. Another aspect to consider is the robustness of the tracking despite the repetitive floor pattern of the square, which in the case of the ARCamera would have probably caused recognition issues. However the third camera alone is not enough in certain cases: we noticed that turning the mobile device by a wide angle or very fast caused tracking to degrade significantly; at the same time, the passing of people sometimes caused the content to be unnecessarily shifted and rotated, due to the change of tracked runtime features (as shown in Figure 5.14). In this case our solution in simply to detect when the angle between the SLAMCamera and the SensorCamera goes beyond a certain threshold due to a sudden movement of the SLAMCamera, as we explained in *Section 3.6* in this situation, we use our usual $\triangle r$ method to detect relative changes in orientation with the SensorCamera. For instance, in the case of Figure 5.14, while the passing of people could produce a result similar to the right image using the SLAMCamera alone, the detection of the unexpected

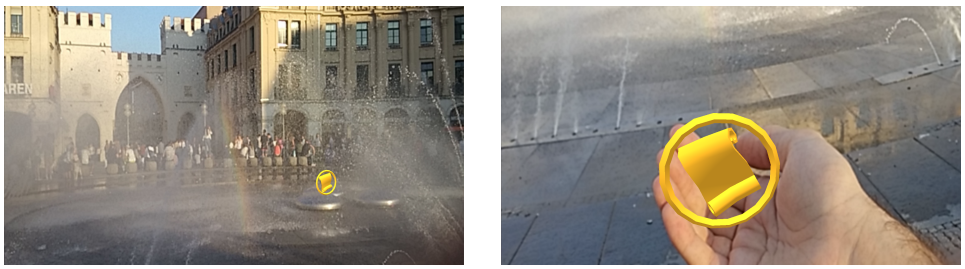

*Figure 5.15: Sometimes tracking flaws can turn out to be useful: in the above images, the user can "catch" a virtual object with his hands and bring it close to the camera, hypothetically enabling some event or interaction.*

camera movement allows us to give priority to the SensorCamera, which, detecting no real device motion, will still render the image on the left. Unfortunately, still much work needs to be done in this direction, especially if we consider the problem of occlusion: for now, even if a person passes in front of our content (i.e. between our device and the object), it will still be rendered on top of the person. Also, despite we are potentially able to define the real-world boundaries in which objects can move thanks to our editor, dynamic elements such as people cannot be modeled a-priori, thus the zombie cannot be aware of their position in space without adding some sort of real-time, dedicated image processing feature.

A relatively curious aspect we discovered regarding the SLAMCamera is that the tracking issue we just mentioned can turn out to be useful in some cases: Figure 5.15, shows how the flaw can be leveraged for "catching" objects with hands. Obviously there are no guarantees of keeping the exact same scale and orientation of the object, but it is a cheap way to achieve what a bulky Leap Motion [63] or a more complex OpenCV [67] monocular gesture detection algorithm could do on a mobile device.

# Chapter 6

# Conclusions

In this document we have presented our location-based, spatial approach to mixed reality through the implementation of both a mobile and an editor application. Differently from many augmented reality applications, our method is based on the principle of geolocating content in a tridimensional space, creating a 1-to-1 mapping between real-world and virtual world coordinates.

By leveraging both mobile sensors and visual tracking techniques through our "dual camera" approach, we are able to estimate an absolute 6DOF pose of the camera: while the user walks in the real world with his device, a virtual camera moves and rotates accordingly in the virtual space, defining what content needs to rendered on top of the live video stream acquired by the hardware mobile camera. This way, we can take advantage of the benefits provided by both location-based augmented reality and pattern-based image recognition, since defining the 3D position and size of fiducials allows us to define an abstraction that deals in the same way with the two approaches. Indeed, in *Section 3.6* we demonstrated how adding a new type of tracking as a third helper camera (the SLAMCamera) does not change the mechanics of our framework. In particular, some of the advantages provided by our approach are:

- The possibility to combine different types of tracking in a single reference system, where every virtual position is directly associated to a real world location.
- Performing extended tracking and displaying content even if visual tracking is not available (e.g. a fiducial has been lost), leveraging mobile sensors as a fallback.
- Knowing the absolute position of user and content, allowing to load

and unload content according to the user location, pruning the dataset of tracking images based on proximity, leveraging real world constraints and site-specific considerations that characterize the positioning or virtual content.

- Providing intelligent camera smoothing to remove the jitter normally provided by AR applications; in particular, we focused on stabilizing camera pose estimation in case of sudden changes in tracking availability.

- Using rotational information in order to understand how much a user is moving away from a target object, orienting him towards new content or making him return to the desired position; at the same time, our approach allows showing content dynamically to avoid overlapping in 3D space and is able to signal incoherent absolute and relative poses, in order to enable the application of corrective countermeasures.

- The possibility to define smart user interfaces, that take into account context-specific information to suggest the user how to experience at best mixed reality.

- Being easily extensible and agnostic with respect to the visual tracking and sensor correction algorithms.

Additionally, we have leveraged our spatial representation of mixed reality in order to create an editor application, aimed at easily defining the position of content and fiducials in 3D space: still exploiting the mapping we defined between the two worlds, each object inserted in the editor application is meant to appear in the corresponding real-world position when a user will look at that direction with his mobile device. After defining the user interface and the controls for customizing content in the editor application, we proposed a system architecture enabling real-time editing of mixed reality content: if the designer moves a virtual object 3 units to the left in the editor, a user in that location will see at real-time that object moving to the left by 3 meters. We showed also how a real-time architecture may be very helpful for representing users directly inside the editor: shown as avatars, they are characterized by the location and orientation defined by their mobile device, which update as they move in the real world. This representation allows us to study the behavior of users and at the same time to take design decisions aimed at "correcting their MR experience, by enabling the designer to assume the user's perspective. Starting from here, we have also explored possible new types of interaction between the designer and the

people using the mobile application, like assuming and previewing a user's perspective, streaming audio and video and introducing the designer himself inside the mixed reality world. In particular, we believe that decoupling tracking techniques and content definition is a relevant contribution with respect to the authoring process of MR applications, allowing the creation of experiences remotely without the need of knowing in advance which tracking methods will be used.

In order to prove the feasibility of our approach, we developed two case study mobile applications compliant with our method: with *Chicago 0,0* we showed how it is possible to insert bidimensional virtual content in 3D space to match specific user views, relying on extended tracking with the available environmental features due to the high inaccuracy of mobile sensors in downtown Chicago; with *DigitalQuest* we demonstrated how our protocol can be easily extend to create more complex MR experiences and we showed how our approach behaves with tridimensional content in cases in which mobile sensors are more reliable. In both cases, we also dealt with user interface aspects based on the user and content location and aimed at improving the overall MR experience. Ultimately, we explored the addition to our standard approach of a SLAM-based type of tracking, able to track environmental features detected at runtime: by applying it to both *Chicago 0,0* and *DigitalQuest*, we showed in different settings how it can improve the realism and robustness of the overall MR experience.

## 6.1 The future of MR

Despite our work mostly relies on the significant technology improvements in mobile computing that have characterized the last few years, much work still has to be done in the world of mobile and wearable mixed reality.

Mobile displays now have a much greater resolution and commercial smartphone processors may now rely on up to 16 cores allowing an incredible computational power if compared to the early 2000s. Nowadays almost all wearable and mobile devices are equipped with more accurate inertial sensors, whose drift and inaccuracies have decreased significantly thanks to the development of sensor fusion techniques. The positioning of other type of technologies, like GPS, has been improved by combining different new techniques. More and more open-source code or public libraries become available each day on the internet and a greater number of developers, considering also the rapidly decreasing cost of hardware, has had the opportunity to develop its own applications outside of research laboratories.

Despite the enabling technologies of mixed reality were defined more

than 20 years ago, only few years ago augmented reality has concretely tried to expand to consumer products and to the advertising world in particular. This has been possible thanks to all the recent changes mentioned above and almost everyone now can afford a smartphone supporting this type of technology; however, the he terminology "mixed reality" is still pretty unknown to the general public, which eventually refers to the "classical" conception of marker-based or pattern-based AR of making virtual content appear on top of some sort of fiducial - which has very limited applications if compared to the whole MR spectrum. The real innovation, in fact, has not been brought by augmented reality, but by another realm along the virtuality continuum that was a popular field of research in the 1990s: virtual reality.

At Los Angeles *Vision Summit 2016* and at the *IEEE VR 2016* conferences, this year has been (arguably) defined multiple times as the year in which VR will become available to everyone. For sure the previous two versions of Oculus Rift [66] and of Google Cardboard [21] had already started changing the way VR was developed in the 90s, but the new technologies presented with Oculus Rift v3 and HTC Vive [40] really represent great improvements for consumer scale solutions in the near future, even if their price is still too high for mass distribution. A great interest from the world of cinema, videogames and marketing has recently brought incredible amounts of money to this field; even big companies like Google and Facebook started trying to bring VR solutions to a wider public, for instance by making 360 videos available on both platforms with the support for Google Cardboard. Recent virtual reality research has focused mostly on improving tracking, especially combining sensors and visual techniques for room-scale environments, design of head-mounted displays, resolution and interaction, through new types of physical controllers or gesture based interfaces like Leap Motion [63]. A good amount of these improved technologies has many features in common with MR: indeed, only recently the term "mixed reality" has become more popular for intending something that is more flexibly oriented towards virtual reality with respect to the constrained augmented reality the public has always been used to. In this sense, Milgram's taxonomy [61] has lost part of its original meaning, since outside research labs MR often is not considered anymore a superset of AR, but a more flexible mixture between AR and VR. The most significant example that made used of the term "mixed reality" is Microsoft Hololens [57], which has been just released on the market and heavily relies on marker-less SLAM techniques for building maps of the environment: from a certain point of view, it is a headeset very similar to the the ones that have become popular for virtual reality, except for the fact it allows the user to interact with the real world. This

is why it will probably be easier to conceive mixed reality as an evolution of the current state-of-the-art well-known VR technology, more than an extension of the aumented reality few people knew about ten years ago. An interesting role will be played also by Playstation with the introduction of its Playstation VR headset [82], that will try to bring VR headsets at a more reasonable price in the houses of many people, in a similar way to how Google has done with Cardboard but aiming at a high-end dedicated solution. At the same time, the company Lenovo has recently launched the first "mixed reality enabled" mobile phone in Summer 2016, with a particular camera configuration able to support the technology developed with Google's *Project Tango* [35]. However, as we also discussed in *Section 1.2*, the goal is not just to create a technology but also to make it widely available to people: despite the recently published Pokémon Go [45] application still has a limited number of features, it had the great contribution of making millions of people more acquainted to the world of mobile augmented reality, especially in relation to monocular SLAM-based techniques - that will probably become the standard approach for the near future.

There has been a significant change in the technology in order to allow all of this, but we believe the greatest revolution is happening right inside the mind of people, which are embracing the interaction with more and more virtual content during their daily lives and are getting used to the idea of using wearable technologies that time ago ago would have been inconceivable for mass production. Both technical and sociologic progresses recently made by VR represent a significant contribute to MR, but, if we want to consider mixed reality as an evolution of virtual reality, we also need to admit that MR still has even more challenges that the ones which VR is still trying to solve. If we think about the narrow field of view offered by Microsoft Hololens, it seems almost to go back to Azuma's survey of augmented reality in 1997 [12]. The accuracy results shown by our tests describe even more how we are still distant from the ideal of mixed reality we have. At Vision Summit 2016 [1] it was predicted that, 10 years from now, mixed reality will completely surpass virtual reality in many fields and become widely available in our daily lives. For the moment, we can just say that much work still needs to be done in order to achieve reasonable results, but at least we can proudly say we are walking the right path.

# Bibliography


[1] Vision Summit 2016. Vision vr/ar summit. http://visionsummit2016.com/, 2016. [Online; accessed 08/03/2016].

[2] Adobe. Adobe after effects cc. http://www.adobe.com/products/aftereffects.html, 2016. [Online; accessed 04/03/2016].

[3] Pratik Agarwal, Wolfram Burgard, and Luciano Spinello. Metric localization using google street view. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 3111–3118. IEEE, 2015.

[4] Android. Android ndk. http://developer.android.com/tools/sdk/ndk/index.html, 2016. [Online; accessed 06/03/2016].

[5] Apple. Event handling guide for ios: Motion events. https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/motion_event_basics/motion_event_basics.html, 2015. [Online; accessed 06/03/2016].

[6] Apple. Apple. http://www.apple.com/, 2016. [Online; accessed 06/03/2016].

[7] Clemens Arth, Christian Pirchheim, Vincent Lepetit, and Jonathan Ventura. Global 6dof pose estimation from untextured 2d city models. *arXiv preprint arXiv:1503.02675*, 2015.

[8] ARToolkit. Open source augmented reality sdk. http://artoolkit.org/, 2016. [Online; accessed 06/03/2016].

[9] ARToolkit. Training artoolkit natural feature tracking (nft) to recognize and track an image. http://artoolkit.org/documentation/doku.php?id=3_Marker_Training:marker_nft_training, 2016. [Online; accessed 06/03/2016].

[10] Aurasma. Aurasma. http://aurasma.com/, 2016. [Online; accessed 08/03/2016].

[11] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. Recent advances in augmented reality. *Computer Graphics and Applications, IEEE*, 21(6):34–47, 2001.

[12] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators and virtual environments*, 6(4):355–385, 1997.

[13] Daniel Lélis Baggio. *Mastering OpenCV with practical computer vision projects*. Packt Publishing Ltd, 2012.

[14] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer vision–ECCV 2006*, pages 404–417. Springer, 2006.

[15] Hung-Lin Chi, Shih-Chung Kang, and Xiangyu Wang. Research trends and opportunities of augmented reality applications in architecture, engineering, and construction. *Automation in construction*, 33:116–122, 2013.

[16] Stéphane Côté, Philippe Trudel, M Desbiens, Mathieu Giguère, and Rob Snyder. Live mobile panoramic high accuracy augmented reality for engineering and construction. *Proceedings of the Construction Applications of Virtual Reality (CONVR), London, England*, 2013.

[17] Carolina Cruz-Neira, Daniel J Sandin, Thomas A DeFanti, Robert V Kenyon, and John C Hart. The cave: audio visual experience automatic virtual environment. *Communications of the ACM*, 35(6):64–73, 1992.

[18] Davide Antonio Cucci and Matteo Matteucci. A flexible framework for mobile robot pose estimation and multi-sensor self-calibration. In *ICINCO (2)*, pages 361–368, 2013.

[19] Davide Antonio Cucci and Matteo Matteucci. On the development of a generic multi-sensor fusion framework for robust odometry estimation. *Journal of Software Engineering for Robotics*, 5(1):48–62, 2014.

[20] DAQRI. Daqri. http://daqri.com/, 2016. [Online; accessed 08/03/2016].

[21] Google Developers. Google cardboard. https://developers.google.com/cardboard/, 2016. [Online; accessed 16/04/2016].

[22] Ji-Liang Doong, Ching-Huei Lai, Kai-Hsiang Chuang, and Chun-Chia Hsu. Learning effects of location based mixed reality game: A pilot study. *Procedia Manufacturing*, 3:1603–1607, 2015.

[23] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[24] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *Computer Vision–ECCV 2014*, pages 834–849. Springer, 2014.

[25] Jonas Etzold, Michael Englert, Paul Grimm, Yvonne Jung, and Marcel Klomann. Mipos: towards mobile image positioning in mixed reality web applications based on mobile sensors. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, pages 17–25. ACM, 2014.

[26] Jonas Etzold, Michael Englert, Paul Grimm, Yvonne Jung, and Marcel Klomann. Mipos: Towards mobile image positioning in mixed reality web applications based on mobile sensors. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, Web3D '14, pages 17–25, New York, NY, USA, 2014. ACM.

[27] Alessandro Febretti, Arthur Nishimoto, Terrance Thigpen, Jonas Talandis, Lance Long, JD Pirtle, Tom Peterka, Alan Verlo, Maxine Brown, Dana Plepys, et al. Cave2: a hybrid reality environment for immersive simulation and information analysis. In *IS&T/SPIE Electronic Imaging*, pages 864903–864903. International Society for Optics and Photonics, 2013.

[28] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[29] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[30] Kiev Gama, Rafael Wanderley, Daniel Maranhao, and Vinicius Cardoso Garcia. A web-based platform for scavenger hunt games using the internet of things. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*, pages 835–840. IEEE, 2015.

[31] Joseph A. Gonzales, Casey Fiesler, and Amy Bruckman. Towards an appropriable cscw tool ecology: Lessons from the greatest international scavenger hunt the world has ever seen. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 946–957, New York, NY, USA, 2015. ACM.

[32] Google. Google glass. https://www.google.com/glass/start/, 2015. [Online; accessed 06/03/2016].

[33] Google. Google earth. https://www.google.com/earth/, 2016. [Online; accessed 08/03/2016].

[34] Google. Google maps. http://maps.google.com/, 2016. [Online; accessed 04/04/2016].

[35] Google. Project tango. https://www.google.com/atap/project-tango/, 2016. [Online; accessed 04/03/2016].

[36] Google. Street view - google maps. https://www.google.com/maps/streetview/, 2016. [Online; accessed 08/03/2016].

[37] Alastair Hampshire, Hartmut Seichter, Raphaël Grasset, and Mark Billinghurst. Augmented reality authoring: generic context from programmer to designer. In *Proceedings of the 18th Australia conference on Computer-Human Interaction: Design: Activities, Artefacts and Environments*, pages 409–412. ACM, 2006.

[38] M. Haringer and H. T. Regenbrecht. A pragmatic approach to augmented reality authoring. In *Mixed and Augmented Reality, 2002. ISMAR 2002. Proceedings. International Symposium on*, pages 237–245, 2002.

[39] Tobias Höllerer, Steven Feiner, Tachio Terauchi, Gus Rashid, and Drexel Hallaway. Exploring mars: Developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers and Graphics*, 23:779–785, 1999.

[40] HTC. Htc vive. http://www.htcvive.com/us/, 2016. [Online; accessed 08/03/2016].

[41] Simon Julier, Marco Lanzagorta, Yohan Baillot, Lawrence Rosenblum, Steven Feiner, Tobias Hollerer, and Sabrina Sestito. Information filtering for mobile augmented reality. In *Proceedings of the IEEE International Symposium on Augmented Reality*, pages 3–11, 2000.

[42] Junaio. Metaio developer portal. https://my.metaio.com/dev/junaio/, 2016. [Online; accessed 08/03/2016].

[43] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[44] Kudan. Kudan ar sdk. http://kudan.eu/, 2016. [Online; accessed 31/03/2016].

[45] Niantic Labs. Pokmon go. https://www.microsoft.com/microsoft-hololens/en-us, 2016. [Online; accessed 08/16/2016].

[46] Tobias Langlotz, Stefan Mooslechner, Stefanie Zollmann, Claus Degendorfer, Gerhard Reitmayr, and Dieter Schmalstieg. Sketching up the world: in situ authoring for mobile augmented reality. *Personal and ubiquitous computing*, 16(6):623–630, 2012.

[47] Layar. Augmented reality — interactive print. https://www.layar.com/, 2016. [Online; accessed 08/03/2016].

[48] Gun A Lee, Claudia Nelles, Mark Billinghurst, and Gerard Jounghyun Kim. Immersive authoring of tangible augmented reality applications. In *Proceedings of the 3rd IEEE/ACM international Symposium on Mixed and Augmented Reality*, pages 172–181. IEEE Computer Society, 2004.

[49] Mathieu Loiseau, Élise Lavoué, Jean-Charles Marty, and Sébastien George. A multiplayer learning game based on mixed reality to enhance awareness on archaeology. *EAI Endorsed Transactions on Serious Games*, 1(3):e3, 2014.

[50] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[51] Yan Lu, Joseph T Chao, and Kevin R Parker. Hunt: Scavenger hunt with augmented reality. *IJIKM*, 10, 2015.

[52] Wendy E Mackay. Augmented reality: linking real and virtual worlds: a new paradigm for interacting with computers. In *Proceedings of the working conference on Advanced visual interfaces*, pages 13–21. ACM, 1998.

[53] MapNav. Mapnav geolocation toolkit. http://recursivearts.com/mapnav/home.html, 2016. [Online; accessed 04/15/2016].

[54] Mechdyne. Mechdyne corporation: Advanced technology solutions. http://www.mechdyne.com/, 2016. [Online; accessed 08/03/2016].

[55] Metaio. Metaio: The augmented reality company. http://www.metaio.com/, 2016. [Online; accessed 06/03/2016].

[56] Microsoft. Bing maps. http://www.bing.com/mapspreview, 2016. [Online; accessed 08/03/2016].

[57] Microsoft. Hololens. https://www.microsoft.com/microsoft-hololens/en-us, 2016. [Online; accessed 04/03/2016].

[58] Microsoft. Kinect for xbox one. http://www.xbox.com/en-US/xbox-one/accessories/kinect-for-xbox-one, 2016. [Online; accessed 08/03/2016].

[59] Microsoft. Xbox one controllers. http://www.xbox.com/en-US/xbox-one/accessories/controllers, 2016. [Online; accessed 08/03/2016].

[60] Branislav Micusik and Jana Kosecka. Piecewise planar city 3d modeling from street view panoramic sequences. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2906–2912. IEEE, 2009.

[61] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.

[62] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598, 2002.

[63] Leap Motion. Leap motion. http://www.leapmotion.com, 2016. [Online; accessed 31/03/2016].

[64] Mozilla. Webgl apis. https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API, 2016. [Online; accessed 06/03/2016].

[65] WM Mularie. World geodetic system 1984–its definition and relationships with local geodetic systems. *Department of Defense, NIMA USA*, 2000.

[66] Oculus. Oculus rift. https://www.oculus.com/en-us/rift/, 2016. [Online; accessed 08/03/2016].

[67] OpenCV. Open source computer vision. http://opencv.org/, 2016. [Online; accessed 06/03/2016].

[68] Charles B Owen, Pm Xiao, and Paul Middlin. What is the best fiducial? In *Augmented Reality Toolkit, The First IEEE International Workshop*, pages 8–pp. IEEE, 2002.

[69] Oyewole Oyekoya, Ran Stone, William Steptoe, Laith Alkurdi, Stefan Klare, Angelika Peer, Tim Weyrich, Benjamin Cohen, Franco Tecchia, and Anthony Steed. Supporting interoperability and presence awareness in collaborative mixed reality environments. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, pages 165–174, New York, NY, USA, 2013. ACM.

[70] Wayne Piekarski and Bruce Thomas. Arquake: the outdoor augmented reality gaming system. *Communications of the ACM*, 45(1):36–38, 2002.

[71] Johanna Pirker, Christian Gutl, Patrick Weiner, Victor Manuel Garcia-Barrios, and Melanie Tomintz. Location-based mobile application creator creating educational mobile scavenger hunts. In *Interactive Mobile Communication Technologies and Learning (IMCL), 2014 International Conference on*, pages 160–164. IEEE, 2014.

[72] Cristina Portalés, José Luis Lerma, and Santiago Navarro. Augmented reality and photogrammetry: A synergy to visualize physical and virtual city environments. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(1):134–142, 2010.

[73] Ivan Poupyrev, Desney Tan, Mark Billinghurst, Hirokazu Kato, Holger Regenbrecht, and Nobuji Tetsutani. Tiles: A mixed reality authoring interface. In *INTERACT 2001 Conference on Human Computer Interaction*, pages 334–341, 2001.

[74] QUALCOMM. Wireless technology and innovation. https://www.qualcomm.com/, 2016. [Online; accessed 06/03/2016].

[75] Gerhard Reitmayr and Tom Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 109–118. IEEE Computer Society, 2006.

[76] Gerhard Reitmayr and Dieter Schmalstieg. *Collaborative augmented reality for outdoor navigation and information browsing.* na, 2004.

[77] Jun Rekimoto. Matrix: A realtime object identification and registration method for augmented reality. In *Computer Human Interaction, 1998. Proceedings. 3rd Asia Pacific*, pages 63–68. IEEE, 1998.

[78] Katja Rogers, Julian Frommel, Larissa Breier, Sinan Celik, Harry Kramer, Stefan Kreidel, Julia Brich, Valentin Riemer, and Claudia Schrader. Mobile augmented reality as an orientation aid: A scavenger hunt prototype. In *Intelligent Environments (IE), 2015 International Conference on*, pages 172–175. IEEE, 2015.

[79] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.

[80] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[81] Sony. Playstation move. https://www.playstation.com/en-us/explore/accessories/playstation-move/, 2016. [Online; accessed 25/03/2016].

[82] Sony. Playstation vr. https://www.playstation.com/it-it/explore/playstation-vr/, 2016. [Online; accessed 25/03/2016].

[83] R Stouffs, P Janssen, S Roudavski, and B Tunçer. What is happening to virtual and augmented reality applied to architecture? In

*Conference on Computer-Aided Architectural Design Research in Asia (CAADRIA 2013)*, volume 1, page 10, 2013.

[84] Ivan E Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 757–764. ACM, 1968.

[85] Gabriel Takacs, Vijay Chandrasekhar, Natasha Gelfand, Yingen Xiong, Wei-Chao Chen, Thanos Bismpigiannis, Radek Grzeszczuk, Kari Pulli, and Bernd Girod. Outdoors augmented reality on mobile phone using loxel-based visual feature organization. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*, pages 427–434. ACM, 2008.

[86] Jerry O. Talton, Daniel L. Peterson, Sam Kamin, Deborah Israel, and Jalal Al-Muhtadi. Scavenger hunt: Computer science retention through orientation. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '06, pages 443–447, New York, NY, USA, 2006. ACM.

[87] Hideyuki Tamura, Hiroyuki Yamamoto, and Akihiro Katayama. Mixed reality: Future dreams seen at the border between real and virtual worlds. *Computer Graphics and Applications, IEEE*, 21(6):64–70, 2001.

[88] Stefaan Ternier, Roland Klemke, Marco Kalz, Patricia van Ulzen, and Marcus Specht. Arlearn: Augmented reality meets augmented virtuality. 18(15):2143–2164, aug 2012. http://www.jucs.org/jucs_18_15/ARLearn_augmented_reality_meets.

[89] Three.js. Javascript 3d library. http://threejs.org/, 2016. [Online; accessed 06/03/2016].

[90] Christian Tonn, Frank Petzold, Oliver Bimber, Anselm Grundhöfer, and Dirk Donath. Spatial augmented reality for architecture–designing and planning with and within existing buildings. *International Journal of Architectural Computing*, 6(1):41–58, 2008.

[91] UIC-EVL. Omegalib. https://github.com/uic-evl/omegalib, 2016. [Online; accessed 08/03/2016].

[92] Unity. Game engine, tools and multiplatform. https://unity3d.com/unity, 2016. [Online; accessed 06/03/2016].

[93] Unity. Networking overview. http://docs.unity3d.com/Manual/UNetOverview.html, 2016. [Online; accessed 26/03/2016].

[94] Unity. Unity documentation: Remote actions. http://docs.unity3d.com/Manual/UNetActions.html, 2016. [Online; accessed 04/03/2016].

[95] Vuforia. Natural features and image ratings. https://developer.vuforia.com/library/articles/Solution/Natural-Features-and-Ratings, 2016. [Online; accessed 06/03/2016].

[96] Vuforia. Vuforia. http://vuforia.com/, 2016. [Online; accessed 06/03/2016].

[97] Xiangyu Wang. Augmented reality in architecture and design: potentials and challenges for application. *International Journal of Architectural Computing*, 7(2):309–326, 2009.

[98] Richard Wetzel, Lisa Blum, and Leif Oppermann. Tidy city: a location-based game supported by in-situ and web-based authoring tools to enable user-created content. In *Proceedings of the international conference on the foundations of digital games*, pages 238–241. ACM, 2012.

[99] Richard Wetzel, Lisa Blum, and Leif Oppermann. Tidy city: A location-based game supported by in-situ and web-based authoring tools to enable user-created content. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, pages 238–241, New York, NY, USA, 2012. ACM.

[100] Wikitude. Wikitude - the world's leading augmented reality sdk. http://www.wikitude.com/, 2016. [Online; accessed 08/03/2016].

[101] Yoonsik Yang, Jiwook Shim, Seungho Chae, and Tack-Don Han. Mobile augmented reality authoring tool. In *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*, pages 358–361. IEEE, 2016.

[102] Neil Zhao. Full-featured pedometer design realized with 3-axis digital accelerometer. *Analog Dialogue*, 44(06), 2010.

[103] Fabio Zünd, Mattia Ryffel, Stéphane Magnenat, Alessia Marra, Maurizio Nitti, Mubbasir Kapadia, Gioacchino Noris, Kenny Mitchell,

Markus Gross, and Robert W Sumner. Augmented creativity: bridging the real and virtual worlds to enhance creative play. In *SIGGRAPH ASIA 2015 Mobile Graphics and Interactive Applications*, page 21. ACM, 2015.

[104] Fabio Zünd, Mattia Ryffel, Stéphane Magnenat, Alessia Marra, Maurizio Nitti, Mubbasir Kapadia, Gioacchino Noris, Kenny Mitchell, Markus Gross, and Robert W. Sumner. Augmented creativity: Bridging the real and virtual worlds to enhance creative play. In *SIGGRAPH Asia 2015 Mobile Graphics and Interactive Applications*, SA '15, pages 21:1–21:7, New York, NY, USA, 2015. ACM.