Graz University of Technology

Institute for Computer Graphics and Vision

Dissertation

Handheld Augmented Reality

Daniel Wagner

Graz, Austria, October 1st, 2007

Thesis Supervisor
Prof. Dr. Dieter Schmalstieg

Referees
Prof. Dr. Blair MacIntyre
Prof. Dr. Mark Billinghurst

TO MICHAEL

Abstract

Augmented Reality (AR) aims at developing new user interfaces. Although research has produced a large number of application prototypes and AR frameworks in the last 20 years, no project has yet been practical enough to create a mass market success.

There are many reasons for this. Traditionally, AR researchers have primarily created prototypes that aim to solve engineering problems such as maintenance or new interfaces for complex environments such as head up displays for navigation and battlefield systems. Most researchers still see AR as a basic research area. Developing easy to use, practical applications, such as for home users, is therefore usually not a goal. Another problem with many Augmented Reality systems is the highly complex hardware setup, often including expensive commercial sensors, input devices and output devices. These devices are often bulky and fragile, since they were never meant to be operated by untrained users.

Research at the Vienna University of Technology and the Graz University of Technology has aimed at moving Augmented Reality to a mass-market. Instead of specialized and expensive hardware, this project targets low cost mobile devices, namely mobile phones. In contrast to traditional AR hardware, people already own these devices and know how to operate them. Recently, processing capabilities of mobile phones have reached a level that makes these devices capable of running standalone AR applications and renders them ideal candidates for mass marketed Augmented Reality solutions.

This thesis presents a framework that for the first time allows for the creation of practical AR applications on end user-owned devices. The software runs on a broad range of devices and has been used for several–some even commercial–applications. To prove the applicability of the new platform the author of this thesis has performed evaluations with untrained users in real-life environments such as museum exhibitions or conferences.

Acknowledgements

I'd like to thank everybody without whom this dissertation would not have been possible: First and foremost, I would like to thank my supervisor Prof. Dieter Schmalstieg for given me the opportunity of joining his research group and working in the area of Augmented Reality. During my time with him at the Vienna and Graz Universities of Technology I learned a lot about research, leadership and team work. I want to thank him especially for all inspirational discussions and giving me the freedom to follow my own ideas.

Although this thesis was finished in Graz it started at the Vienna University of Technology. I want to thank all staff of TU-Wien for the very well designed courses on computer science. I am grateful to the members of the Interactive Media Systems Group (IMS), especially to Istvan Barakonyi and Joseph Newman with whom I shared an office for two years and who helped me to improve my English language. I'd also like to thanks Gerhard Reitmayr and Hannes Kaufmann for introducing me into the many complex topics of AR software development. A big thanks also goes to Thomas Pintaric who co-developed most of my projects at TU-Wien.

I want to thank the Institute for Computer Graphics and Vision (ICG) for letting me join the Graz University of Technology. I am much obliged to Michael Kalkusch who shares an office with me for three years now and for the many fruitful discussions on our car rides between Vienna and Graz. Special acknowledgements goes to Albert Walzer, who developed many ideas for my games, created most of the content and shared a flat with me for two years in Graz.

I appreciate the cooperation with Imagination GmbH, especially with Matthias Stifter and Christian Traxler. I am grateful to Michael Gervautz who among the first to believe in the commercial success of handheld Augmented Reality.

I am grateful to my parents who always believed in my scientific career and who supported me in my studies. I am much obliged to my wife Valerie who tolerated my long working hours and always supported me in my plans.

Contents

Chapter 1

Introduction

1.1 Augmented Reality

Many systems today are too difficult to use because of complex user interfaces. This is partially due to a lack of competence in designing user interfaces many engineers suffer from. A more important reason is that with the growing computational power of modern systems, devices and applications become more complex and integrate more features. Soft- and hardware that was only available to a small amount of specialists a few decades ago, is now a well-integrated part of everyone's daily life. Good user interface design is therefore no longer an option but a hard requirement for developing highly usable applications.

Augmented Reality (AR) research aims at developing new human computer interfaces. Instead of showing information on isolated displays, it puts data right where it belongs: into the real world. AR thereby blurs the distinction between the real world and the user interface and combines them in a natural way allowing the creation of simple and intuitive user interfaces even for complex applications.

Until today there is no clear definition of AR. Although first AR-like systems were developed in the 1960s, Augmented Reality only separated itself from virtual reality and became a research area in its own rights in the beginning of the 1990s. Today two main definitions exist that describe Augmented Reality. Due to a lack of an official agreement on the term, both are accepted. Following the definition of Azuma [3] an AR system has to fulfill the three requirements:

- Combine the real and virtual
- Registered in the real world in 3D

- Interactive in real time

The first requirement is a fundamental description of AR in that it combines the real world with virtual content. The second requirement separates Augmented Reality from the more general concepts of mixed reality or mixed media by requiring that the virtual content must be registered in 3D within the real world. Finally “Interactive in real time” requires the system to react to the user and update in real time which distinguishes AR from all off-line augmentations such as the use of computer graphics in movies.

According to the older Virtuality continuum proposed by Milgram [69] (see Figure 1.1), AR is just one possible manifestation of Mixed Reality (MR), which brings together real and virtual within a single display. The Virtuality continuum juxtaposes AR and Augmented Virtuality (AV). AR is mostly grounded in the real world, with a limited set of virtual objects mixed in. The inverse concept, AV, is conceived as a Virtual Environment with some real aspects - a recurring example for AV are video-textured avatars (showing a live video feed of real people) within a Virtual Environment. The boundary between AR and AV is not strictly defined.

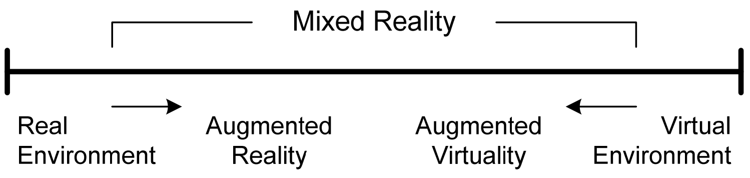


Figure 1.1: Milgram's reality-virtuality continuum.

Augmented reality (AR) is a natural complement to mobile computing research, since a mobile AR system can assist the user directly at the workplace instead of requiring the user to attend to stationary workstations. There has been a lot of work in creating mobile AR setups using mobile personal computer hardware such as notebooks. The advantage of those approaches is that hard- and software very similar to traditional non-mobile AR systems can be used. While there are many working systems composed of a notebook and a head mounted display (HMD), most of these setups have been designed as mere proof-of-concept and do not provide a usable form factor. Usually wearable prototypes have all their hardware mounted to a large backpack, including heavy power supplies for items not designed for mobile use. While such backpack/HMD combinations unite superior performance with hands-free operation, they severely affect dexterity, prevent practical use and are socially unacceptable. They are maintenance intensive and lack robustness due to their complex hardware setups. Most of the devices used were not designed for mobile deployment and therefore not only require heavy batteries but also use fragile connectors and cables. Furthermore, the prohibitive cost of these setups prevents deploying them in a commercial market. In addition, the development of HMD technology, which is an indispensable part of such an approach to wearable AR, is not keeping pace with the advances in computer and sensor technology.

At the same time, broad consumer interest in very small form factor devices and displays, such as cell phones and handheld computers, is dramatically accelerating the development in this area. We therefore consider it to be obvious that one of the next major steps in mobile AR development will be a shift to smaller and more ergonomic devices.

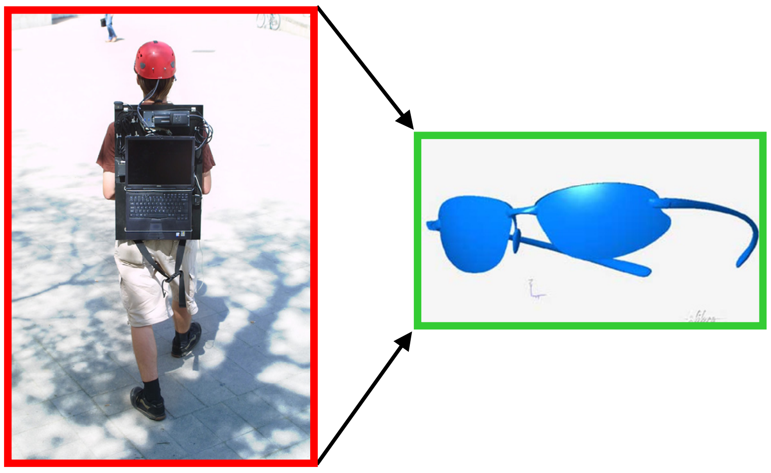


Figure 1.2 Ergonomic considerations for Augmented Reality

Ideally an AR setup would look like the right image in Figure 1.2: instead of carrying specialized, complex and expensive hardware a single pair of glasses would be sufficient for video as well as audio augmentations. Of course such a system will not be feasible for at least a decade. Instead the work in this thesis concentrated on mobile devices that are available today: mobile phones and PDAs, which together with Tablet PCs and the so called Ultra Mobile PCs (UMPCs) form the hardware basis for handheld Augmented Reality.

1.2 Handheld Augmented Reality

Personal digital assistants (PDAs), game handhelds and smartphones have fueled the interest in mobile gaming from both a commercial and a scientific perspective. While there is a lot of previous research on pervasive computing, little work has been done on mobile multi-user Augmented Reality applications.

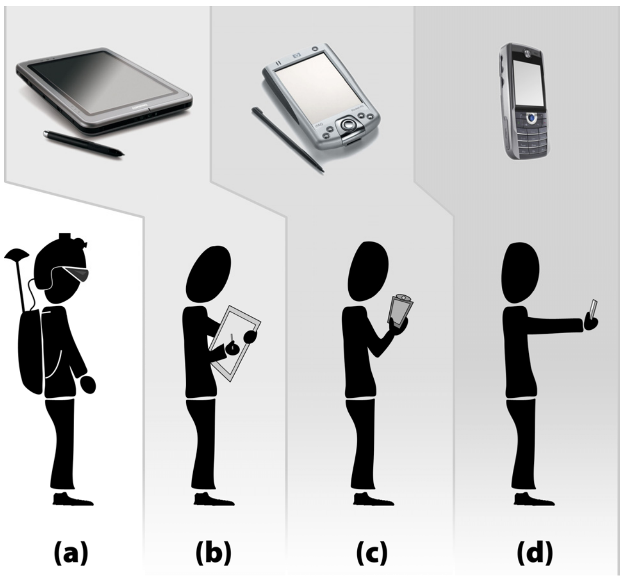


Figure 1.3: Form factors of Mobile Augmented Reality systems:
(a) traditional "backpack" computer & HMD, (b) Tablet PC, (c) PDA, (d) Mobile phone

In this thesis we define handheld AR as a setup, where the user holds the mobile device actively in his hand (b, c and d in Figure 1.3): Handheld AR is different from AR using wearable computers with HMDs where users have both hands free (a in Figure 1.3). A phone-based AR setups allows the user to use her phone as an AR interaction device, although some of the processing or application intelligence might not be implemented on the phone itself.

Each of the device classes in Figure 1.3 above has unique characteristics: Backpack setups provide most processing power. Furthermore their HMDs give the highest immersion among all four device classes. The digitizer inputs of Tablet PCs make it possible to create highly accurate, yet easy to use user interfaces. Their processing power is similar to backpacks although these devices have to make compromises due to weight and size restrictions.

PDAs have recently merged with mobile phones. Classic PDAs without phone capabilities hardly exist anymore as today most PDAs include networking capabilities too. They are typically more powerful than mobile phones, have larger displays and touch screens. Mobile phones are the smallest and most ubiquitous device class. In contrast to the previously mentioned device classes they are today a fully integrated part of most people's daily life. Most mobile phones have similar screen resolutions as

PDAs, but their displays are smaller and most often they are not equipped with touch screen input. Naturally they possess the smallest amount of processing power of all device classes.

Besides ergonomic factors and processing capabilities, availability of ready-to-use, commercial systems in large numbers is a major factor for practical projects and setups. While mobile phones and PDAs usually come as a fully AR-capable package, many Tablet PCs require an external camera for video-see through Augmented Reality. While this can be easily achieved with minimal technical knowledge, the implementation of a back-pack setup is a difficult and non-trivial engineering task. The only company known to produce commercial back-pack systems is a_rage[1], a university spin-off. Their current status and commercial success is unknown.

Another decisive factor for practical setups is the cost per client device. Tablet PCs typically start at ~1000€. The combined hardware costs of backpack setups though, including a notebook, HMD and tracking solution usually sums up to several 1000€. A commercial system can therefore be expected to cost between 5000€ and 10000€, creating severe problems for mass user deployment. PDAs and mobile phones typically cost between 300€ and 600€. Furthermore, most people today already own mobile phones, which makes targeting the end users' personal devices a major goal for reducing costs. While only a small number of these phones are today capable of running AR applications, their number is constantly increasing.

1.3 Hypothesis

This thesis discusses the suitability of mobile phone based Augmented Reality. It therefore poses the following hypothesis statements that are examined throughout the remainder of this document:

H1 Augmented Reality on phones can work as well as on personal computers, despite the fact that phones are less powerful, have smaller screens and inferior input capabilities.

H2 Using phone based AR, larger mobile Augmented Reality systems than previously shown can be built at reasonable costs.

H3 The phone's form factor is more suitable for untrained users than HMD-based setups.

[1] http://www.a_rage.com

The statements above are phrased vaguely on purpose and require a more detailed explanation in order to not be misunderstood:

This thesis defines AR on phones as using off-the-shelf mobile phones for AR input and output devices. This does not prevent developing systems with backend servers for outsourcing tasks, as long as this backend is not perceived as part of the user interface. Naturally phones possess inherent weaknesses compared to PC-based systems such as inferior processing power. A major part of the work done in the course of this thesis therefore concentrated on developing techniques that balance the available resources.

By "size of a system", this thesis defines the number of parts and people interacting or collaborating. A larger system therefore integrates more interactions, applications, devices or users. As will be shown in the remainder of this thesis, the advantage of low cost in client devices is a major advantage for phone based AR – especially compared to backpack setups. The development of such large systems requires a toolkit that is easy to use yet capable to support complex requirements of large setups, or as Einstein said: "Everything should be made as simple as possible, but not one bit simpler."

HMD-based setups clearly have the advantage of (potentially) higher immersion than handheld mobile AR systems. Yet, they are not well accepted by end users for several reasons including cabling, fragility, ergonomic factors and motion sickness. This thesis does not present a formal study comparing HMD-based setups versus TabletPC and phone based alternatives. Still, our own experiences over many years with all three variants clearly show that handheld solutions are most often preferred over HMD-based ones.

The positive reception of handheld AR by end users throughout the many trials conducted during this thesis work, confirms a baseline of wide acceptance of handheld AR.

1.4 Contribution

The contribution of this thesis is the design and development of a fully working Augmented Reality framework that works on end-users' devices and was tested in multiple practical applications outside research labs. It is empirical proof of the hypothesis formulated above. The system is highly novel in being the first and currently only implementation of a complete handheld AR platform. This includes in particular:

- A **fiducial marker tracking library** that is suitable for mobile phones and PDAs.
- A **scene-graph based render engine** for AR applications, specifically targeting the restrictions of mobile phones.
- A **communication framework** that targets the specific requirements of distributed applications with spontaneous connectivity
- A flexible **Augmented Reality framework** the combines the back building blocks listed above into an easy to use prototyping toolkit.
- **Various applications** that explore the possibilities and weaknesses of AR on mobile phones and similar devices.

1.4.1 Marker Tracking System for Phones

Pose tracking is an integral part of every AR and VR application. The user's or device's pose must be measured accurately, robustly and in real-time. While there are many commercial tracking systems available that perfectly fulfill these requirements, these solutions typically target stationary setups. For mobile setups tethered and stationary systems are not suitable, which rules out an outside-in tracking approach that is the basis for most commercial tracking systems.

Targeting a lost-cost solution that should run on unmodified mobile phones, narrows down the available options considerably. Most mobile phones today are equipped with built-in cameras, which naturally lends itself to using computer vision based approaches for tracking. Tracking fiducial markers is a common strategy to achieve robustness and computational efficiency simultaneously.

The ARToolKitPlus library developed in this thesis is based on the well known open source ARToolKit library. It was ported to the Windows CE environment, optimized for the mobile phone platform and heavily extended with features that support mobility, such as automatic thresholding or large amounts of markers. The very latest version of ARToolKitPlus (also known as Studierstube Tracker) is a complete redevelopment that is no longer related to ARToolKit on a source code basis and improves several outstanding issues such as memory consumption, tracking quality or numerical stability.

A well working tracking system is the basic foundation of every AR setup. Hence, ARToolKitPlus was welcomed by the AR community and is used today in many AR applications. The results of the work on ARToolKitPlus have been reported in [113]. Details on the developed solutions are given in chapter 3 of this thesis.

1.4.2 A scene-graph based render toolkit for mobile phones

Next to tracking, rendering is the second most important aspect of every AR system. Although Augmented Reality does not define rendering as a hard requirement, most systems concentrate on graphical augmentations, rather than other feedback such as audio, tactical or olfactory.

While professional solutions for rendering on mobile phones exist, these libraries are usually expensive, closed source and not sufficiently general, which makes them hard to extend and to use on commercial projects.

Studierstube Scene-Graph (StbSG) developed in the course of this thesis was created to provide a similar flexibility such as well known scene-graph libraries on desktop computers. Yet, care was taken to respect the specific restrictions of mobile phones including low memory footprint, fast operation and support for native renderers such as OpenGL ES and Direct3D. StbSG has been reported in [92]. Details are given in chapter 6 of this thesis.

1.4.3 Communication framework for mobile AR

While many AR systems today are single-user setups, the full power of phone-based AR can only be unleashed in multi-user applications that allow a large numbers of users to share the virtual space on their phones introduces them to. There has been a large amount of research for distributed VR systems, but only little for AR applications. Existing research into this direction commonly assumes a stable connection with high powered servers as well as clients.

For truly mobile and pervasive setups though, connectivity loss is not an exception but rather a rule. An ideal multi-user system would run stand-alone when no networking is available and instantly make use of spontaneous network connectivity. Hence, applications as well as the underlying communication framework must target these circumstances in their design.

The Muddleware communication framework developed in this thesis specifically targets these requirements. It is built around a server-hosted XML database that can be addressed via XPath queries. The client software is lightweight and allows accessing the shared database either explicitly via query commands or by using a shared-memory mechanism. Its scripting capabilities support the development of data-driven applications.

Muddleware is used in several internal projects at the Graz University of Technology as well as by other researchers. Details on Muddleware have been reported in [114] and are presented in chapter 5 of this thesis.

1.4.4 Studierstube ES Framework

The contributions listed above build the basis for a practical, mobile AR system. Yet, creating a real application requires tremendous efforts when directly combining these basic building blocks. Studierstube ES (StbES) is a framework that integrates these components into an easy to use solution than can be adapted to many client devices. Details on StbES have been reported in [92] and are given in chapter 6.

1.4.5 Evaluations in practical Applications

The Handheld AR system developed in this thesis allows for the first time to build large, practical AR systems on phones. In the course of this thesis several applications were developed and evaluated. The results of these evaluations and the experiences gained are important outcomes for other researchers, who aim to build similar setups.

Early Handheld AR applications reported in this thesis such as “Kanji Teaching”, “Invisible Train”, the “Virtuoso” game and the “MARQ game at Technisches Museum Wien” were developed using predecessors of StbES. While these applications were based on PDAs with attached cameras, later applications such as “Expedition Schatzsuche” or “Signpost 2007” were deployed on unmodified and therefore more robust devices.

The applications developed, experiences gained and evaluations performed have been reported in [109], [110], [111], [115] and [116]. Details are given in chapter 7.

1.5 Results

This document describes the development and architecture of the handheld AR system that was developed as part of this thesis. It mainly concentrates on presenting Studierstube ES, which is the latest version of the handheld AR project and the third generation of this software.

Over the last four years the author has continuously improved the software and benefited from the increasing capabilities of newly introduced mobile phones and PDAs at the same time. There is reason to believe that handheld AR is now suitable for real life usage by regular end users (customers, home users) which is not equally true for other variants of AR setups. The work presented in this thesis is based on the assumption that handheld AR is suitable as a mass-market interface if it fulfills the following requirements:

- **Low cost:** A practical solution to be used at anytime and anyplace must be low cost and therefore run on off-the-shelf commercial devices.
- **Robust and fool-proof:** To be usable by untrained users in unsupervised situations, AR system must be robust and fool-proof. This requires soft- and hardware that was specifically designed to be used by non-experts. It also requires the creation of intuitive user interfaces.
- **Self contained operation and networking support:** Support for collaboration is fundamental to unleash the full potential of AR applications, which requires networking. On the other hand users expect their devices to run at any time and any place which demands self contained operation. A successful system must therefore be able to run standalone as well as take benefit of networking capabilities.
- **Tracking support:** Real-time tracking is probably the most fundamental requirement to any AR system. While many AR research approaches employ high-quality, commercial tracking solutions, a system for the masses must rely on simpler solutions that allow taking advantage of built-in device capabilities.
- **Rapid prototyping:** Although solutions developed in this thesis target a mass market of end users there is still a lot of movement in the actual type and design of applications and user interfaces. It is therefore important to support the fast creation of new applications to evaluate new concepts.
- **Content creation:** After a first "wow-effect" wears off, users demand practical benefits which, in the case of AR, require a strong content creation pipeline. Other than for pure research it is not enough to get data "somehow" into the AR application. Instead a clean chain of tools that relies on industry standards is required.

Building upon the lessons learned in this thesis, researchers can more easily evaluate Augmented Reality concepts and applications scenarios in realistic environments. Until now, applications only included a small number of users. No other mobile setups with more than 3 users have been reported so far. Finally, our system allows creating and deploying applications evaluating the following scenarios with real users in real environments:

- **Massive Multi User AR:** Augmented Reality can create shared virtual spaces that are collocated with the real world. The full potential of these concepts can only be exploited with many collaborative or competitive users. Until today the high costs of typical AR systems prevented the introduction of AR “to the masses”.
- **Evaluating AR applications with real users:** So far applications have typically been evaluated with test users in test environments (research labs). Bringing AR to the people and making it run in their devices allows evaluating usage in natural environments and conditions.
- **High quality content:** Content creation is an expensive business and only pays off if the market is large enough to support it. The number of deployed AR capable mobile phones is predicted to reach one billion by 2012[2] making it larger than any console or PC gaming market. The prediction provides a strong backup for continuing work on handheld Augmented Reality.

1.6 Collaboration statement

This thesis builds upon work done in collaboration with other researchers. The following list of publications gives an overview of the people involved in the handheld AR project:

- Schmalstieg, D., Wagner, D., Experiences with Handheld Augmented Reality, The Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR’07), 2007, Japan
  This paper presents the MARQ Museum Augemented Reality Quest application and the Studierstube ES system it builds upon.
- Wagner, D., Schmalstieg, D., First Steps Towards Handheld Augmented Reality. Proceedings of the 7th International Conference on Wearable Computers (ISWC 2003), pp. 127-135, 2003, USA
  This paper represents the first publication in the course of thesis and describes the Signpost 2003 application.
- Wagner, D., Schmalstieg, D., ARToolKit on the PocketPC Platform, The Second IEEE International Augmented Reality Toolkit Workshop, 2003, Japan

[2] Canalys report from Nov. 2006, http://www.canalys.com

This publication discusses the predecessor of ARToolKitPlus and hence the first real-time 6DOF tracking solution running natively on off-the-shelf PDAs.

- Wagner, D., Barakonyi, I., Augmented Reality Kanji Learning, Proceedings of the 2nd IEEE/ACM Symposium on Mixed and Augmented Reality (ISMAR 2003), 2003, Japan
  This publication presents the AR Kanji Teaching application, a simple memory game that trains the player in reading Kanji symbols.
- Wagner, D., Schmalstieg, D., A Handheld Augmented Reality Museum Guide, Proceedings of IADIS International Conference on Mobile Learning 2005 (ML2005), 2005, Malta
  The publication gives an overview of the handheld AR project and its work in the area of AR in museums.
- Wagner, D., Pintaric, T., Ledermann, F., Schmalstieg, D., Towards Massively Multi-User Augmented Reality on Handheld Devices, Proceedings of Third International Conference on Pervasive Computing (Pervasive 2005), 2005, Germany
  The paper presents the Invisible Train application and draws conclusions for the suitability of handheld AR for the masses.
- Wagner, D., Billinghurst, M., Schmalstieg, D., How Real Should Virtual Characters Be?, Conference on Advances in Computer Entertainment Technology (ACE 2006), 2006, USA
  The paper written during the internship of the first author at the HITLabNZ, studies the influence of virtual characters in Augmented Reality.
- Wagner, D., Schmalstieg, D., Handheld Augmented Reality Displays, Proceedings of IEEE Virtual Reality (VR2006), 2006, USA
  This position statement publication gives a broad overview on the latest developments in the handheld AR project.
- Wagner, D., Schmalstieg, D., Billinghurst, M., Handheld AR for Collaborative Edutainment, Proceedings of 16th International Conference on Artificial Reality and Telexistence (ICAT), 2006, China
  This paper presents a user study that compares the paper, desktop and handheld AR variants of a multi-player edutainment game.
- Wagner, D., Schmalstieg, ARToolKitPlus for Pose Tracking on Mobile Devices, Proceedings of 12th Computer Vision Winter Workshop (CVWW'07), 2007, Austria
  The paper gives a state of the art report on 6DOF tracking on mobile phones.
- Wagner, D., Schmalstieg, D., Muddleware for Prototyping Mixed Reality Multiuser Games, Proceedings of IEEE Virtual Reality 2007 (VR2007), 2007,

USA
The paper presents Muddleware, the networking middleware developed during this thesis.

The following people deserve specific mentioning, since a considerable part of this thesis would not have been possible without them:

- **Istvan Barakonyi** of Graz University of Technology co-developed the Signpost2007 application and animated the Virtuoso virtual character. He also co-developed the idea of learning Kanjis using Augmented Reality and helped in translating symbols and finding correct 3D models for the Kanji teaching application.
- **Michael Kalkusch** of Graz University of Technology and Thomas Psik of Vienna University of Technology wrote the Java client implementation of Muddleware.
- **Hirokazu Kato** of NAIST, Japan and **Mark Billinghurst** of HITlab New Zealand developed the original ARToolKit library that ARToolKitPlus builds upon. Mark Billinghurst also co-designed the user study on virtual characters and developed most of the conclusions from the evaluation results.
- **Nina Mayer** of Landesmuseum Kärnten researched and designed all the hotspots for the MARQ application. Together with **Uwe Neuhold** of Verdandi, Nina Mayer moderated the evaluation at the Landesmuseum Kärnten.
- **Thomas Pintaric** co-developed the Invisible Train applications, built most of the props as well as organized and joined most of the demonstrations that led into the evaluation results. He's also the author of the DSVL Directshow Wrapper library that is used for development on the PC.
- **Christian Pirchheim** of Graz University of Technology developed the graphical user interface creation tool "Thekla" that builds on top of Muddleware.
- **Matthias Stifter** of Imagination GmbH developed most of the MARQ content and extended the first prototype into its final form.
- **Albert Walzer** of Graz University of Technology created graphics and videos for most of the applications and was an invaluable help for designing game concepts.

The remainder of this thesis discusses how the requirements stated in chapter 1.2 were met: After related work and state of the art is evaluated in chapter 2, chapter 3 presents the solution developed for tracking, while chapter 4 describes the rendering toolkit. Chapter 5 discusses the requirements on networking for handheld AR applications and describes the “Muddleware” networking middleware developed in this thesis. Chapter 6 gives details about the handheld AR framework Studierstube ES. It also presents Sphinx, an engine for creating multi-player AR treasure hunt games. Chapter 7 reports on applications developed and studies performed during this thesis. In chapter 8 conclusions are drawn and guidelines are given for development of handheld AR systems. The chapter finishes with an outlook into future research directions. The appendix in chapter 9 presents source code examples of simple AR applications developed with Studierstube ES. It also contains all questionnaires of the studies performed.

Chapter 2

Background and Related Work

This chapter reports on existing work in all areas that this thesis touches. Since the thesis reports on an extensive system design and draws inspiration from many areas, only the most relevant aspects are examined here. After looking at general aspects of previous work in Augmented Reality in section 2.1, this chapter discusses relevant work on 3D user interfaces in section 2.2., followed by the related issue of content creation in section 2.3. More technical aspects are examined in the rest of the chapter: Section 2.4 discusses pose tracking solutions for embedded devices, section 2.5 examined multiuser technology for AR and section 2.6 provides background on wireless connectivity technology, which is relevant for this work.

2.1 Augmented Reality

In 1968 Ivan Sutherland created the first head-mounted display (HMD) [101][102]. Due to limited processing power, his application showed only a simple wireframe model overlaid onto the real world. Yet, it marks the first application that fulfills the definition by Azuma and Milgram (see above).

The first Augmented Reality applications evolved from basic research, used enormously expensive hardware and consequently mostly covered research and technical problems only. In his 1995 survey paper Azuma lists six categories for AR applications: medical, manufacture and repair, annotation and visualization, robot path planning, military aircraft and entertainment. Some seminal works in these areas are given in the following.

Researchers at UNC Chapel Hill conducted first trials of overlaying 3D representations of ultra-sound data onto patients [5]. In the “Knowledge-based

Augmented Reality for Maintenance Assistance" (KARMA) project Feiner et al. created a laser printer maintenance application [24]. Milgram developed the ARGUS system [70] to create an easier way for robot path planning.

With the introduction of powerful portable computers and notebooks, mobile AR setups became possible. The Touring Machine [31][30] was among the first to use this new hardware platform for mobile systems. A later project of the same research group was MARS (Mobile Augmented Reality Systems) [49]. Presented in 1999, it was one of the first truly mobile augmented reality setups, which allowed the user to freely walk around while having all necessary equipment mounted onto his back. Several similar platforms such as BARS [21], Tinmith [77] and Studierstube [56] examined various application areas.

Due to the recent availability of Tablet PCs and UMPCs many researchers use these devices to bring existing software to smaller devices. Newman et al. use these mobile devices for experiments on wide area tracking [75]. Reitinger uses UMPCs to gather data in an urban environment [81]. After starting with backpack setups the iPERG project [62] then switched to UMPCs and Tablet PCs due to their lower costs and hardware maintenance requirements. The AMIRE[3] project used Tablet PCs for a museum guide.

2.1.1 Augmented Reality on Handheld and Embedded Devices

Even before the success of the smartphones as mass-marketed items, pioneering projects started using small displays for custom see-through devices. Many early works at least partially outsourced processing tasks to a nearby server via tethered or wireless networking.

As can be seen in Figure 2.1 there are four different levels of outsourcing processing tasks to a server: In the ideal case Figure 2.1(a), all work is performed natively by the client making it independent of the server and infrastructure. At the other extreme, many early handheld AR applications were based on a thin client approach with a "video-in/video-out" communication mechanism for receiving assistance from a computing server, which is shown as Figure 2.1(d). Such a setup does not only require a frame-by-frame communication but also requires sending video images in both directions requiring maximum performance of the network connection.

[3] http://www.amire.net/

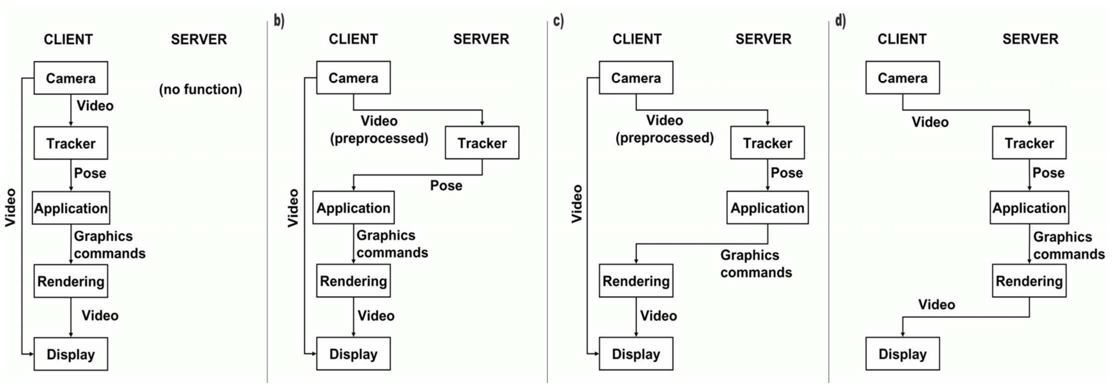


**Figure 2.1: Different level of outsourcing to a server:
a) All tasks are run natively by the client, b) Server performs tracking,
c) Server performs tracking and application logic, d) All work is done by the server**

However, these two solutions are just extreme examples of how work may be allocated among a server and handheld client. Depending on circumstances, solutions in between these extremes may be useful and necessary.

If one limits the discussion to a typical AR system which uses a single video source for both tracking and video see-through display, the processing pipeline is composed of the following main tasks: video acquisition, tracking, application computation, rendering, display. Offloading some of these tasks to a computing server is an instance of horizontally distributed simulation [65], and it is established knowledge that a scalable solution (many clients, many servers etc.) requires cautious use of the available network bandwidth [118]. Communication of raw video streams in both directions (Figure 2.1c) does not satisfy such bandwidth constraints. A better alternative seems to be streaming graphics commands back to the client such as done in the Chromium [52] framework.

The approach depicted in Figure 2.1(b) offloads the tracking task to a computing server, which requires upstream communication of pre-processed, compressed video for visual tracking purposes, followed by downstream communication of pose information. The advantage of this approach is that a very concise, but generic and computationally expensive task is offloaded to the server, while all application details are handled exclusively by the client, thus dependencies between client and server are minimal. For example, while tracking of artificial fiducials can be performed in real-time on embedded clients now, natural feature tracking can benefit from the greater computational power of a server for at least several more years.

Amselem's work [1] and Fitzmaurice's Chameleon [26] used small tethered LCD displays for location based information. Rekimoto's NaviCam [85] used color-coded stickers to track objects in the environment. Due to the tethered trackers in these early works, the degree of mobility was rather limited. mPARD [80] is a variant using analogue wireless video transmission to replace tethers. The Transvision [83] project by Sony CSL introduced handheld AR devices for a shared space. Researchers at HITLab later improved this concept [71] with a better user interface and an optical tracking solution re-using the camera needed for video see-through. All these works use simple tethered displays and cameras for the mobile device and are therefore extreme examples of Figure 2.1(d).

From 2000 on, PDAs with wireless networking were considered suitable for thin-client solutions outsourcing computationally intensive tasks such as rendering, tracking and application to a nearby workstation. The Batportal [54] used non-mixed 3D graphics using VNC, while the AR-PDA project [35] used digital image streaming from and to an application server. Both projects again use the method describe in Figure 2.1(d). Shibata's work [95] aims at load balancing between client and server - the weaker the client, the more tasks are outsourced to a server. It can therefore vary between all situations described in Figure 2.1. ULTRA uses PDA-based AR to support maintenance workers, but concentrates on augmenting "snapshot" still images [68]. In the absence of real-time tracking for infrastructure independence it performs all tasks natively (Figure 2.1a).

In 2003 the author ported ARToolKit [57] to the PocketPC and developed the first fully self-contained PDA AR application [115]. This platform was used in a peer to peer game in [111]. Möhring et al. were the first to successfully target a consumer smartphone for mobile AR [72]. The scarce processing power of the target platform allowed only a very coarse estimation of the object's pose on the screen. Henrysson ported ARToolKit to the Symbian platform and created the first two-player AR game [45] on current-generation smartphones.

Summarizing these developments one can conclude that there is no ideal solution for systems with scarce processing capabilities. An infrastructure independent solution, as developed in the work of this thesis is desirable, but not feasible for all situations. E.g. when artificial feature tracking is not an option, embedded devices simply do not have the processing capabilities yet. While this will certainly change in the future, new more demanding problems will emerge too.

2.1.2 Handheld Augmented Reality Gaming

Augmented Reality (AR) as a new user interface technology has yet to see its breakthrough into mainstream acceptance - but why? Generally speaking, new user interfaces are often employed first in professional applications, where potential gains in productivity can justify high investments and even allow for some user interface specific training if the learning curve is not too shallow. In contrast, entertainment applications must immediately appeal to a mass audience, need to be self-explanatory and do not permit high hardware cost. On the surface it seems that serious applications have an edge over gaming as a vehicle for user interface research, but actually the opposite may be true: Players of computer games tend to tolerate to glitches in software quality that would be deemed unacceptable for professional applications, as long as play value and usability of the interface are outstanding. This makes games very suitable to test research on user interface technologies such as AR.

Returning to AR interfaces, the main barrier that has hindered bringing AR games to a mass audience is the lack of an inexpensive hardware platform. The advent of ARToolKit [57] as a free tracking/graphics starter kit has led to significant growth of individuals (most of them not researchers) experimenting with desktop AR. However, desktop AR with a stationary camera (webcam) looses a lot of appeal over direct viewpoint control with a head-mounted display (HMD), and neither HMDs nor high quality mountable cameras are standard peripherals available to a wide audience. It is likely that the unavailability of a commercial off-the-shelf device to show AR content has severely affected the potential growth of this technology.

The proliferation of handheld computing devices may bring a solution to this problem. Handhelds in the form of tablet PCs, personal digital assistants (PDAs) or smartphones are well-engineered, widely available and inexpensive. Using live images from their built-in cameras as a video background, they can display video-see through AR. This style of interaction is sometimes called magic lens metaphor [15][106]. The wide-spread adoption of handhelds allows researchers to draw from a large target audience of users already familiar with the general operation of the target device; many users may even own handheld devices already.

Casual games are becoming increasingly popular on cell phones, so that handheld AR games are also perceived as socially acceptable, but at the same time new and exciting. The expectation that casual games should have short playtimes helps researchers to set up satisfactory experiences without having to produce a lot of game content. Possible target platforms range from conventional cell phones, on which software-only solutions could allow immediate commercial marketing, to high-end handheld and Tablet PC solutions which are useful for proof-of-concept implementations until the lower end of the market has reached sufficient performance levels. In the following some examples are described.

Penalty Kick [90] (see left and middle image in Figure 2.2) uses a coarsely registered 3D marker, which can be printed on a poster or product package. The aim of the game is to shot a soccer ball into a goal printed on the product package. The player can aim where to shot the ball by rotating and tilting the phone. The virtual goal keeper will then try to hold the ball.

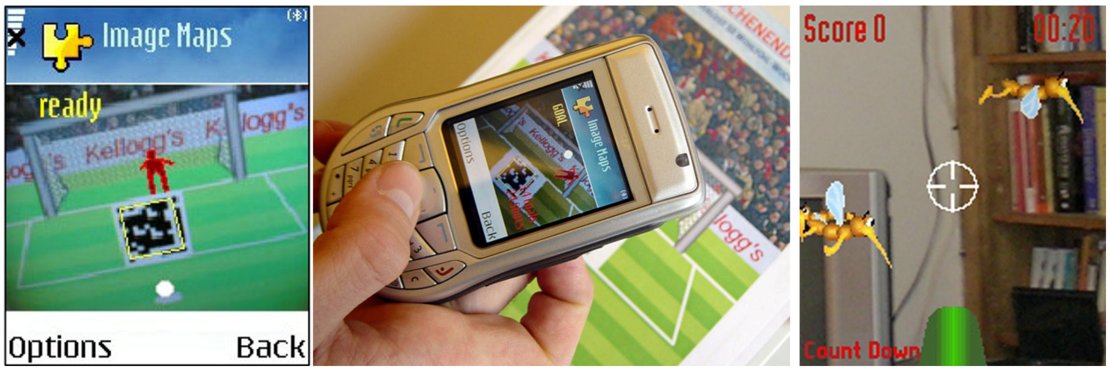


Figure 2.2: Left and middle: Penalty Kick (images property of Michael Rohs); Right: Mosquito Hunt (image property of Siemens)

Mosquito Hunt (see right image in Figure 2.2) challenges the player to shoot mosquitoes. The gun is pointed at the mosquitoes by moving the phone in space. A simple pixel flow detection algorithm makes the mosquitoes stay fixed relative to the environment. The world is captured by the build in camera acts as a backdrop for the game, while the mosquitoes are rendered as 2D sprites on top of that background. The game only measures orientation, so that actually only the orientation of the device matters, while the position is irrelevant.

Mobile Maze [22] (see left image in Figure 2.3) goes one step further in the direction of pure VR by turning the handheld device into a purely simulated handheld maze game. The player has to guide a ball through a maze by tilting the physical maze itself. The software visually tracks the phone's orientation using a marker, but does not display any video image. Mobile Maze displays the whole maze on the screen, so that the impression of a handheld physical maze is suggested. Another variant of the same idea, Marble Revolution[4] always centers on the Marble, while scrolling the game field, which is much larger than the screen. Marble Revolution has a physical interface, but otherwise no aspects that qualify as Mixed Reality. Instead, the player has to navigate a ball through large, scrolling levels by moving and tilting the phone. In contrast to Mobile Maze, Marble Revolution uses pixel flow detection to navigate the ball and does therefore not require fiducial markers.

AR Soccer [36] (see middle image in Figure 2.3) shows a virtual soccer goal and blends in the handheld's video image in the bottom half of the screen, letting a user

[4] http://www.bit-side.com

view his own foot in the soccer environment. The aim of the game is to shot a virtual ball into a virtual goal. To do that, the player has to kick the ball with his foot, which is tracked using an advanced pixel flow detection algorithm. In contrast to simpler pixel flow methods such as used by the Mosquito Hunt game, ARSoccer accurately detects the edge of the moving foot and can thereby calculate the exact speed and direction of the foot hitting the ball.

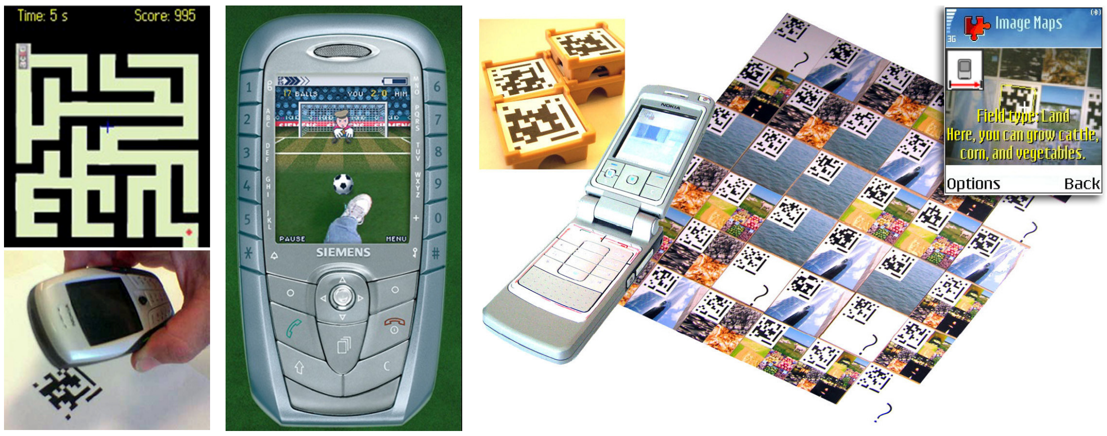


Figure 2.3: Left Mobile Maze (images property of HitlabNZ); Middle: AR Soccer (image property of Siemens); Right: Impera Visco (image property of Michael Rohs).

Impera Visco [90] (see right image in Figure 2.3) is a typical turn-based (also called "hot seat") multiplayer strategy game for cell phones that includes many physical elements of classical board games such as dices, pieces and cards. The game uses 36 cards that represent different resources and operations. In each game the cards are arranged differently, requiring the players to scan the game board with the mobile at game start. The mobile phone acts as a game manager rather than a 3D graphics display. Since the game is turn-based, only one phone is required, which can be passed on to the next player.

Symball [42] (see left image in Figure 2.4), a multi-player table tennis game for Symbian phones was developed in 2005 by Video Processing Team at VTT (Finland). The game shows a table tennis game from a player's perspective. Although the table and the ball are shared conceptually, no tracking is performed on these and therefore no shared space as described above exists. The game tracks the phones movements by detecting objects of certain color in the camera's video feed. While the table is painted as a static image, the paddle can be moved by tilting the phone. Two players can connect their phones via Bluetooth to compete in a game. The disjoint players' spaces in Symball theoretically enable remotely playing together, but in practice the short range of Bluetooth limits this option.

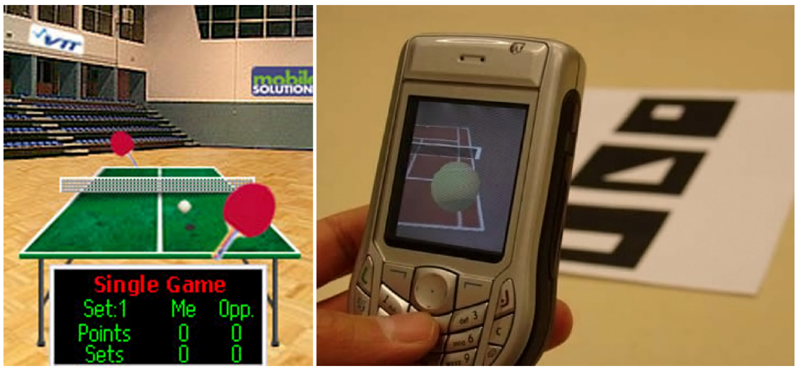


**Figure 2.4: Left Symball/Pingis (image property of VTT);
Right: AR Tennis (image property of HitlabNZ).**

AR Tennis [45] (see right image in Figure 2.4), developed in 2005 at HIT Lab New Zealand also lets two players share a game of tennis but uses markers to establish a shared space for the players. The phone itself is used as a paddle to hit the ball, which requires a lot of physically movement with the device. Each phone can be fitted with a marker on the back so that it can be detected by the opponent's phone for visual feedback.

2.1.3 Augmented Reality for Edutainment

AR as a new medium is attractive for education institutions such as museums aiming at increasing the interest in their traditional exhibits through technology. The incorporation of AR enhanced exhibits can range from a single high-performance AR display [18] to an entire exhibition realized using Mixed Reality special effects [98]. On the one hand, stationary AR exhibits allow the use of technology without compromising the experience with respect to form factor or power consumption. On the other hand, mobile AR technology can offer an attractive replacement for the traditional audio-tape tour guide.

Tour guides are a recurring application theme for mobile AR research, partially because they show the strength of mobile AR, namely to present useful information registered to static locations in the real world, such as historic edifices. Some examples of outdoor AR guides are Situated Documentaries [48], ArcheoGuide [107], GEIST [50] and Signpost [87]. Work on mobile indoor AR has specifically targeted museum environments, for example the Guggenheim Bilbao museum [39] or Sweet Auburn [67].

Also relevant in a broader sense are mobile multimedia tour guides that do not use AR displays, but provide location-based interaction with the environment. Prominent examples are CyberGuide [64], Lighthouse [20], MIT's Mystery at the Museum [58] and the phone guide at Senckenberg Museum [32].

2.1.4 Virtual Characters

Aiming at developing intuitive and easy to use user interfaces, this thesis also evaluates the use of virtual characters, which are today a standard mechanism for tutorials and guides in many applications including games. Virtual characters guiding unfamiliar users have a long tradition in computer research and applications. The most prominent, yet suboptimal examples are the office assistants such as the well known paper clip in Microsoft's Office until version 2003.

The work presented in thesis evaluates character representation and particularly the presentation of virtual characters in an augmented reality setting. For the first time, AR technology allows virtual characters to exist in the same real space as the user. Despite this, the use of AR characters has also not been well studied.

In the Welbo system [2] an animated virtual robot assists an HMD equipped user in setting up virtual furniture. Welbo has speech synthesis capabilities and can understand simple instructions. It is aware of the user's actions and movement and reacts to commands by moving furniture or pointing to objects.

In one of the first examples of an AR character, Balcisoy et. al. [6] created a virtual agent that could play checkers with a person in the real world. The agent didn't have any conversational ability, nor was it able to respond to a real user's speech and gesture commands. Simply by appearing in the same space as a real user, the authors say it creates a strong sense of presence. However, Balcisoy also did not report on any formal user studies exploring how the user's felt about the agent.

In the AR Puppet project Barakonyi studied how animated characters improve the man-machine communication in AR applications. In his work he focuses on the interaction of virtual characters with their virtual as well as physical environment. In the AR game MonkeyBridge [9] two players have to help their autonomous agents in form of monster-like characters to cross a river. The characters are not scripted but intelligently decide which virtual and physical objects to use in order to accomplish the task. The AR Lego [8] application employs two agents: a physical robot and a virtual animated repairman to assist an untrained user in assembling and maintaining a LEGO Mindstorms robot.

Augmented Reality interaction with mobile devices is inheritably different from using HMDs or projection systems. As discussed in the next chapter using mobile displays allows new types of AR applications. These mobile applications typically use the touch screen or devices buttons rather than data gloves or 3D pointing devices as traditional AR setups do. Furthermore since the AR system itself is extremely small it allows new ways of shared applications, for example by passing the display from person to person.

Most real life applications can not be operated with a typical AR interface that uses 3D user I/O only. Instead, many applications require 2D interfaces, which are a natural strength of devices such as mobile phones over HMDs or projection-based systems.

2.2 3D User Interfaces

3D user interfaces seem to be the most natural UI method for AR applications. In the past much research concentrated on using data gloves in VR and AR setups, on gesture recognition [47] and object manipulation [63]. While data gloves can be used to create natural and fully 3D user interfaces [103], they usually do not allow accurate selection and manipulation and suffer from supporting complex action sequences and extensive tool or command sets.

Augmented Reality using tracked displays mandates the design inherently different user interfaces. The magic lens metaphor afforded by handhelds imposes very specific constraints to interaction design. The device must be held at a distance of about 50cm, with the camera normally tilted downwards, to allow for prolonged use without significant fatigue and also to let the user focus on the screen. The field of view defined by the small handheld screen is therefore very limited. This means that the amount of content that can be displayed - both world-registered and screen-registered - is rather constrained.

It also implies that in order to observe a physically large environment, the device needs to be frequently moved or rotated. Ergonomic constraints and the necessity to keep a line of sight to the display limit the type and amount of possible movements of the handheld. While rotation and movement with the supporting arm are quick, moving the device through physical walking is more disrupting since it is often difficult to keep the screen in view while physically navigating the environment.

Many application designs will therefore aim to minimize such physical movements. For example, devices that feature a touch-screen can be held still while interacting with the environment using the stylus. A similar approach may be taken using the miniature joysticks often found on cell phones. However, the author observed that the enjoyment of physically navigating the environment is one of the key contributions to the appeal of handheld AR applications in general and games specifically. Of course a part of this success must be attributed to the fact that a larger, navigable environment is specifically prepared to support the application. The discussion will return to the issue of complex infrastructure below.

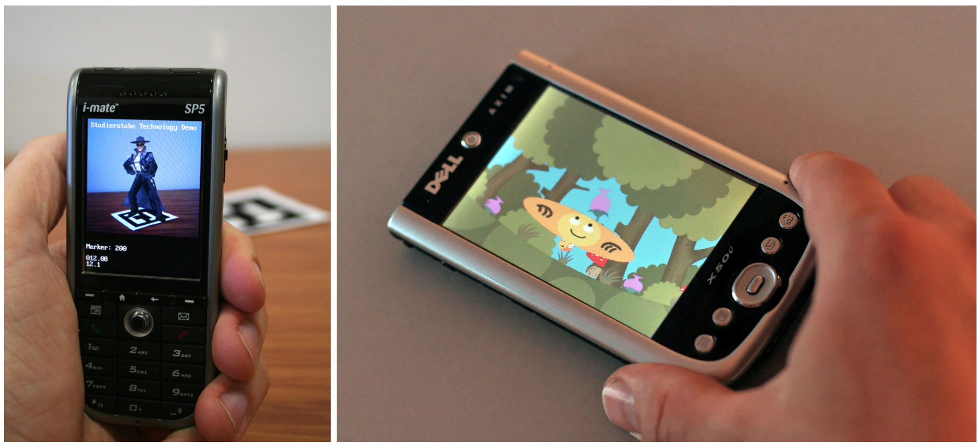


Figure 2.5: Left the screen of the handheld represents the AR display; Right: the handheld itself represents the AR display.

The handhelds' small field of view introduces some ambiguity when trying to assess AR applications with respect to the Virtuality continuum: A user will typically focus on the handheld's screen, but simultaneously perceive context from the real environment around the handheld. The handheld is so small that it can be interpreted as a kind of "cursor" into the physical environment. Therefore, there are two possible interpretations of an "Augmented Reality display":

- **The screen of the handheld represents the AR display**: This interpretation is most meaningful if the handheld display superimposes computer graphics on top of a video stream from the built-in camera. In this case the physical environment is duplicated in miniature format on the display, and becomes a conceptual part of the application. A handheld or Tablet PC with a slightly larger display and a stylus is likely to strongly bind the user's attention to the device, while diminishing the user's perception of the surrounding. The left image in Figure 2.5 shows an example of this concept.
- **The handheld itself represents the AR display** (in this case, more an MR display): The handheld display shows exclusively virtual content, but this content is still fully registered to the physical environment. Moving the device

in reality also moves the position (viewpoint) in the virtual world. In this case the handheld itself is the AR display, since the display content must be interpreted together with the surrounding real world. One could say that the handheld is at the same time an AR display and a tangible interaction device. This definition is more in line with Benford's "shared space" approach towards mixed reality [14] which is much broader than the Virtuality continuum. The right image in Figure 2.5 shows an example of this concept.

Some designs take a relaxed approach concerning registration and utilize only the rotational degrees of freedom from the device pose, which allows using weak tracking methods such as pixel flow detection. This decouples the virtual world from the real world, since the focus of interaction is no longer important and rotational changes are measured incrementally. Examples for this form of interaction are the Mosquito Hunt, Mobile Maze and Marble revolution games which are covered in chapter 2.1.2 of this thesis. All three games use optical flow detection interpreted as rotational movement.

Even when the real and virtual worlds are fully registered, the use of the real world may be purely to navigation mechanics, but have nothing to do with the application semantics. A popular approach for instant AR is the placement of a marker on a table, which is then tracked by the handheld's camera. While the virtual content will remain registered to the real world while the handheld is moved, the marker itself has no meaning other than defining a coordinate system for interaction. The location of the marker is arbitrarily chosen by the user and has no influence on the application unless it is moved. An interesting variety of using tracked objects such as markers is that if multiple such objects are used, their identity and placement can be used to manipulate the application with a tangible user interface. In particular, tracked objects can be moved while the handheld remains relatively stationary. Examples for this type of interaction include AR Kanji Teacher [109] as well as the AR soccer game referenced in chapter 2.1.2.

Another aspect of UI design is how applications share the virtual space. The following list considers multiplayer games to outline various possibilities:

- **Sharing a single device:** The simplest form of multiplayer sharing is the sharing of the handheld device itself. While this obviously has the disadvantage that no simultaneous interaction is possible, it suits turn-taking games very well and is also popular in desktop games. Technically, the advantage of device sharing is that only one device with suitable software is required, which obliterates the need for software installation or networking, and is very suitable for instant, casual play. A game using this interaction paradigm is Impera Visco [90].

- **Co-located simultaneous interaction:** Simultaneous AR gameplay using multiple networked devices can either be constructed from multiple individual AR spaces, i.e., one per user, or by constructing a shared space jointly observed by the players. Disjoint AR spaces are technically simpler and work for users who are in different locations. A shared space has more stringent requirements concerning technical issues such as tracking accuracy or network latency, but has the unique appeal of combining computer games with physically playing together, being able to engage in a lively conversation and observing the opponent's reactions. We have found these social aspects to be a strong factor of motivation and enjoyment in AR games. The Symball [42] game uses this principle to share a static and non-registered virtual ping-pong table.
- **3D registered shared space:** A shared space can be defined by a generic object such as a marker as done in the AR Tennis [45] game. In a minimalist setting no further real world aspects are considered. Larger instrumented environments typically have room for multiple players, as was implemented it in the Virtuoso game (see chapter 7.2).
- **Large area shared space:** In the extreme the game space can encompass a large area (e.g. a campus), supporting both face to face and remote gameplay at the same time. This option is extremely compelling, since the immersion in the "game world" is paid back in heightened excitement of the players. Nevertheless, one must consider the effort involved in preparing such a larger, navigable environment to support the gameplay, which is definitely not reconcilable with instant, casual gaming. While placing a game board shipped with an AR game on a tabletop may be straightforward, a game that involves physical museum exhibits is only playable in exactly the museum it was designed for. Such a complex environment will likely be designed for a larger number of concurrent players, and will make the collaboration between the players more complex. The “Schatzsuche” museum game (see chapter 7.3) is currently the only representative of this kind of application we know of.

2.3 Augmented Reality Authoring

Content creation, also known as authoring describes the part of software development that is not focused on programming but on content creation. For practical content-rich applications content creation is a major topic. While small projects can survive using a complex chain of authoring tools, large applications require a smoothly working content development pipeline. This issue is not unique to AR applications, but has become a special problem in the area of professional game development.

Many research projects use the Virtual Reality Modeling Language (VRML)[5], a powerful 3D scene description language with an OpenInventor-like[6] syntax. While tools exist that allow creating simple scenes without programming, it is not an authoring tool in the classic sense. Although VRML has been superseded by the XML-based X3D standard[7] it is still one of the most widely used graphics formats.

The Designers Augmented Reality Toolkit (DART) [66] specifically targets non-programmers. Based on Macromedia Director[8] it provides a user interface that is well known to graphics artists and designer. The timeline concept allows the creation of non-linear content in an intuitive graphical way. Another project targeted at non-programmers is the MARS authoring toolkit [40]. Similar to DART, the custom developed editor uses a timeline to author media that are linked in a hypermedia fashion. The Authoring Mixed Reality (AMIRE) [39] project focuses on combining basic building blocks, called Gems into a mixed reality application. Developers can use a visual editor to connect and configure the Gems. Yet, the authoring, rather than application development aspect itself is only minimally present in AMIRE. APRIL [60] is an XML-based scripting language that uses the Studierstube framework and adds high level concepts on top of it. APRIL uses a finite state machine (FSM) to create non-linear stories and provides abstraction mechanisms to describe the needs of the specific application that are then automatically mapped to the available hard- and software.

Other projects use in situ modeling where the authoring editor itself is an AR application. While such an approach is definitely not optimal for every type of application it allows editing AR content in place, which is especially useful for mobile scenarios. The Tinmith-Metro system uses the mobile Tinmith AR setup to provide content developers with tools for creating virtual content directly in place using the mobile setup. Predefined virtual objects can be dropped into the real environment and modified using a tracked glove. Güvem et al. [41] use a mobile device to create and edit hyperlinked situated media on-site. They explore several novel interaction paradigms to work with these linked annotations. Lee et al. use tangible AR for

[5] http://www.w3.org/MarkUp/VRML
[6] http://oss.sgi.com/projects/inventor
[7] http://www.web3d.org/x3d
[8] http://www.adobe.com/products/director

"immersive authoring" [61] that enables the user to create virtual scenes and script simple commands directly in the AR environment.

The tools mentioned above can be classified into two main categories: authoring in the AR runtime itself and authoring using professional toolkits. Both are not optimal for phone-based AR: The reduced input and processing capabilities of mobile phones make these devices unattractive to be used as platform for complex authoring applications. Furthermore, the typically employed vision based tracking systems do not allow working accurately. Using professional toolkits such as in the DART system raises the problem of deploying the corresponding runtime on the target platform. For the example of DART, no Adobe Shockwave runtime exists for mobile phones. Hence the author of this thesis decided to develop a custom authoring solution as presented in chapter 6.2.

2.4 Pose Tracking

Any Augmented Reality system requires some kind of tracking the user's or display's pose in order to register it in respect to the real world. Pose tracking must run in real-time, typically requiring solutions that estimate poses in less than 50 milliseconds. Furthermore it must be robust under many conditions such as varying lighting. In case tracking is lost, the system must be able to recover quickly.

Much work in mobile AR has focused on wide-area tracking. Most commercial solutions such as optical or infrared trackers cover only a limited work area, so researchers have aimed at using e. g., GPS [34], inertial sensors [4], and vision [88] for tracking. The Bat System [54] from AT&T allows building-wide accurate tracking of people and objects outfitted with badges that are tracked by a 3D ultrasonic location system, but at the cost of deploying a building-wide electronic infrastructure.

2.4.1 Natural Feature Tracking

Recently processing power has reached a level that allows natural feature tracking in real time. Some recent examples are: Bleser [19] uses a 3D CAD model to initialize the tracking process. The system can then extend its model of the environment automatically and even adapt to changes. Reitmayr [86] uses textured 3D models of the real environment to track in urban outdoor environments. Pilet tracks and augments predefined deformable objects [78] in real time. Vacchetti combines edge- and textured-based tracking for realtime pose estimation [105].

The tracking methods mentioned above are state of the art and therefore not suitable for handheld or embedded devices due to limited processing and video capturing capabilities. Natural feature tracking using optical flow has been successfully implemented on these devices though. Among the first applications to use pixel flow detection on mobile phones was the Mosquito Hunt game on the Siemens SX-1 phone. Since then more mobile phones games have used this tracking method. Wang recently released an open source pixel flow tracking library called TinyMotion [108].

2.4.2 Marker Tracking

If limited computational resources do not permit robust markerless tracking, fiducial marker tracking is often used in AR applications. One of the first projects using camera-based 6DOF tracking of artificial 2D markers was Rekimoto's 2D Matrix Code [82] in 1996. It pioneered the use of a square planar shape for pose estimation and an embedded 2D barcode pattern for distinguishing markers. In 1999 Kato used a similar approach to develop ARToolKit [57], which was released under the GPL license and therefore became enormously popular among AR researchers and enthusiasts alike. Since then, many similar square tracking libraries have emerged among which the most prominent ones are ARTag [25], Cybercode [84], the SCR marker system [117] and the IGD marker system used in the Arvika project [33].

ARToolKit is the basis for several projects concentrating on 6DOF tracking on handheld devices. It uses black and white square marker that can be easily detected even under low lighting conditions. Furthermore a single marker is sufficient for full 6DOF pose estimation. The author’s port of ARToolKit to Windows CE [112] led to the first self-contained handheld AR application [115] in 2003. This work evolved later into the ARToolKitPlus library [113] detailed in Chapter 3. In 2005 Henrysson [45] created a Symbian port of ARToolKit partially based on the ARToolKitPlus source code.

Other researchers tried making best use of the restricted resources of low to mid-range mobile phones by using simpler models with very restricted tracking accuracy. In 2004 Möhring [72] created a tracking solution for mobile phones that tracks color-coded 3D markers. At the same time Rohs created the Visual Code system for smartphones [89]. Both techniques provide only simple tracking in terms of position on the screen, rotation and a very coarse distance measure. VTT developed a marker system [96] very similar to Rohs' Visual Codes that does not provide pose estimation though.

2.5 Multiuser Systems

Over the last decade multiplayer gaming has become extremely popular and today represents the part of PC-based gaming where the highest revenue is made. Early multiplayer games used peer-to-peer network topologies with a limited number of clients. Today's multi-player games typically use client-server techniques, which allow for better scalability and also an improved separation of concerns in the overall system architecture. Despite a massive amount of research in the area of designing and implementing massive multiplayer services, they still represent a challenging endeavor.

As mobile devices are becoming increasingly available, shared spaces that were previously bound to desktop computers within a fixed infrastructure are now moving to handhelds such as cell phones or PDAs. Mobile applications and services are not simply a trend towards a more convenient platform, but a fundamental paradigm shift. For example, mobile games have the potential to combine wireless networking with location based computing. The resulting Mixed Reality Games [98] bring the game play out of the virtual world and back into the real world. Hence, the requirements for MR and AR games are inherently different from conventional online games.

This chapter analyzes the most common problems of mobile, multi-user Augmented Reality applications, which were addressed with the Muddleware system, before it then looks in detail at solutions developed in previous research projects:

- **Communication** is not guaranteed to be always available. When relying on wireless communication such as WiFi or GPRS, high-quality, poor or no communication at all may be available depending on the current location of the device. Hence, the system must support both strongly as well as loosely coupled connections. Loosing network connection during application execution is not an exception but rather a rule. Users, thus, must be able to enter and leave sessions at any time and even continue despite being temporarily disconnected [10]. These special circumstances affect the design space for applications and are therefore best made explicit in the networking middleware.

- **Heterogeneity** in the client platform is common for situated Augmented Reality applications. AR clients differ significantly in their performance, user interface capabilities, and mobility. Making the best use of this diversity can be an important aspect of AR game design rather than an unwanted side effect. For example, users of mobile and stationary clients can exploit their complementary capabilities for collaborative problem solving. Moreover, games that leverage the player's location in the real world for gameplay – for example, by requiring a player to perform certain actions at a specific location – require the game platform to interface with a large variety of sensing devices and other input/output facilities in the real world. All this diversity makes it unreasonable to expect the deployment of heavy-duty AR and networking packages on client devices. Software for game and non-game clients must be lightweight, highly modular and available for a very wide variety of platforms, so that a client can participate in the shared space with only a minimum of required software.
- **Persistence** of application content and state becomes an increasing concern as the level of heterogeneity and distribution increases in Augmented Reality applications. Like online role-playing games, AR games are situated in a persistently evolving universe – however, in case of AR, this is the real world. Since predictable behavior or even reachability of clients in the real world cannot be expected, all important application data reflecting the real and virtual parts of the application world must be stored persistently in a reliable database. The difference to conventional online databases is that the structure of the database must reflect the diversity of the real-world entities, and can usually not be limited to just a few artificial object categories without severely compromising the design space of the application or game.
- **Rapid prototyping** becomes increasingly important since there are no established design practices or genre standards for Augmented Reality applications or games as there are for their desktop counterparts. AR games are currently mostly developed by small teams in academic or commercial research labs. While this situation may evolve over time into a more traditional development process, in the near future the possibility to quickly change game database content and game protocols is of prime importance. From a technical perspective, this means that network communication between the distributed application components should be accessible via loosely typed, interpreted scripting.

Virtual Reality researchers have long ago started working on distributed systems and tele-presence, allowing non-colocated users to share virtual environments. Among the first projects is the work done by Benford [12], Zyda [119] and Snowdon [97] in the early 1990s. While CORBA[9] is a well established solution for distributed systems, many researchers favor more specialized approaches such as distributed scene-graphs [74][46][104]. Although a tremendous amount of work has been put into these systems, they have not been widely successful, although developers of massive multi-user games nowadays benefit from these works.

In contrast to Virtual Reality, only few AR projects tackled the problem of distributed systems. Among these is DWARF [11], a component-based distributed AR framework. These systems are similar to those developed in the area of ubiquitous computing since they share the same problems: They deal a lot with research allocation and management as well as service discovery. Yet, most multi-user AR systems allow just either ad-hoc or stationary networking. Consequently the number of concurrent users in these systems is generally very low.

To address these specific needs for AR middleware for many users in unreliable and spontaneous networked environments, the author of this thesis has developed a communication framework called Muddleware. It is originally inspired by the concept of a Tuplespace [37], an associative memory that stores a collection of tuples. The name Muddleware refers to a complex "pile" of data tuples. Tuplespaces are sometimes also called Blackboards or Whiteboards. They are the theoretical foundation for Linda [38], a programming language for parallel algorithms. The idea is still popular in modern implementations such as IBM TSpaces [53] and JavaSpaces [27].

While a classic Tuplespace is a very general data structure that is hard to implement efficiently the approach described in this thesis is specialized for the use of multi-user setups. Another projects that is based on the concept of a Tuplespace is LIME [73] (Linda in a Mobile Environment), extends the original Tuplespace concept by transiently sharing Tuplespaces among multiple clients. The Tuplespace is broken up into many partial Tuplespaces, each one permanently associated to and stored on a specific client unit. When a new client connects to the federation of Tuplespaces, its current content is merged into the shared space by making it available to all other clients. While all clients can query tuples from all other clients and even add new tuples, all individual Tuplespaces remain on their original client. When a client leaves the federation it takes all its content with it, which is then no longer available for the other clients. The transient sharing makes LIME very dynamic, but is inappropriate

[9] http://www.corba.org

for deterministic application state since the availability of essential data cannot be guaranteed.

In contrast, the Event Heap [55] was very influential for the development of Muddleware. The Event Heap is an extended Tuplespace for synchronous groupware. It allows sharing events among multiple users, applications and devices in a distributed, interactive workspace. The Event Heap replaces the event queue commonly used in operating systems, which was found inappropriate for distributed groupware setups. The Event Heap extends the original Tuplespace concept with new capabilities such as non-destructive reading of data, a lifetime property for tuples to realize a kind of garbage collection, and a publish-subscribe mechanism for reliable causally ordered communication. Among the key techniques useful for interactive applications demonstrated in the Event Heap is the use of Tuplespace-style queries for filtering events that are interesting to a client without the need for explicit producer-consumer network channels.

The Enchantment Whiteboard [23] developed in the MIThrill project uses a Tuplespace like system as a central hub for data exchange between various wearable components in a Body Area Network. It extends the Tuplespace idea by allowing clients to subscribe to portions of the whiteboard to automatically receive updates when changes occur. To support concurrency, clients can lock parts of the whiteboard before posting updates. Furthermore the Enchantment Whiteboard supports symbolic links to other whiteboard servers allowing transparently referring data across the network.

Another system that specifically addresses the problems of unreliable wireless network connections was developed for the Battlefield Augmented Reality System (BARS) [21], an Augmented Reality application that superimposes battlefield information for soldiers directly onto the environment. In the BARS project Brown et al. "Event-Based Data Distribution Mechanism" based on replicated databases and custom network protocols.

Several generic middleware solutions for massive multi-player online gaming exists, such as the Distributed-organized Information Terra (DoIT) platform [51] and the Internet Communication Engine (ICE) [44]. ICE represents a light-weight alternative to CORBA with advanced features such as grid computing, persistence services and encryption. BigWorld Technology[10] is a commercial MMO suite that covers the complete development process from server application to 3D client libraries. All these commercial multiplayer solutions can be categorized as message-passing or remote procedure call platforms. The ADAPTIVE Communication

[10] http://www.bigworldtech.com

Environment (ACE) [93] is a network programming library that is used intensively in Muddleware.

For Muddleware, the Tuplespace-inspired approach was chosen over more conventional message passing for very similar reasons as stated by the creators of the Event Heap and Enchantment Whiteboard. A Tuplespace allows data-driven prototyping, flexible communication patterns and implicit persistence. While the Muddleware approach shares those virtues with previous work, its implementation differs in the choice of system characteristics and increased expressive power of the communication primitives.

2.6 Wireless Connectivity

As outlined in the introduction of this thesis, network connectivity is a fundamental requirement to exploit the full potential of handheld augmented reality. Especially for gaming applications – collaborative or competitive as well – a shared virtual space is essential.

An ideal network solution would allow connecting mobile clients in a peer-to-peer fashion as well as to stationary servers. Of course such a solution should provide coverage over a large area with a sufficient bandwidth and response time. Finally this network infrastructure should have low costs for deployment as well as for permanent usage. Of course such a solution does not exist and one has to choose between the strength and drawbacks of those networking technologies that are available right now. The following list analyzes all these systems. While it would be preferable to give a clear preference of a most suitable solution for handheld AR, the applicability of all systems strongly depends on the actual application.

- **GPRS** is a mobile data service based on the widely spread GSM phone network. As a consequence GPRS service has the highest coverage of all available systems. Other than UMTS or WiFi it is available basically everywhere in Europe. Its major disadvantage is extremely low response times of usually around a second which renders it unsuitable for live data exchange for many applications. Since GPRS is usually lower prioritized than speech the GSM service, quality can vary enormously over time. It is most suitable for single user applications that only require connectivity to download missing data such as 3D models or annotations. Another current disadvantage that probably disappears in the future is the high costs of transmitting data.

- **UMTS** is a 3rd generation mobile phone network. Other than with GSM, data services are fully integrated rather than added on top of it later on. Although UMTS is marketed as high performance network, in practice network performance and signal quality vary heavily. UMTS requires a much higher density of base stations than GPRS which means that even in the long term future, high performance networking via UMTS will probably only be available in populated areas. Response times are typically around 200ms and therefore a lot lower than when using GPRS.
- **WiMAX** is a high performance wide area network that provides data service only. As such it compares to WiFi rather than UMTS. Other than WiFi it requires licenses to operate and therefore creates provider/client partnerships similar to UMTS and GRPS though. WiMAX is specified to provide up to 70MBit/sec at short range and 10MBit/sec at long distance (10km). Since WiMAX is a very new technology coverage and number of providers are still very low.
- **Bluetooth** is a standard short range (so called personal) network for up to 7 client devices. Its primary purpose is to wirelessly connect multiple devices of a single user such as notebook, mouse and head-set. While the connectivity range class 3 devices is only about one meter, class 1 devices can transmit data up to a hundred meters. The Bluetooth standard is constantly improved. The upcoming revision 3.0 will allow full USB 2.0 speed at 480 MBits/sec, but current devices max out at 700kBit/sec only. Being a personal, local network solution there are no costs of running the network. Due to its inherent restrictions such as maximum number of client devices or the short range Bluetooth requires other hardware and software infrastructure to cover an area as large as a building or campus.
- **WiFi** (also known as wireless LAN or WLAN) is the wireless counterpart to regular tethered networking. Its technology is fully compatible to regular Ethernet systems including routing and switching. Its range of operation is typically room or building wide. While projects such as Funkfeuer[11] cover complete cities they rely on stationary setups with large, accurately placed antennas which prohibits this technology in mobile applications. Practical bandwidth is around 20MBit/sec and response times are of usually below 10 milliseconds which renders WiFi an ideal solution for live data streaming and exchange. A disadvantage is that spontaneous peer-to-peer networking is usually not possible. Instead base stations have to be used to cover specific areas.

[11] http://graz.funkfeuer.at

From the solutions described above WiFi will be preferred, for most purposes. Once a network is deployed the costs of running it are very low as there are no fees as opposed to GPRS, UMTS and WiMAX. The performance of WiFi is best among all available options. A disadvantage is the small area of network coverage though. As soon as wide are networking is required other solutions have to be used.

Many smartphones or similar devices do not support WiFi though. For short area networking Bluetooth can be a competitive alternative. Installing WiFi base stations equipped with Bluetooth USB sticks can be used to create a building network wide Bluetooth clouds that are interconnected via the WiFi network. More details on this technique can be found in chapter 7.4.1.

2.7 Discussion

This chapter presented a large body of state of the art in Augmented Reality from a wide range of different research topics with the aim of justifying the various different solutions developed in the course of this thesis, despite the many existing solutions published before. The next four chapters present technology developed for mobile phones based AR and several applications. Although previous work on each of these categories exists, we found these to be insufficient for the specific requirements and hypothesis of this thesis:

- Pose tracking (Chapter 3): No previous system was capable running at a meaningful performance on mobile phones and similar devices. Hence, porting existing and development of new solutions was mandatory.
- 3D rendering (Chapter 4): While professional 3D rendering solutions for mobile phones exist, we decided to develop a solution specifically tailored to the requirements of AR applications.
- Distributed computing (Chapter 5): Previous work on multi-user applications mostly assumes fast and reliable network connections, which is not the case in mobile computing. The solution described in Chapter 5 fulfills these requirements plus the need for a portable thin client module and support for rapid prototyping.
- AR framework (Chapter 6): The introduction of completely new building blocks as described above requires the development of a new framework that combines these into an optimal solution.

Chapter 3

Pose Tracking

This chapter introduces the concepts of fiducial marker tracking and presents ARToolKitPlus, a solution for marker tracking on mobile phones. It gives details on the phone specific features and presents performance measurements on typical devices.

Augmented Reality (AR) and Virtual Reality (VR) require real-time and accurate 6DOF pose tracking of devices such as head-mounted displays, tangible interface objects, etc. Pose tracking must be inexpensive, work robustly in changing environmental conditions, support a large working volume and provide automatic localization in global coordinates. However a guaranteed level of accuracy is generally not required.

Solutions that fail to address these requirements are not useful for VR and AR applications. In particular for mobile AR applications, all the requirements must be met while working with very constrained technical resources. The typical mobile AR configuration involves a single consumer-grade camera mounted on a head-worn or handheld device. The video stream from the camera is simultaneously used as a video background and for pose tracking of the camera relative to the environment. This inside-out pose tracking needs to execute in real-time with the limited computational resources of a mobile device.

Tracking fiducial markers is a common strategy to achieve robustness and computational efficiency simultaneously. While the visual clutter resulting from the fiducial markers is undesirable, the deployment of black-and-white printed markers is inexpensive and quicker than accurate off-line surveying of the natural environment. By encoding unique identifiers in the marker, a large number of unique locations or objects can be tagged efficiently. These fundamental advantages have led to a proliferation of marker-based pose tracking despite significant advances in pose tracking from natural features.

Today, tracking rectangular fiducial markers is one of the most widely used tracking solutions for video see-through Augmented Reality applications. This chapter describes ARToolKitPlus, an open source marker tracking library developed by the author of this theses and designed as a successor to the open source ARToolKit library [57], which is the by far most successful marker tracking library. Unfortunately ARToolKit was designed to run on standard PCs only. ARToolKitPlus is unique in that it performs extremely well across a wide range of inexpensive devices, in particular ultra-mobile PCs (UMPCs), personal digital assistants (PDAs) and smartphones (see Figure 3.1).

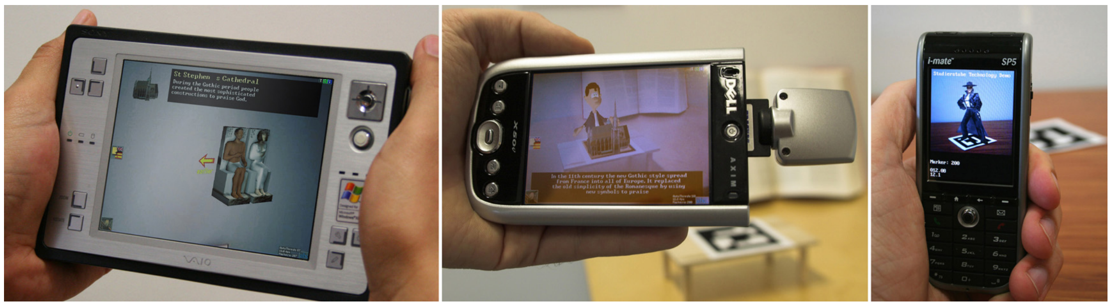


Figure 3.1: Devices running ARToolKitPlus: Ultra Mobile PC, PDA, Smartphone

3.1 Camera calibration

Before camera-based 6DOF tracking can be performed, the camera must be calibrated once in a pre-processing step. The results of this step are a perspective projection matrix as well as the image distortion parameters of the camera. These include the principal point, which defines the center of projection and is usually not exactly in the center of the image of a real camera. Furthermore the intrinsic parameters include the focal length and radial distortion parameters. The latter describe the lens' distortion using a radial model centered at the principal point, which represents a good approximation for most real cameras. Usually this distortion is rather low near the principle point of the image and increases at the corners. An extreme case of radial distortion can be seen in the left picture in Figure 3.2. All these parameters together form the "intrinsic camera parameters" and are saved in a calibration file that is loaded later on during the start-up phase of the tracking system.

ARToolKitPlus provides two methods for camera calibration. Users can use the ARToolKit camera calibration tool that requires holding a printout of 6x4 black dots in various poses in front of the camera (see left picture in Figure 3.2). For each pose, the user has to manually select all dots. When enough poses have been registered, the calibration tool estimates the camera's parameters. A complete calibration of a camera typically takes ~15 minutes.

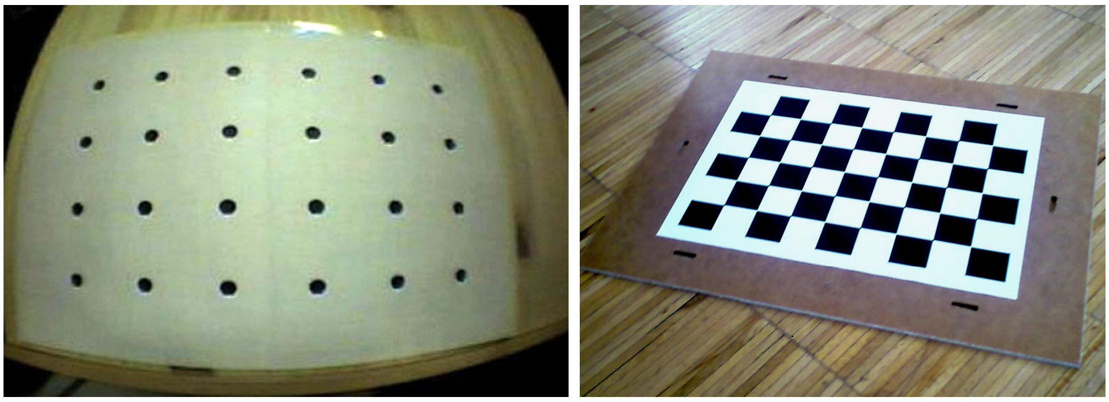


Figure 3.2: ARToolKitPlus camera calibration. Left: Using the ARToolKit calibration pattern. Right: Using the MATLAB checker board pattern.

Alternatively, ARToolKitPlus introduces calibration based on the freely available MATLAB camera calibration toolbox[12]. While this requires the user to own a license of the MATLAB software, it is more convenient, faster and produces more accurate results. Furthermore it allows calibrating cameras of devices such as mobile phones that are not capable to run the calibration software directly. For this calibration method the user has to take multiple pictures (usually between 5 and 20) of a checker board (see right picture in Figure 3.2). Since the image acquisition stage is decoupled from the calibration stage, any type of camera of can be calibrated.

The MATLAB camera calibration toolbox then loads the previously acquired images. First the user specifies basic settings such as the size of the checker board. Calibration can work in two modes: in automatic mode the toolbox tries to find all corners of the board autonomously, while in manual mode the user has to select the outer four corners of the checker board by clicking on them. In case the automatic mode fails on an image, it falls back to manual mode. The whole procedure typically takes 5 minutes.

[12] http://www.vision.caltech.edu/bouguetj/calib_doc/

3.2 Runtime Tracking Pipeline

The runtime tracking pipeline is executed for every new camera image and results in a set of zero or more estimated poses, depending on the number of markers identified in the image. It consists of the five basic steps symbolized as shaded rectangles in Figure 3.3. These steps are outlined in detail in the following chapters 3.2.1 to 3.2.5.

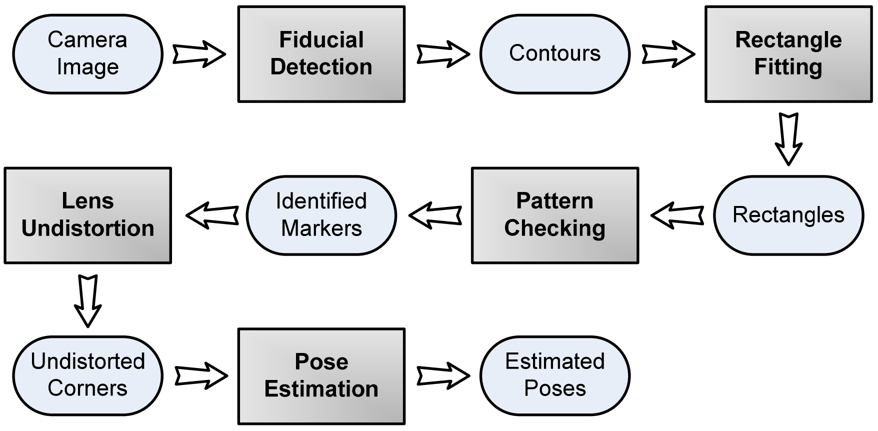


Figure 3.3: ARToolKitPlus runtime tracking pipeline

3.2.1 Fiducial Detection

In the sense of image based tracking, fiducials are objects or parts of objects in an image that are of interest to the tracking system. The natural first step is therefore to detect these fiducials so that they can be used for further processing in later steps of the tracking pipeline.

In ARToolKitPlus, fiducials are black rectangular markers that must be positioned in front of a bright background. Typically these markers are printed onto white paper. While the markers can be cut out of the paper, it is important to keep a white border around the marker to ensure that it can be robustly detected. A disadvantage of this method is that printer ink is highly reflective. Areas which are black can therefore appear as white in the camera image. For better results it is therefore suggested to create the dark marker regions using self-adhesive velvet foil, which does not reflect light under any circumstances and therefore results in more robust marker detection.

ARToolKitPlus uses a very simple fiducial detection system that is based on edge following: As a first step ARToolKitPlus searches line by line ("scanlines"), left to right for edges. There are multiple ways to define an edge. ARToolKitPlus uses constant thresholding for this operation: Pixels with a luminance value below a certain threshold are treated as dark and those above the threshold as bright. An edge of a black marker in a scanline is then defined as dark pixels that follow after bright pixels. When such a sequence of dark-right-of-bright pixels is found, it is considered as a candidate for a marker's border. The software then follows this edge until it either closes the loop back to the start pixel or until it reaches the border of the image. Due to the finite resolution of pixel images these are the only two outcomes of contour following. All pixels that have been visited are marked as processed in order to prevent following edges more than once. In case of a closed loop, the contour is stored as a poly-line and considered for further processing. In the other case the edge is simply discarded.

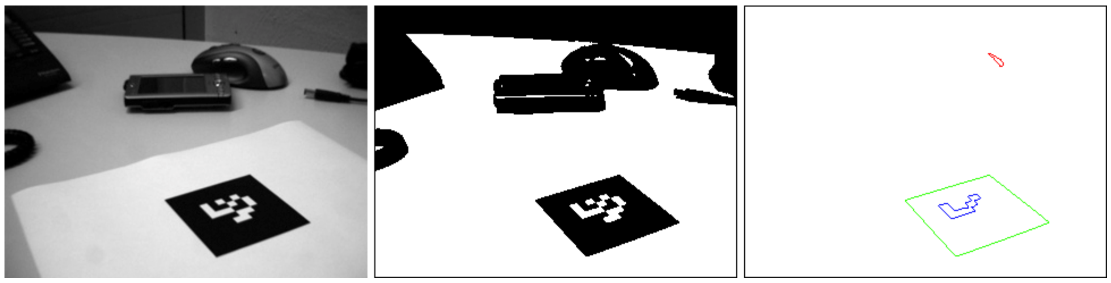


Figure 3.4: Fiducial Detection in ARToolKitPlus. Left: Source image; Middle: Threshold image; Right: Three closed polygons as candidates for rectangle fitting.

Figure 3.4 points out the workflow of the process described above: The left picture show the source image (as grayscale). The middle picture shows the binarized version of the left picture, thresholded with a constant value of 80. The right picture shows three candidates which are used for further processing. All other contours were discarded since they are not closed or too small.

3.2.2 Rectangle Fitting

As a next step all closed polygons need to checked for actually being rectangles. For this purpose a rectangle is defined as a 2D structure with 4 mostly straight lines that intersect in 4 corner points. Since a physical marker might not be perfectly flat and because the radial distortion can warp the camera image a considerably, it is necessary to use a relaxed method instead of searching for perfect straight line. An obvious choice would be line fitting, but this method is not feasible since the closed contours found in the previous step can consist of only four points (the corner points) which is unsuitable for line fitting. Furthermore the algorithm would not only have to estimate the line but also detect which points belong to the line and which do not.

Instead ARToolKitPlus uses an iterative process that incrementally detects more and more corners along the contour until either no more corners can be found or more than four corners were detected. In the latter case ARToolKitPlus discards the contour as not being a rectangle. Only if exactly four corners were detected the contour is identified as rectangle and stored for further processing.

To find corners, ARToolKitPlus guesses a first corner by selecting a contour point that lies at the maximum distance to an arbitrary point of the contour. For the case of a rectangle – even in the case of distortions – this operation always detects corner points. ARToolKitPlus then calculates the centre of mass of all edge points and creates a line through the first corner and this centre position. It then finds one corner on each side of the line by searching for those points that have the largest distance to this line. Having found three corners, ARToolKitPlus uses the same method to find more corners on further lines that are formed by already detected corners.

ARToolKitPlus uses the length of lines for selecting thresholds that determine how far points must be apart from the line to be treated as a corner. If a point falls below this threshold it is treated as part of the line rather than a corner. Finally ARToolKitPlus calculates the area of the rectangle as a second check that discards markers which are too small.

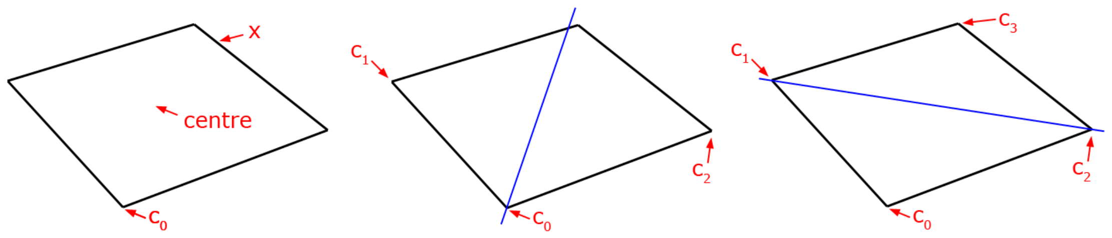


Figure 3.5: Example for fitting a rectangle to a polygon.

Figure 3.5 shows the process described above using an example rectangle. In the left picture, ARToolKitPlus selects an arbitrary point x and determines the point that with the largest distance from x, which must be a corner point labeled c_0. It calculates the centre of mass from all edge points and creates a line through c_0 and this new point (blue line in middle picture of Figure 3.5). It then finds those points which are most left and right of this line and labels them corner points c_1 and c_2. By building more lines using the points c_0, c_1 and c_2. it recursively determines more corner points. In the example above only one more point is found and labelled as c_3. Creating new lines from c_2 to c_3 and c_3 to c_1 reveals that no more corners exist. Finally the corner indices are sorted into clockwise order.

The rectangle fitting step runs purely in simple integer arithmetic (mostly multiplications, almost no divisions). It is therefore very fast and requires only a small percentage (~1-2%) of the overall performance on mobile phones.

For comparison, a different method for finding rectangles is using a line based approach, such as done by ARTag [25]. Instead of specifically searching for begins of contours and following them to check if they are closed, ARTag finds all pixels that contribute to edges independently and then groups those edge pixels into lines. An advantage of this approach is that it can tolerate partially occluded markers by allowing lines to be disconnected but still forming rectangles. A downside of this approach is that its performance strongly depends on the amount of lines visible in the camera image. Tests done by the author of ARTag reveiled that scenes with highly structured backgrounds create severe performance bottlenecks for rectangle detection on mobile phones.

3.2.3 Pattern Checking

After ARToolKitPlus successfully detected polygons with four corners and a suitable size it needs to check if these quadrilaterals are valid markers. To do this it first unprojects the markers' interior regions into a normalized arrays of pixels. For perspectively correct unprojection ARToolKitPlus calculates the homography matrix using the markers' corner point coordinates in the image and the knowledge that the marker must be regular rectangles with 90° angles in the 3D world (see left image in Figure 3.6). The homography matrix is then used to sample pixels (see middle image in Figure 3.6) from the image which are then written into a pattern structure (see right image in Figure 3.6). The size of the sampled patterns is arbitrary and depends on the pattern checking method used next.

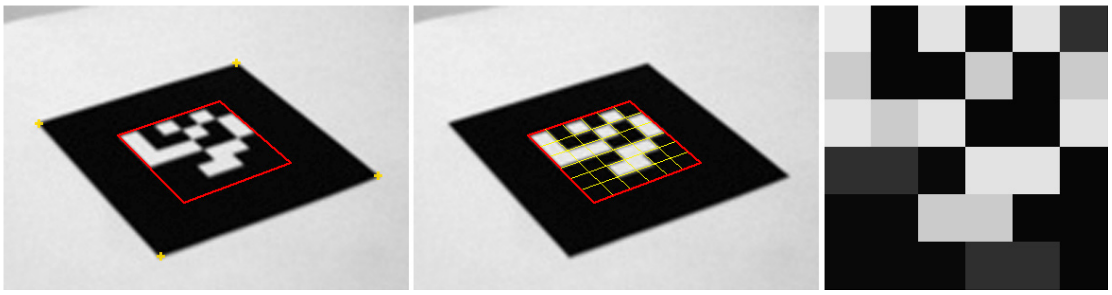


Figure 3.6: Unprojecting the marker content. Left: Interior area defined by corner points; Middle: sampling grid for unprojection; Right: Unprojected interior marker area.

ARToolKitPlus currently supports four different pattern checking methods. For applications that need to use arbitrary marker images, users can select ARToolKit template matching (see left picture in Figure 3.7), which checks the pattern area against a database of images using cross correlation. It thereby calculates the sum of squared differences between all pixels in the sampled and the database patterns. The disadvantage of this method is that it is computational expensive and scales badly with large numbers of known patterns: Each marker must be checked at four rotation steps. N visible and M known markers therefore require 4*M*N template matching operations.

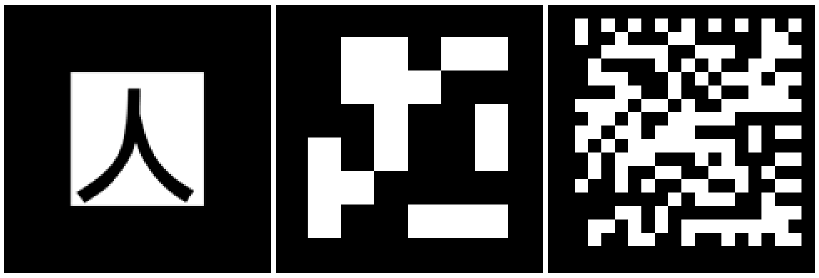


Figure 3.7: Marker types in ARToolKitPlus. Left: Template markers; Middle: ID-markers; Right: DataMatrix markers.

Alternatively users can select to use ID-based patterns (see middle picture in Figure 3.7). In contrast to template patterns, the black and white pixels in the unprojected pattern are directly interpreted as bit code from which a marker ID can be calculated. ARToolKitPlus' simple-ID patterns use simple four-fold redundancy and can encode nine bits in a 6x6 pattern image. Alternatively ARToolKitPlus' BCH (Bose, Ray-Chaudhuri, Hocquenghem) code patterns use cyclic redundancy checks (CRC) which require less redundancy for similar robustness. BCH markers can therefore store 12 bits in the 6x6 pattern image.

Detection of ID-markers is always faster than for template-markers since no image matching using cross correlation is required. Currently ARToolKitPlus supports up to 4096 id-markers. More markers could be supported at the cost of decreasing the id-detection robustness. ID-markers offer several more advantages over template markers (besides better performance): Although ARToolKit allows the user to choose almost any image for marker patterns, most users still choose their patterns out the small set of markers that comes with the ARToolKit distribution. With id-markers, the user does not have to provide marker images, but can freely choose any marker from a fixed set of 4096 patterns. In contrast to template markers, the user is not required to train ARToolKitPlus with new patterns since any valid marker is implicitly known to the system. The encoded id is highly redundant and is therefore robust against 90° rotation steps, which is a natural problem with square template markers.

Recently a fourth pattern type was introduced to ARToolKitPlus: Using DataMatrix[13] codes (see right picture in Figure 3.7), markers can encode complete URLs or small binary data sets. The DataMatrix ISO standard defines patterns up to 144x144 pixels that are able to store 1558 bytes or 2335 characters. Such large patterns are not suitable for tracking though since they'd only be correctly decoded with high camera resolutions and under small perspective distortions.

3.2.4 Lens undistortion

Before an identified marker can be used for pose estimation it must be undistorted using the intrinsic camera parameters that were estimated during the offline camera calibration step.

ARToolKitPlus does not undistort the complete image but only the coordinates of those points that are required for pose estimation. ARToolKitPlus has two different methods for calculating the exact position of marker corners: The original ARToolKit marker detector uses line fitting along the markers' borders to intersect the lines for corner estimation. Consequently all points along the borders must be undistorted. ARToolKitPlus introduced corner estimation using Harris corner refinement. Based on pixel-accurate corner positions that were detected during rectangle fitting (see chapter 3.2.2) it uses the Harris corner detection algorithm for sub-pixel accurate refinement. Consequently only these refined coordinates are then undistorted.

[13] http://datamatrix.kaywa.com

3.2.5 Pose Estimation

When markers have been identified in the camera's image and undistorted corner points are available, the final step is to estimate the camera's pose with respect to the markers. ARToolKitPlus can select from various pose estimators, which all use the same basic conecpt: First an initial guess is created that estimates the marker's coarse position and orientation relative to the camera. This first estimate is then refined iteratively until specific quality criteria are met or the maximum number of iterations is reached.

ARToolKitPlus supports tracking of independent markers as well as sets of markers which form a static 3D setup. The latter so called multi-markers have the advantage that the set's pose can be estimated as soon as a single marker is visible, but the pose estimation quality improves as more markers are detected within the image. For doing multi-marker tracking, ARToolKitPlus first estimates the poses of all visible markers independently. The combined pose is then refined by creating a "super-marker" who's "corners" are built from all visible markers of the set. Considering a typical case where 5-10 markers of such a multi-marker are visible, the estimated pose from 20-40 corners is therefore highly accurate and stable.

The original ARToolKit single- and multi-marker pose estimators give good results but suffers from jitter which is inherent to the used algorithm that tends to converge against different minima spontaneously. Although ARToolKit never aimed at accurate pose estimation, the effect of converging against different solutions introduces jitter into the resulting pose which is very noticeable. The "Robust Planar Pose Tracking" (RPP) algorithm [94] by Schweighofer et al. provides improved pose estimation quality with less jitter and improved robustness. RPP takes into account that two local minima exist for the pose estimation error function and specifically deals with these two errors to always find the optimal solution. The RPP algorithm was ported to C++ and added to ARToolKitPlus' set of pose estimators, running well on standard PCs. Unfortunately, due to the high numerical precision requirements of the algorithm, a fixed point port suitable for mobile devices seems currently not feasible.

Recently, the author of this thesis introduced a new pose estimation algorithm that uses non-linear refinement (Gauss-Newton iteration) to ARToolKitPlus (respective to StbTracker). It is implemented in pure fixed point and therefore highly suitable for mobile phones. The algorithm starts by calculating an initial pose from the previously calculated homography (see chapter 3.2.3). Although this initial pose could be used for tracking, it is coarse and jitters heavily. The refinement step uses the algorithm described in appendix 6 (Iterative Estimation Methods) of [43]:

The algorithm aims at refining the 6 parameters (3 for position and 3 for rotation) that make up the pose of the camera. In each iteration step it first calculates the Jacobian matrix (the matrix of all first-order partial derivatives) for the current parameter values. It then uses the following formula to calculate a difference vector:

$$J^T J\, d = -J^T \varepsilon_0$$

In the formula above, J is the Jacobian matrix; J^T is the transpose of the Jacobian; d is the difference vector and ε_0 is the error vector (difference between optimal and reprojected 2D points). Since J^TJ is a symmetric positive-definite matrix, the equation can be efficiently solved using the Cholesky decomposition. The resulting difference vector is then added to the parameters from the previous step. In case the refined pose does not meet the precision requirements yet (estimated by reprojection), another iteration step is executed. See chapter 9.2 in appendix for source code.

3.3 Advanced Features

The following sections list the features that were added to ARToolKitPlus to improve its suitability for mobile devices in general and cell phones specifically. Most of these features were introduced into more than one step of the runtime tracking pipeline, which is why they are listed here separately.

Fixed point

The lack of an FPU is probably the single, most important issue for floating point intensive software on mobile phones and PDAs. To determine the time spent on floating point operations, custom code instrumentation was applied to reveal the most prominent bottlenecks. Tests showed that floating-point usage slowed down especially the pose estimation part of ARToolKit on mobile devices. Replacing the native C float data-type with a system-wide C++ class (emulating all operations with fixed point arithmetic) failed due to strongly varying requirements on precision and numeric range along the pipeline. Instead many functions had to be re-implemented using hand-written fixed point code after determining local range and precision requirements.

Pixel formats

Supporting the native pixel formats of phone cameras is crucial for high performance tracking. Converting to a common format costs too much performance, especially due to the severe memory bandwidth limitations on these devices. Some camera formats already provide data in a format that is ideal for tracking, such as the YUV12 format common on phones. YUV12 stores luminance (Y) at full resolution (8-bits), followed by two chrominance components (UV) at half resolution (effectively 2-bits each). Naturally 8-bit luminance images provide a suitable format for pose tracking from back-and-white markers while minimizing memory footprint. In contrast, formats such as RGB565 require the use of lookup tables for fast format conversion.

Automatic thresholding

In stationary setups lighting can often be controlled to provide well balanced brightness throughout the complete environment of interest. In mobile setups, which can easily span several rooms, floors or even combined indoor/outdoor areas, tracking must adapt to changing lighting conditions. Although many cameras possess auto-gain features today, the final image brightness can still vary heavily which causes severe problems with constant threshold values. Global thresholding, the typical solution for this problem, is computationally too expensive and therefore not suitable for embedded platforms.

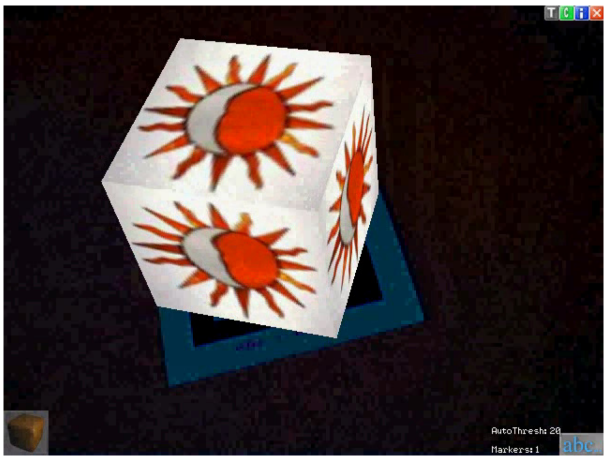


Figure 3.8: Automatic thresholding for tracking in extremely dark environments.

Instead ARToolKitPlus includes a simple, yet very effective heuristic for automatic thresholding (see Figure 3.8) which imposes no measurable performance loss. Instead of looking at the whole image, only the last seen marker is considered. After a marker was found, the median of all extracted marker pixels is calculated and used as a threshold for the next image to process. If the heuristic fails because no marker is found, ARToolKitPlus randomizes the threshold in such a case for every new frame until a new marker is detected. Empirical tests show that after a marker gets lost it takes only a few frames to find a new, working threshold.

Vignetting

Some cameras in mobile phones today exhibit strong vignetting (see left image in Figure 3.9). Thresholding such an image with an image-wide constant value results in an image as can be seen in the middle picture of Figure 3.9. If a marker is close to the border in such an image, it will overlap with the dark areas that were classified as black and the marker would therefore not be detected anymore. To prevent this, ARToolKitPlus provides a simple vignetting compensation feature: The user can specify a radial fall-off from the centre of the image to the corners. This fall-off is specified numerically rather than using an image mask in order to minimize memory bandwidth usage. After activating vignetting compensation even strongly tampered images are thresholded correctly (see right picture in Figure 3.9). Vignetting compensation adds only a minimal performance penalty.

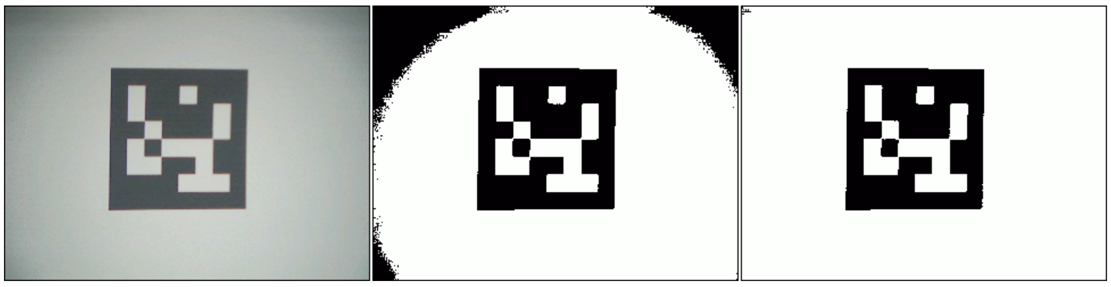


Figure 3.9: Vignetting. Left: original camera image. Middle: constant thresholding. Right: thresholding with vignetting compensation.

Portability

Today's mobile phones run a wide variety of system software. Hence, portability is of high concern. ARToolKitPlus does not include code for camera access or 3D rendering. Its only interfaces for data I/O are a pixel buffer for image input and 4x4 floating point matrices (compatible with the OpenGL matrix format) for tracking results output. It is therefore only limited by the amount of supported input pixel formats. ARToolKitPlus is implemented in pure C++ and is consequently highly portable. Currently Windows XP, Windows CE, Symbian (experimental) and Linux are supported which covers the majority of today's development and target devices. While a Java port would extend the range of supported mobile devices considerably, our informal experiments have shown that the performance of Java on today's mobile phones does not allow sufficiently interactive frame rates.

Custom memory management

Memory is not just a scarce hardware resource on mobile devices but often restricted even further due to deficiencies in mobile operating systems. It is therefore crucial to provide application developers with maximum control over memory de/allocation. Hence all memory management in ARToolKitPlus can be customized by the developer. On most platforms ARToolKitPlus uses the standard memory de/allocation functions per default. On Windows CE ARToolKitPlus' memory manager allocates memory outside the process' memory slot thereby keeping this scarce resource available for DLLs and unmanaged memory allocations. Since ARToolKitPlus' memory footprint is fixed and known at compile-time, the requirements for such a custom memory manager are minimal.

Improved Lens Undistortion Performance

Precise lens undistortion is usually computationally expensive since it requires evaluating non linear functions. ARToolKitPlus uses lookup tables to speed up this process at runtime. Since even the generation of this lookup table can take up to 10 seconds on low-end devices, it can cache the table using the phone’s file storage. When ARToolKitPlus searches for the cached lookup table at startup, it either loads it or automatically creates and stores it for the next startup. Since lens distortion functions are usually continuous, it is adequate to create the lookup table at a smaller resolution than of the camera’s input image and to linearly interpolate values.

3.4 Performance measurements

Over the years, many optimizations were applied to ARToolKitPlus. Besides rewriting major parts of floating point intensive code with fixed point counterparts, ARToolKitPlus makes heavy usage of inline expansion and pre-processor techniques. E.g. the pre-processor is used to generate separate functions for each supported pixel format, which allows switching between those formats at runtime with no performance penalty.

ARToolKitPlus uses lookup tables wherever possible. The pose estimation algorithm intensively uses trigonometric functions that were accelerated with sine and cosine lookup tables. The lens undistortion method of ARToolKitPlus is specified using higher order polynomials which introduce high computational costs at runtime and were therefore replaced by a lookup table too. Matrix fitting requires perspective projection, including (fixed point) divisions which are not implemented in hardware on most ARM CPUs. Replacing these divisions with another lookup table resulted in further significant speedups. Lookup tables can usually not provide the same exact results as algorithmic methods. Special care was taken to always provide enough accuracy so that final results are indistinguishable.

To test ARToolKitPlus' performance for practical applicability, benchmarks on several handheld devices were performed. These tests compare tracking performance with different numbers of visible markers, which is the only criterion that makes a difference in tracking speed. Contrary to expectation, the size of the marker does not influence the tracking speed. The reason for this is that the edge following step generally adds only very little to the overall calculation time.

ARToolKitPlus is primarily CPU bound. So even though all test devices run Windows CE, they represent a good overview of what is currently available on the market. Additionally the benchmarks were run on a PC as a comparison of the processing power on handhelds to a typical PC-based setup. Since several of these devices are available under different brands, list also contains the OEMs' code names. All builds were created using the Microsoft ARM and x86 compilers with full optimization activated (/Ox). Where possible the Intel compiler suite was used to compare different compilers. Benchmarks were performed on the following devices:

- **i-mate SP5** (codename HTC Tornado) is a typical smartphone with a 200 MHz Texas Instruments OMAP850 CPU.
- **HTC MTeoR** (codename HTC Breeze) is a fast smartphone device with a 300MHz Samsung S3C2442 CPU.
- **HTC TyTN** (codename HTC Hermes) is a PocketPC phone with a 400MHz Samsung S3C2442 CPU.
- **Gizmondo** is a mobile gaming console with an nVidia GoForce 4500 3D chip (not used in the benchmark), a built-in camera and a Samsung S3C2440 400 MHz CPU.
- **T-Mobile MDA Pro** (codename HTC Universal) is a high-end PocketPC phone with an Intel XScale PXA270 CPU running at 520MHz.
- **Dell Axim X51v** is a high-end PocketPC PDA with an Intel 2700G 3D chip (not used in the benchmark) and an Intel XScale PXA270 CPU running at 624MHz.
- **Intel 2 GHz Core Duo** represents a standard PC-based setup. On this device ARToolKitPlus was executed with regular floating point code.

Three different scenarios were evaluated: one using single marker tracking and two using multi-marker tracking. In ARToolKit and ARToolKitPlus multi-marker tracking is implemented by first tracking all markers separately, then combining all tracking results and finally optimizing for the complete set. Because of the last step, tracking a multi-marker set with N visible markers is considerably slower than tracking N independent markers.

	Single Marker		Multi Marker (4 markers)		Multi Marker (10 markers)	
Device	**MS**	**Intel**	**MS**	**Intel**	**MS**	**Intel**
i-mate SP5	14.8 ms	13.3 ms	66.4 ms	78.4 ms	234.1 ms	273.8 ms
HTC MTeoR	10.2 ms	n/a	44.6 ms	n/a	153.3 ms	n/a
Gizmondo	8.5 ms	n/a	34.5 ms	n/a	122.7 ms	n/a
HTC TyTN	8.3 ms	n/a	34.9 ms	n/a	128.1 ms	n/a
MDA Pro	6.2 ms	6.0 ms	24.1 ms	29.5 ms	83.4 ms	99.1 ms
Dell X51v	5.4 ms	5.1 ms	20.7 ms	23.25 ms	69.8 ms	81.2 ms
PC	0.55 ms	0.43 ms	6.26 ms	2.77 ms	17.53 ms	8.3 ms

Table 3.1: Benchmarks performed on images with one, four and ten markers. The latter two images were tracked with a multi-marker set of 12 markers of which four and ten were visible.

Due to the aforementioned optimizations, the current version of ARToolKitPlus is roughly 50 times faster on mobile devices than the initial port. Consequently, as can be seen in Table 3.1, single marker tracking represents no major bottleneck on any of the tested devices. It is interesting to notice that with single marker tracking the Intel compiler gains some speed advantage over the Microsoft compiler on those CPUs which can run that code. (Non-Intel CPUs required disabling some optimization flags of the Intel compiler or the generated code would not run).

Multi-marker tracking puts a severe burden on the processing power of today's mobile devices. While tracking a multi-marker set with four visible markers still performs satisfactory on most devices, the cost for tracking ten visible markers is too high for acceptable frame-rates - considering that tracking is only a one small part of a practical application. It is interesting to notice that on all embedded devices the code generated with the Intel compiler performs worse than the code generated with the MS compiler, which is in contrast to the results of single marker tracking. The reason for this behaviour is not revealed yet. Earlier tests with the Intel compiler revealed though that some code, such as the Klimt library (see chapter 4.1) performs generally worse with the Intel than with the MS compiler.

Almost all smartphones and PDAs today use ARM based CPUs. Furthermore, ARToolKitPlus is almost fully CPU bound and hardly memory-bandwidth bound at all. Hence it is not surprising that the tracking performance on the devices in this benchmark increases linear with the CPUs' clock rates. As can be seen in Figure 3.10, all devices process 31.12 (+/- 1.76) frames per second at a normalized speed of 100 MHz.

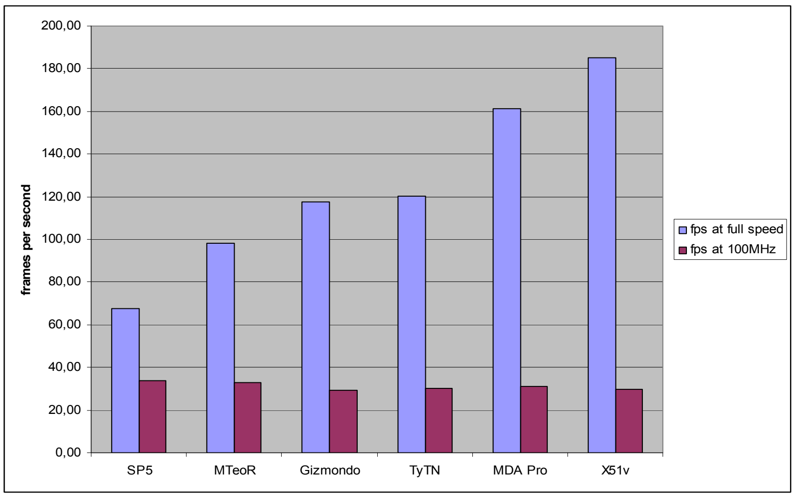


Figure 3.10: Frames per second for single marker tracking on embedded devices.

3.5 Discussion

This chapter presented a solution for tracking of fiducial markers on mobile phones and similar devices. Tracking is a basic building block of every Augmented Reality application. ARToolKitPlus is therefore an important contribution for any AR system running on mobile phones, since it allows regular, unmodified phones to estimate their pose in respect to objects of interest. Taking the limited requirements on performance and accuracy into account, ARToolKitPlus on the phone performs comparable to tracking systems on the PC, thereby contributing to fulfilling argument H1 of the hypothesis in chapter 1.3.

Studies in this thesis (see Chapter 7) show that deployment of fiducial markers is reasonably accepted. Yet we see fiducial marker tracking only as an intermediate step to the long term goal of augmented everything and everywhere, consequently requiring natural feature tracking. Today, most PC-based AR applications today also use fiducial markers due to their unparalleled tracking robustness and performance.

An advantage of fiducial marker tracking is the fast and easy deployment which makes it more practical than natural feature tracking in certain applications, as presented in chapter 7.5.2 on the Signpost 2007 application. For robust location estimation in large areas, natural feature tracking inherently requires models of the real environment ("model based tracking"), which are far more work intensive to create than deploying fiducial markers. Deployment of commercial 6DOF tracking systems is usually not affordable for wide areas too. Hence, also hypothesis H2 benefits from the work presented in this chapter, since it clearly supports building larger AR applications.

Even if natural feature tracking is available, in some application areas, such as advertisement, markers can serve as practical hints for users to activate an AR application on their phone and aim it towards the marker. We therefore believe that a next step will be the combined use of fiducial and natural marker tracking, combining the strengths of both techniques.

Chapter 4

Rendering

Although AR is not exclusively focusing on visuals, most research focuses on graphics augmentations, making rendering the second important aspect after tracking, which was described in the previous chapter. This chapter first outlines graphics concepts and then presents solutions for 2D and 3D graphics developed in the course of this thesis.

Computer graphics researchers are accustomed to working with standardized low-level APIs such as OpenGL. On small mobile platforms like PDAs and cell phones there exists no built-in 3D graphics subsystem as of yet, and no widespread general purpose commercial solutions currently exist. Although OpenGL ES is on its way to become a solid base for 3D on mobile devices, most existing portable 3D applications rely on OpenGL rather than OpenGL ES. Furthermore the AR framework developed in this thesis, generally strives for a higher level of abstraction than OpenGL and similar libraries provide.

An alternative solution to rendering natively on the handheld device is remote rendering. In this scenario, a powerful server takes over the image generation task and sends final images to the mobile device. For example, Lamberti et al. successfully streamed mobile graphics using an MPEG video feed [59], assuming a fast network connection. Such an approach is probably the only viable solution when graphics that go far beyond the client's capabilities have to be displayed. At the downside this method scales badly with the amount of clients and becomes unpractical in low bandwidth or low quality network situations. The handheld AR project therefore concentrated on native rendering only.

This chapter describes graphics solutions that the author either developed for or ported to mobile platforms. Some of them, such as Klimt have mostly historical value anymore since it have been superseded with the wide availability of OpenGL ES implementations.

4.1 Immediate Mode Rendering

In immediate mode rendering, also known as low level rendering, the user has full control over the rendering process. Graphical objects are described on a vertex level and retransmitted every frame. Immediate mode rendering is the basis and lowest level for every current rendering system. The most common immediate mode 3D APIs are OpenGL and Direct3D, of which the latter is only available on Windows platforms and therefore of minor applicability for our purposes. Hence, OpenGL is traditionally strongly represented in research and professional applications. It was therefore important for the handheld AR project to have OpenGL or OpenGL ES implementations available as the basis for all higher level 3D rendering libraries. Yet, since some Windows CE devices with hardware 3D acceleration come with Direct3D drivers only, support for this API was recently added in StbES too.

Figure 4.1 shows the OpenGL render pipeline as defined by the OpenGL 1.5 specification. Application data enters from the left. Programmers can either directly manipulate pixel data using pixel operations or pass in vertex data. All aforementioned operations can be recorded and played back using display lists. In the rasterization stage, primitives are converted to frame buffer addresses called fragment, which are then manipulated using per-fragment operations and finally written into the frame buffer.

Since OpenGL implements the most complete pipeline among all widely used immediate mode graphics toolkits today, it is used to compare it against the pipelines of OpenGL ES, Direct3D Mobile, Klimt and KlimtES in the next chapters.

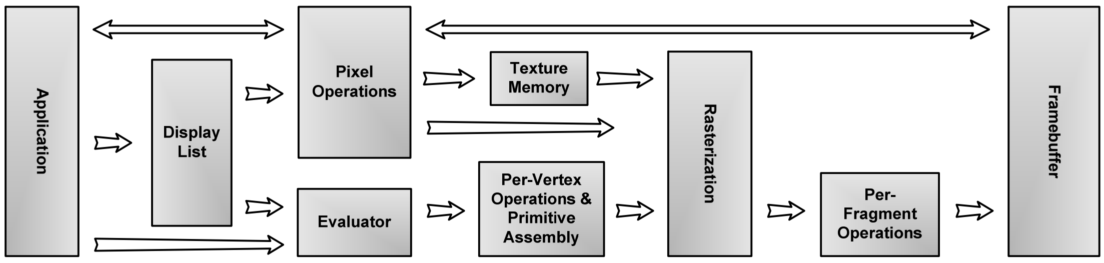


Figure 4.1: The OpenGL render pipeline.

4.1.1 OpenGL ES

OpenGL ES was designed by the Khronos[14] group as a light-weight 3D API for mobile devices. It is a powerful low-level API that is very similar to that of OpenGL and provides most of OpenGL's functionality. Khronos used the chance to clean up the OpenGL API and solve many long outstanding issues such as window bindings.

It is available in hardware and/or software on all mobile key platforms. Hybrid Graphics, a commercial vendor created a software implementation called Rasteroid[15] that is free for non-commercial use. Vincent[16] is a free open source implementation. Both libraries use run-time code generation for the pixel pipeline: Instead of including a fixed rasterizer that would have to efficiently handle all possible pixel operation combinations, these libraries create machine code at runtime that optimally implements the currently specified combination of raster operations. Since this is an extremely low level approach, supporting different CPU architectures poses a considerable amount of work. Unlike Rasteroid, Vincent therefore only supports ARM CPUs and consequently does not run on PCs. For development purposes a reference implementation of OpenGL ES that wraps an underlying OpenGL implementation is available by Khronos.

Reduced Feature Set

A major design decision for OpenGL ES is that it provides no functionality that can not be mapped directly to the underlying implementation in order to simply driver development and reduces the driver's code size. Furthermore all redundancy of the OpenGL API was removed (see Table 4.1). Consequently there is no glBegin/glEnd since programmers are advised to use the more efficient vertex array or vertex buffer functions instead. No major 3D implementation today has native support for primitive types other than points, lines and triangles. Hence, OpenGL ES does not support quads or n-sided polygons. Instead of requiring the driver to triangulate these primitive types this job is left to the programmer.

Many rarely used high level functions such as the GLU functions, evaluators, picking or display lists have been removed too. Since these functions are not implemented in hardware anyway, it was decided to leave the implementation to the application programmer when needed. The overall effect of these streamlining efforts is a heavily simplified graphics API as well as smaller, easier to implement and more robust drivers. Looking at Figure 4.1, OpenGL ES misses the display list, evaluator

[14] http://www.khronos.org
[15] http://www.hybrid.fi
[16] http://sourceforge.net/projects/ogl-es

and most of the pixel operation stages. As a consequence of the reduced feature set, a full OpenGL ES 1.x software implementation can easily fit into a binary of a few hundred kilobytes size.

	OpenGL	**OpenGL ES 1.x**
glBegin/glEnd	yes	no, except SC profile
Primitive Types	all	no quads & polygons
Data Types	float, double, int, etc.	float, fixed
glDraw/Read Pixels	yes	glReadPixels only
Textures	1D,2D,3D,cube	2D
Stencil	yes	optional
Window Bindings	WGL, GLX, etc.	EGL

Table 4.1: Comparing OpenGL versus OpenGL ES.

Fixed Point

OpenGL ES targets primarily embedded systems, which usually do not have floating point units. On these CPUs floating-point usage has to be emulated in software, which is roughly 50 times slower than using native data types. Hence, OpenGL ES adds fixed-point as a new data type over OpenGL. Floating point uses separate representations for exponent and precession, while fixed-point stores both in a single integral value. Fixed point can therefore be based on regular integral values and most arithmetic operations can therefore be performed in hardware. While any integral type can be used for representing fixed point data, the usage of two's complement is most common, since it is the native integral data type of many modern CPU architectures, including ARM and x86.

OpenGL ES defines a 15.16 fixed point format for its API, which uses 15 bits before and 16 bits after the radix point. Consequently a numeric range of -32768 to +32767 can be expressed at a precision of 1/64k or 0,0000153. Despite its advantage of allowing effectively implementation without floating point units, fixed point often creates problems with the small numeric range resulting in numeric overflows.

Window Bindings

OpenGL ES defines a new window binding called EGL that mostly solves the platform dependency issues of the OpenGL windows bindings such as WGL, GLX or AGL, which are completely platform specific.

EGL specifies platform independent functions and handles for most operations. Only minimal set of handles such as for device contexts or bitmaps is platform dependent. Due to platform specific type definitions, these types are automatically resolved at compile time though, posing no extra work for the developer.

EGL knows three different kinds or render targets. An implementation is free to implement only a subset of them. Window surfaces are usually stored in video memory, while PixMap surfaces are always stored in system memory. Furthermore PBuffer surfaces allow off-screen rendering. It is due to the renderer to decide whether to put them in system or video memory. A hardware implementation will usually not support PixMap surfaces since this would require a full software based renderer implementation too. The main advantage of using PixMap surfaces is the access to the pixel data, which not possible using Window surfaces.

4.1.2 Direct3D Mobile

Direct3D Mobile was introduced by Microsoft with Windows CE 5.0 and is therefore part of Windows Mobile since version 5 (also known as 2005). Its API is COM[17] (Component Object Model) based, a Microsoft interface model, which is also the basis for ActiveX. Consequently, the API is object oriented rather than procedural such as the C-API of OpenGL ES.

Similar to OpenGL ES being a reduced version of OpenGL, Direct3D Mobile is a subset of Direct3D. Direct3D Mobile is based on Direct3D 8, but it also incorporates a few elements and behaviors of Direct3D 9. The differences become obvious when comparing the number of classes of both APIs : While Direct3D contains 16 classes, Direct3D Mobile has only 8 classes. Direct3D Mobile misses support for shaders, texture formats other than 2D (such as 1D, cube and volume textures) and stage block (for applying many render state changes at once). Most functionality of Direct3D Mobile is gathered in the IDirect3DMobileDevice “god-class”.

Although the APIs of Direct3D and OpenGL ES 1.x are very different the underlying concepts and the feature set are very close since they target exactly the same class of hardware. Like OpenGL ES, Direct3D Mobile does not include the display list, evaluator and most of the pixel operation stages from Figure 4.1.

Being even closer to metal than OpenGL ES, Direct3D Mobile does not allow specifying vertex data via simple array pointers, but requires specifying a Flexible Vertex Format (FVF), that defines the structure of vertex data passed to the graphics driver. Unlike OpenGL ES, vertex data must always be interleaved, packing all

[17] http://www.microsoft.com/com

components of a vertex such as position, normal, etc. closely together, which is closer to how a GPU processes these elements than OpenGL ES does.

4.1.3 Immediate mode rendering with Klimt

When the Handheld AR project started in 2002, no suitable 3D rendering library was available to fulfill the requirements of an Augmented Reality application. There have been previous attempts to create libraries similar to OpenGL, such as PocketGL and TinyGL, but these projects were either tailored to the specific needs of computer games or have not reached sufficient maturity. The author of this thesis therefore chose to implement a custom software renderer, called Klimt[18], as an open-source 3D library targeted for PDAs and mobile phones.

	Klimt	OpenGL ES 1.x	Direct3D Mobile
glBegin/glEnd concept	yes	no, except SC profile	no
Display Lists	yes	no	No
Primitive Types	all	no quads & polygons	no quads & polygons
Data Types	float, double, int, etc.	float, fixed	float, fixed
Frame buffer access	yes	reading only	device specific
Textures	1D,2D,3D,cube	2D	2D
Stencil	yes	optional	yes
Window Bindings	EGL, WGL, GLX, etc.	EGL	Windows

Table 4.2: Comparing feature set of Klimt, OpenGL ES and Direct3D Mobile.

In contrast to these previously existing render libraries for mobile phones or PDAs, Klimt is flexible enough to fulfil the requirements of an AR application such as off-axis camera support required for tracking (see Figure 4.2). Its API is very similar to that of OpenGL and OpenGL ES, although it would not pass official conformance tests. For example, Klimt can be used as the low-level graphics toolkit for Coin[19], an OpenInventor-compatible scene-graph rendering library which was used in the Handheld AR project at that time (see more about Coin in chapter 4.2.1).

Unlike OpenGL ES, Klimt has a full display list stage that can record most of the GL render commands. Since rarely required it does not implement evaluators or most of the pixel operations. Yet, being a pure software renderer, it provides full access to

[18] http://studierstube.org/klimt
[19] http://www.coin3d.org

the frame buffer. In this sense, the pixel operation stage from Figure 4.1, is mainly a pass-through in both directions.

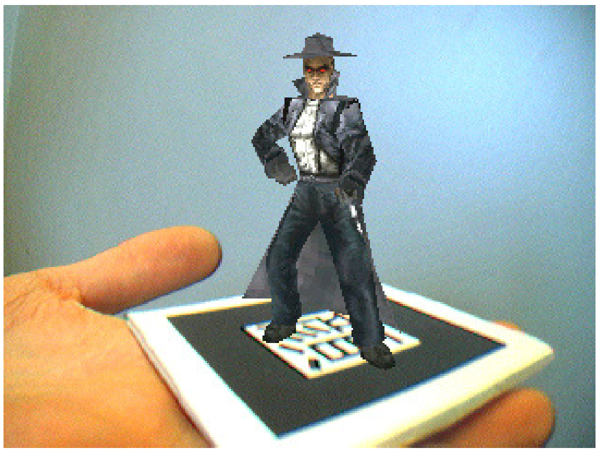


Figure 4.2: Textured avatar "Caleb" rendered on top of an ARToolKit fiducial marker using Klimt.

Software Design

Klimt is implemented as a collection of C++ classes with a C wrapper for OpenGL API compatibility. As a means to optimize runtime performance Klimt makes heavy use of templates and inline functions. Klimt can be configured to use any kind of native (e.g. 32 bit integer…) data type for internal processing. Because of the lack of floating-point units in current mobile devices, Klimt by default performs all internal operations using fixed-point math. Klimt is built from two main components, the render context and the rasterizer (see execution layer in Figure 4.3).

The rasterizer is purely software-based and implements the pixel pipeline. It renders RGB565 only, which is the native pixel format of most of modern mobile phones and PDAs. Nevertheless, it provides most of the features supported by OpenGL and OpenGL ES, such as gouraud-shading, texturing, transparency, alpha blending, fog, z-buffering or any combination of them. Although only this rasterizer has been implemented it would be easy to create other implementations with different pixel formats or even hardware support.

The Context component keeps track of the current render context's state and implements the complete vertex pipeline including transformation, lighting and clipping, as well as important OpenGL concepts such as attribute stacks, display lists and vertex arrays.

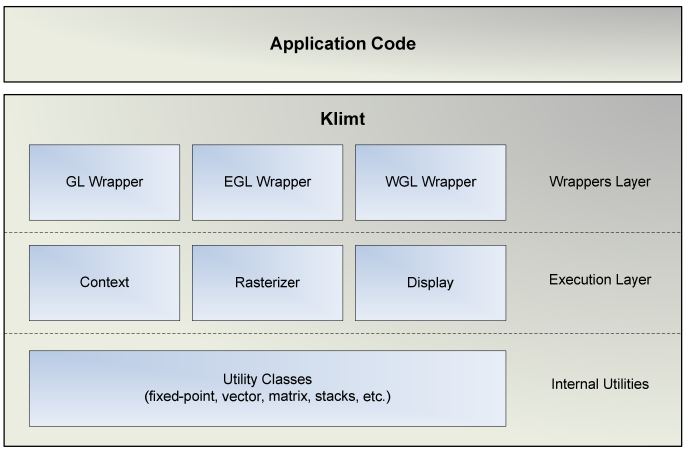


Figure 4.3: Klimt software architecture.

A general problem when creating a high performance implementation for a standard such as OpenGL with a highly configurable render path is the development of fast per-pixel routines. OpenGL allows modifying the per-pixel operations in many ways through texturing, shading, blending, etc. Additionally to specifying how the output color is created OpenGL allows setting a variety of visibility tests such as z-buffering, clipping, alpha tests, stencil tests, etc. A naïve implementation would check on a per-pixel basis which operations to apply. Since this chain of commands contains about 10 operations, such an implementation would be extremely slow.

An early version of Klimt used compile-time code generation to generate the most important combinations of these pixel operations. A Python script generated 735 combinations of scanline function setups which where then expanded using the compiler's pre-processor. For all other operations a slow general purpose implementation was available as fall back. While this resulted in a very fast rasterizer, the resulting binary (DLL) was between 1 and 2 megabytes in size (depending on compiler settings), which created a serious problem for distributing and deploying the software onto mobile devices.

A later, revised version performed these operations not per pixel but per scanline. A triangle's scanline was first shaded, then textured, etc. To make optimal use of early z-tests, all visibility operations are performed first for the whole scanline and the results are stored in a run-length encoded list that allows skipping these pixels quickly for other pixel operations. Since a single scanline fits easily into the processor's cache there is only minimal overhead over the previous implementation. The resulting binary is only a few hundred kilobytes large and therefore fits the requirements of mobile devices.

Since Klimt is internally implemented as a set of C++ classes it requires wrappers to comply with the standard C APIs (see wrappers layer in Figure 4.3). Klimt includes wrappers for OpenGL and OpenGL ES as well as WGL and EGL window bindings.

4.1.4 Immediate mode rendering with KlimtES

With the advent of mobile phones and PDAs with hardware 3D support a pure software-based Klimt implementation is no longer a competitive solution. While Klimt can be extended to support hardware accelerated rasterization these devices usually come with a complete OpenGL ES implementation. Instead of just using an underlying OpenGL ES support for rasterization purposes only it makes more sense to take advantage of the already implemented vertex stage too. Although OpenGL ES can be used directly a lot of existing code relies on OpenGL instead. The author of this thesis therefore created a wrapper that restores functionally missing in OpenGL ES.

KlimtES exposes almost the same OpenGL compatible API as Klimt and therefore allows to run many existing software packages that rely on OpenGL features without modification. At startup KlimtES searches for an existing OpenGL ES DLL. It supports the common as well as common-lite profiles of OpenGL ES. In the latter case, which does not support floating point, the data type conversion is performed by KlimtES.

KlimtES implements the glBegin/glEnd clause by internally storing the specified vertices in vertex arrays that are handed over to the OpenGL ES driver when the arrays are full or the primitive type is changed. Furthermore KlimtES allows using quads, quad-strips and polygons by internally triangulating these into vertex arrays. The most crucial missing feature is floating point support though since OpenGL does not include fixed point functions at all. While it is simple to convert from floating point to fixed point these operations are very slow and therefore degrade performance seriously. It is therefore advised to rewrite higher level applications or libraries to directly use fixed point as was done with Coin ES (see chapter 4.2). Other OpenGL features provided by KlimtES include support for WGL window binding, integer index types and state queries missing in OpenGL ES.

Like Klimt, KlimtES is implemented in C++ and therefore requires C-wrappers to conform to the OpenGL and OpenGL ES APIs. From an application point of view, KlimtES completely replaces the OpenGL and OpenGL ES implementations. It therefore wraps (pass-through) even those functions that are not modified in their behavior.

4.1.5 Software vs. Hardware Rendering

While both hardware and software implementations of OpenGL ES and Direct3D mobile provide the same feature set, their performance characteristics are highly different. Besides pure software and hardware implementations, mixed approaches are common. Similar to the beginning of hardware 3D support on desktop PCs in the 1990s, many mobile designs today only implement the pixel stage in hardware, running the vertex stage in the driver and therefore on the CPU.

Hardware rendering obviously has the advantage of providing much more raw processing power than software rendering, which usually removes the need to carefully reduce the vertex count of 3D meshes. Furthermore texturing is typically as fast as gouraud shading, making texturing a free option to select. Unfortunately, these GPUs, which are primarily targeting games, are not fast in transferring textures, which poses a severe problem for efficient rendering of the video background.

While pure hardware implementations are usually well balanced, smaller designs often implement only the rasterization stage in hardware. Pure software renderers typically suffer from bottlenecks in the pixel pipeline, but mixed designs are more often vertex limited. The Intel 2700G as well as the nVidia Goforce Go 4500 GPUs are typical examples for this category.

	Pure Software	**Mixed S/W-H/W**	**Pure Hardware**
Vertex Stage	Software	Software	Hardware
Pixel Stage	Software	Hardware	Hardware
Typical Limits	Pixels	Vertices	-
Framebuffer Access	Yes	No	No
Fast Texturing	No.	Yes	Yes

Table 4.3: Comparing software, hardware and mixed implementations.

The main bottleneck of pure software renderers is typically the pixel pipeline, especially when intensively making use of texturing. As a unique advantage, these implementations usually allow direct frame buffer access, which enables copying the video background directly into frame buffer, thereby by-passing slow texture mapping routines. Furthermore, since the frame buffer of these designs is always in system memory, this copy operation is extremely fast. Under specific circumstances such as when rendering simple 3D graphics in Studierstube ES on the Gizmondo (powered by a Goforce Go 4500 GPU), a pure software OpenGL ES implementation can outperform the built-in graphics chip.

A high level toolkit such as Studierstube ES is therefore required to implement multiple render paths for making optimal use of the strengths of each kind of renderer. Studierstube ES contains different routines for 2D graphics operations, such as drawing video background or buttons. When running on hardware 3D, direct frame buffer access is not possible, and StbES therefore relies on texturing to draw bitmaps onto the screen. In the software rendering case though, these bitmaps are copied directly into the frame buffer.

4.2 Retained mode rendering

Retained mode rendering describes a high level rendering concept on top of the lower level immediate mode presented in the previous chapters. In retained mode, objects and their relationships are described as a whole rather than on per-vertex basis. Retained mode introduces the concept of a scene that is made up of objects which together form a virtual environment. While immediate mode requires procedural programming, retained mode has a more descriptive flavor: instead of telling the graphics subsystem how to render an image, users specify objects and leave the actual rendering task to the rendering middleware, which is free to apply suitable rendering strategies and optimizations.

Today, scene-graphs are the most common concept for retained mode rendering and especially popular in research and for prototyping. A scene-graph structures objects hierarchically and describes their interrelations as well as properties such as materials, rendering styles or priorities.

4.2.1 Scene-graph Rendering with Coin ES

Coin3D, developed by Systems in Motion[20], is a high-level scene-graph rendering library that implements the OpenInventor API [99]. Instead of having to work directly with low level APIs such as OpenGL or OpenGL ES, Coin3D allows application developers to concentrate on high level tasks such as application design and data flow. Among the large number of scene-graph libraries, OpenInventor is probably the most flexible one and therefore ideally suited for prototyping graphical data-driven applications. On the other side this flexibility comes with the price of

[20] http://www.sim.no

lower performance and a non-deterministic real-time behavior. The Studierstube team around Prof. Schmalstieg has a long history of using OpenInventor and Coin3D which made it a natural choice for the handheld AR project too.

Coin3D integrates a vast number of features. Its roughly 500 C++ classes provide support for loading various file formats such as VRML and 3ds as well as its native Inventor (.iv) file format. Furthermore it has support for 3D audio and modern texturing functions such as 3D textures, multi-textures, huge textures and bump-mapping. It can output not only to video memory but to Postscript as well. Coin3D can read a large number of texture sources including JPEG, PNG, GIF, TGA and many more. Naturally this packet carries a heavy weight - VRML support alone increases the overall size of Coin3D considerably. While this is not an issue on a PC, it is too big for embedded platforms such as mobile phones or PDAs.

It was therefore mandatory to remove unnecessary functionality and to optimize the performance for handheld devices as well. Besides minor modifications that make Coin3D run with Klimt and KlimtES functionality was removed that is not required for handheld AR such as draggers, manipulators, VRML and Postscript support. Instead of the heavy weight window bindings that provide highly comfortable viewers, a custom lean window binding was developed that only runs in full screen and has minimal mouse and keyboard support. Further removing support for unnecessary texture source formats resulted in a binary with roughly half of the original size. Yet, a binary library with integrated support for loading JPEGs, PNGs and many other image formats is still almost 3 Mbytes in size.

Although Coin3D runs on top of Klimt and KlimtES with only minimal modifications, its performance was still unsatisfactory. Targeting high-end applications on powerful workstations and VR systems, Coin3D's memory consumption is enormous for mobile devices. Therefore a custom memory manager was added to overcome virtual memory restrictions under Windows CE[21]. Coin3D naturally uses floating point as its internal data format. While Klimt can automatically convert this data to fixed point, doing this on a per-frame basis for mostly static models is a huge waste of processing power. Hence, the Coin3D primitive cache was extended to support fixed point natively and thereby only have to convert data once.

A general problem of Coin3D is the large overhead of traversing the scene-graph. Due to the way OpenInventor handles internal changes to the graph structure and fields, the complete scene-graph can be traversed many times per frame. Large and complex scene-graphs therefore pose a serious performance problem for low end platforms. It is therefore of utter importance that the application designer carefully constructs the scene-graph and optimizes it for minimal size. While the binary file

[21] http://msdn2.microsoft.com/en-us/library/ms836325.aspx

size and memory consumptions are still enormous for embedded platforms, the resulting rendering performance of a well designed scene-graph is close to direct rendering using Klimt or KlimtES.

4.2.2 Scene-graph rendering with StbSG

While Coin3D is a powerful library and very suitable for developing AR applications on the PC, it is still too big to be practical for mobile devices – even after our efforts of reducing its feature set and size. Larger applications such as Studierstube 4 on the PDA continued to run out of memory. Furthermore it is unrealistic to ask end-users to install applications of many Megabytes in size of their phones.

Hence, the design decision was made to not rely on components anymore that would prevent a practical deployment on end-user devices. Unfortunately there are very few high level rendering libraries that support OpenGL ES directly and that were specifically designed and implemented for mobile devices. A notable exception is the Mobex3D[22] game engine, which is available for Windows Mobile, Windows XP and MacOS. Mobex3D has many advanced features such as particle systems, skeletal animation and exporters for 3dsMAX, but it is expensive for commercial use. The free version is very limited and allows only scripting, but no native C++ development.

Software Design

Studierstube ES therefore comes with its own custom scene-graph library called StbSG (Studierstube Scene Graph) and optimized it for the specific requirements of handheld Augmented Reality applications. StbSG is an integral part of Studierstube ES rather than a separate library. Its design is similar to that of OpenInventor which makes it easy for experienced Studierstube developers to migrate to Studierstube ES. StbSG works on top of OpenGL ES and Direct3D Mobile and does therefore not require an external wrapper such as KlimtES. It is optimized to work as a thin layer between the application and OpenGL ES or Direct3D Mobile. Consequently it does not expose graphical features that can not be mapped to the underlying graphics toolkit implementation such as quad or polygon primitives.

StbSG has most of the typical features of scene-graph libraries. Before a node can be used in the scene-graph it has to be registered to the scene-graph database that stores the reflection information of all nodes and allows searching scene-graphs by node types and names. This registration process is especially important for the

[22] http://www.mobex3d.com

loading and saving mechanisms of StbSG. For all internal nodes this happens automatically during startup. For custom nodes, developers have to call the corresponding class methods manually. This mechanism allows developers to extend StbSG with arbitrary new nodes that can even be loaded and instantiated from different binary files at runtime.

Scene-graph Nodes

A scene-graph can hold an arbitrary number of cameras that are automatically activated, when traversed. StbSG comes with implementations for orthographic, perspective and general purpose camera models. The latter one can be freely configured providing a projecting matrix and is therefore suitable for accurately overlaying video backgrounds using off-axis projection.

Geometry can be specified in various ways. Using SgGeometry nodes provides most flexibility since it only (optionally) stores indices and is not responsible for providing per vertex data. These data sets are typically defined by other nodes such as SgGeometryVertices, SgGeometryNormals, SgGeometryColors or SgGeometryTexCoords. As an advantage of splitting up the data into different nodes, data dependency is reduced. Consequently other nodes can also provide per-vertex data such as the SgAnimatedGeometryVertices node, which defines animated vertices.

The geometry-related nodes described above all store their data in text form in the XML scene-file. While this allows maximum flexibility, it increases file sizes and loading times. StbSG therefore also contains a SgStaticMesh node that loads its data from an external binary file. The node supports storing and loading data at full precision in fixed-point and floating-point as well as in reduced precision, which results in a file size reduction of roughly 60% at no noticeable quality loss.

The third type of geometry node is SgProgressiveMesh which allows progressive streaming and rendering of polygonal data. Objects can be streamed via network and rendered as soon as a few triangles are available. When more data is accessible, the mesh is incrementally refined. Furthermore, the mesh can reduced for usage as fine grained continuous level of detail (LOD).

Often animations on a per-vertex level are not required. Instead complete objects can be animated by modifying their position and rotation via keyframe sampling using the SgPoseAnimator node.

The appearance of objects can be influenced via SgTexture nodes that define single- or multi-texturing. SgDirectionalLight and SgMaterial nodes specify how light influences objects. SgDrawingStyle can modify front facing of triangles, depth mode, stenciling, as well as depth buffering and blending.

SgRenderAction allows introducing more procedural approaches by clearing the render target's depth, color or stencil buffer, thereby enabling multi-pass rendering. For more specialized render commands, developers can use SgCallback nodes that invoke client code when traversed.

At the moment of writing this thesis, StbSG contains 37 node classes, which are listed by category in Table 4.4.

	Nodes
Camera	SgCamera, SgMatrixCamera, SgOrthographicCamera, SgPerspectiveCamera
Geometry	SgCube, SgGeometry, SgGeometryColors, SgGeometryNormals, SgGeometryTexCoords, SgGeometryVertices, SgLineSet, SgProgressiveMesh, SgStaticMesh
Transformation	SgMatrixTransform, SgPoseAnimator, SgTransform, SgTransformSeparator
Appearance	SgDirectionalLight, SgDrawingStyle, SgLight, SgMaterial, SgProgressiveTexture, SgTexture, SgTextureSeparator
Animation	SgAnimatedGeometryVertices, SgAnimator
State Specific	SgGLStateModifier, SgLightSeparator, SgRenderAction, SgRenderOnce
AR Specific	SgBackground
Non-Graphical	SgCallback., SgFileNode, SgMultiSwitch, SgNode, SgScene, SgSwitch

Table 4.4: List of all nodes in StbSG.

Low Level Graphics Toolkit Abstraction

For optimal platform support, StbSG supports OpenGL ES as well as Direct3D Mobile for those devices that do not come with OpenGL ES drivers. The abstraction of the underlying rendering toolkit is implemented using the two classes Renderer and GeometryBuffer, which are implemented for both toolkits. Stacks for state variables and querying values of state variables are important operations for scene-graph libaries. Since neither OpenGL ES nor Direct3D support stacks or queries for most state variables such blending, depth sorting or matrices, all stacks are implemented in the Renderer class.

GeometryBuffer abstracts passing geometrical data to the graphics toolkit. When multiple options are available it automatically decides whether to store vertex data in system or video memory. The two implementations of GeometryBuffer effectively hide the differences between specifying vertex buffers in OpenGL ES and Direct3D.

Especially under Direct3D, the manual definition of the data structures for the Flexible Vertex Format (FWF) is cumbersome and error prone.

Developers can use separator nodes to limit the effect of specific nodes such as transformations, lighting or textures to certain sub-trees in the graph. After a sub-graph has been traversed, the separator node undoes all changes of the GL states that it protects.

Reflection

Similar to OpenInventor, StbSG uses fields to store attributes. Before a field can be used, its type and valid values must be defined. StbSG includes fields for many types such as integer, float, vector or matrix types, as well as enumerations. A field's type and values are defined using c-macros when the field's owner (typically a node) is registered.

StbSG make use of reflection to allow querying any node's list of fields. Every field has a name, a type and a default value. A system of C-macros automatically implements a complete type system including reflection of all fields a node owns Figure 4.4. Fields of compatible types can be connected via field connections to let data flow across the graph or to connect data streams from outside into the graph. For example, Studierstube ES uses this mechanism to stream live tracking data from the tracking subsystem directly into transformation nodes of the scene-graph.

```
REGISTER_FIELD_TYPE(FieldTypeInt, stencilRef, 1);

REGISTER_FIELD_TYPE(FieldTypeVec3, translation, Vec3(0,0,0));

REGISTER_FIELD_TYPE(FieldTypeVec4, ambient, Color4(0.2f, 0.2f, 0.2f, 1.0f));

REGISTER_FIELD_TYPE(FieldTypeTriState, colorWrite, TRI_UNDECIDED);

REGISTER_FIELD_TYPE_ENUM(frontMode, FRONT_NO_CHANGE,
    ENUMS_3(FRONT_NO_CHANGE,FRONT_CW,FRONT_CCW));
```

Figure 4.4: Example field registrations.

Figure 4.4 presents several examples for registering fields. The first line registers an integer-valued field with name stencilRef and a default value of 1. The second and third lines of code register a 3D vector and a color, while the fourth line registers a tri-state field. Tri-states are important for flags that can either enable, disable or keep targets unchanged. Finally, the last line shows how an enumeration value with three possible values is registered.

Field connections are always updated before graph traversal. A dirty flag indicates if a field has changed, which is used for caching such as for transformation matrices: Only when the transform's position, rotation or scale values have changed the matrix is recomputed.

StbSG uses XML for saving and loading of scene-graphs. Nodes are mapped to XML elements and fields are mapped to XML attributes. Pure data fields such as vertex arrays are mapped to XML text section. Every node must implement methods for creation as well as for loading from and saving to XML files. Most work for loading and saving is performed automatically. Figure 4.5 shows a code snippet from the SgTransform.cpp file that creates a transform node (if not done yet from the super class factory method) and then reads three fields from the element's list of attributes. Due to StbSG's reflection capabilities a single call to readAttribute() is enough to decide the field's type and its default value in case the attribute is stored in the element. Finally the factory function calls the factory function of the node's base class in order to read more attributes.

```
SgNode*
SgTransformFactory::readNode(TiXmlElement* nElement, StringVector& nAttrToIgnore, SgNode* nNode)
{
        if(nNode && !nNode->isOfType(SgTransform::getClassType()))
                return NULL;

        SgTransform* transform = nNode ?  reinterpret_cast<SgTransform*>(nNode) :
                                          SgTransform::create();

        readAttribute(nElement, transform->translation);
        readAttribute(nElement, transform->rotation);
        readAttribute(nElement, transform->scaleFactor);

        return callBaseReadNode<SgTransform>(nElement, nAttrToIgnore, transform);
}
```

Figure 4.5: Code sample to load an SgTransform node from an XML file.

Field connections reference the name of target fields which must therefore be unique across a scene. StbSG uses a two pass approach for loading scene-graph files: In the first pass the complete graph is instantiated, while in the second pass all field connections are resolved.

Nodes can be added to the scene graph multiple times by adding it once and referencing it via its name. Since every attribute has a well known default value, only fields with non-default values have to be stored in the XML file. See chapter 9.1 for example scene-graph XML files.

To keep the file size and memory footprint at a minimum, StbSG itself has no capabilities to import from other data sources than its own XML file format. Studierstube ES includes with a VRML reading tool on the PC though that imports VRML files and creates a corresponding StbSG instance. StbSG's regular file saving capability is then used to store the converted scene. While VRML is an old and mostly outdated format, it provides all features required to create content suitable for mobile phones, including support for per-vertex as well per-object animations. Unlike many other file formats, high quality VRML exporters for all major graphics editors exist.

StbSG can clearly not compete with large established scene-graph libraries such as Coin3D, OpenSceneGraph or OpenSG. Yet, neither of these libraries is optimized for running on mobile phones. While for some of these libraries ports to OpenGL ES exist, these ports naturally loose a lot of performance due to inherent design differences (e.g. floating point vs. fixed point). Furthermore, none of these libraries runs on top of Direct3D Mobile, which turned out to be an important aspect for StbSG.

On the other side there are high quality, commercial 3D libraries such as EdgeLib and Mobex3D that specifically target mobile phones. These libraries provide high performance and many more features related to professional content creation such as skeletal animation. Yet, their closed source nature and high licensing costs reduce their suitability for deployment in academic as well as commercial environments at the same time.

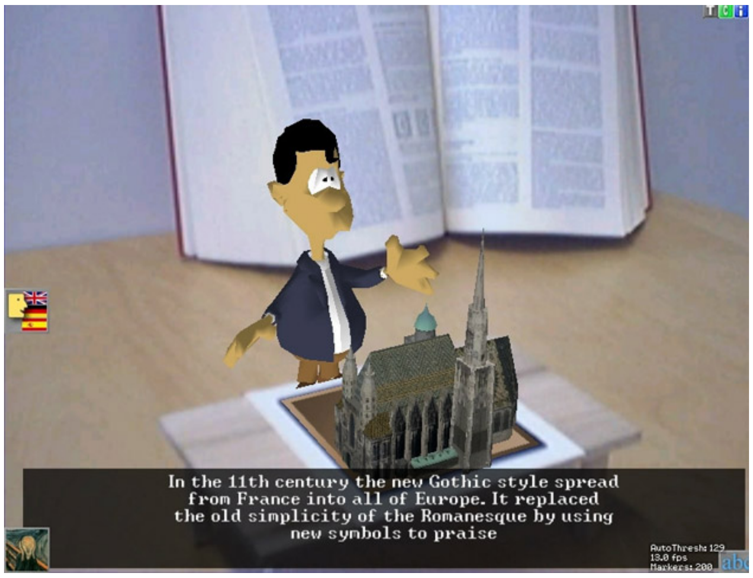
4.3 3D Animations using the FPK library

The lack of processing and graphics power on mobile devices prevents the use of traditional character animation techniques. For example most existing animation libraries make heavy use of floating-point operations which are not available in hardware on today's PDAs. Many of today’s state-of-the-art animation techniques such as skeletal animations or vertex skinning are not suitable for embedded devices due to the lack of processing power and missing dedicated hardware support. While these techniques have been used for a long time in offline rendering, only their recent support by graphics hardware introduced them to real-time graphics on desktop computer.

A computationally less expensive method for animating polygonal meshes is keyframe-based animation. Other than the previously mentioned approaches that deform a single mesh on-the-fly with respect to an underlying bone-structure, keyframe-based animation relies on playing back and interpolation of pre-calculated (3D) animations. A keyframe is a specific point on the time-line for which a complete deformed version of the animated mesh is stored. Keyframes are often sampled in regular intervals, which facilitates the creation and playback of the animation. More advances approaches analyze before-hand, which parts of the animation are more active than others and therefore require denser keyframe intervals. Consequently the number of keyframes required can be reduced considerably.

Another decisive factor for the quality of playback is the method of interpolation, also called “tweening” (for “in between”). Since keyframes are sampled at a much lower rate than playback interpolation is required for smooth animations. The simplest and least computationally expensive method is linear interpolation, which results in jerky animations if the keyframes are sampled sparsely. Advanced methods use more sample points such as cubic spline interpolation or even derivatives as in the case of Hermite interpolation, which result in much smoother animations. Naturally these methods are computationally more demanding.

In the course of this thesis there was a demand for a simple character animation solution with minimal computational costs. The author of this thesis therefore developed an animation package called FPK (Fixed Point Keyframe) that makes best use of the particular restrictions of mobile devices. FPK supports only linear interpolation and restricts texture coordinates to remain unchanged during animation which further reduces the computational overhead.



In the 11th century the new Gothic style spread
from France into all of Europe. It replaced
the old simplicity of the Romanesque by using
new symbols to praise

```
AutoThresh: 129
13.0 fps
Markers: 200
```

Figure 4.6: Virtual character animated using FPK

Coordinates are stored in 16-bit fixed-point with values a 5.10 precision. While this restricts models to a size of -32 to 32 units, it give millimeter accurate results and a large working volume when selecting meters for the units. Instead of storing normals directly, only an index to 16-bit a lookup table is kept. All together a single vertex requires only 8 bytes of memory per frame. Compared to storing a complete vertex including texture coordinates in floating point the memory load is thereby reduced by 75%. Keyframes are interpolated linearly using pure fixed-point math for highest play-back performance.

As a result, playing back an animated textured and lit 3D character (see Figure 4.6) on a mobile device causes almost no noticeable overall performance drop. FPK includes tools to import data from the Quake2 MD2[23] character format as well as the popular open source Cal3D[24] skeletal animation package. Furthermore a Maya plug-in that allows rigging and exporting a character directly into FPK files is under development.

FPK includes a high level scripting layer on top of the low level keyframe animation API. The higher layer exposes a simple scripting interface via a proprietary XML dialect. Using this XML dialect authors can create complex sequences of animations including precise timings for audio dubbing and subtitle rendering such as in Figure 4.6.

```
<Actor>
 <Setup geometry="mr_virtuoso.fpk" texture="mr_virtuoso.jpg" />
 <Action name="GeVenus" next="Idle">
  <Animation name="WalkToTable" speed="5.0" />
  <Animation name="Explain1" speed="4.0" />
  <Animation name="Explain2" speed="4.0" />
  <Animation name="Explain1" speed="4.0" />
  <Animation name="WalkToBook" speed="4.0" />
  <Subtitle language="german" time-from="6" time-to="18" text="Diese kleine..." />
  <Subtitle language="german" time-from="18.2" time-to="31.0" text="..ist wie.." />
  <Subtitle language="english" time-from="6.0" time-to="18.0" text="This female..."/>
  <Subtitle language="english" time-from="18.2" time-to="31.0" text="..as always" />
  <Audio language="german" file="venus_german.ogg" time="6.0" />
 </Action>
<Actor>
```

Figure 4.7: A short sample script of the Virtuoso application.

Figure 4.7 shows a short excerpt of the script that animates the 3D character in the Virtuoso application. The script begins by defining the keyframe and texture files to load for the animation. It then continuous by defining actions that can be triggered by the application. Due to space reasons, Figure 4.7 shows only a single action that

[23] http://tfc.duke.free.fr/coding/md2-specs-en.html
[24] http://cal3d.sourceforge.net/

includes the German voice over and sub titles for explaining the Venus item. Since the audio dubbing have different timings for each language, different animation and sub title timings are required too. In the sample above, The action starts by making the character walk to the table and then plays back several animations while Mr. Virtuoso is talking. At the end the character walks back to the book and another action called "Idle" is automatically invoked. While the animations are played back, the script player starts sub titles and audio playback according to the specified languages.

4.4 2D graphics and animations with Adobe Flash

As a final building block, 2D graphics solutions suitable for handheld devices will be considered. Practical, large and content-rich AR applications typically require not only 3D, but also 2D content such as 2D user interfaces or short animated movies. The Synchronized Multimedia Integration Language[25] (SMIL) is an open standard defined by the W3C committee and open source implementations exist. Unfortunately though the standard is very complex, limited in its practical features and no high quality open source players exist. Finally SMIL plays a negligible role outside academia; most graphics artists are used to Adobe Flash instead, which is the most widely established format due to its high quality tools for creating and playing back multimedia content.

The Adobe Flash plug-in[26] was therefore integrated, which is available for the Windows and Windows Mobile platforms, into our framework. An AR application can switch between full screen AR and Flash modes at any time. Flash applications can use the built-in scripting language ActionScript to communicate with the AR framework. Examples of the integration of 3D AR and Flash will be given in chapter 7.3.2.

4.5 Discussion

This chapter presented solutions for rendering graphics 3D and 2D content on mobile phones. The solutions developed are flexible and powerful enough for typical AR

[25] http://www.w3.org/AudioVideo/
[26] http://www.adobe.com/products/flash/

applications and facilitate the creation of graphical user interfaces, thus contributing to fulfilling argument H1 of the hypothesis of chapter 1.3.

A scene-graph library allows rapid prototyping of graphic intensive applications and is therefore preferable over immediate mode graphics programming (OpenGL ES, Direct3D Mobile). Consequently development time is reduced which supports hypothesis H2 for building larger AR systems.

Despite their small screen size, the resolution of today's mobile phones is similar to that of typical PC and console games in the mid 1990s. It is therefore sufficient for the creation of intuitive 3D und 2D user interfaces, which is explored in more detail in Chapter 7. The smaller screen size (rather than resolution) reduces the requirements on graphical content and hence makes it more economic to create large applications with many AR hotspots (see chapter 7.4 on the Schatzsuche game).

A natural advantage of AR over VR (and classical computer games) is that the area between points of interest does not have to be modeled, since it already exists in the real world.

Chapter 5

Distributed System

This chapter presents Muddleware a middleware for prototyping multi-user applications. It outlines the design rationales that led to the development of Muddleware and presents example applications that are built on Muddleware.

Today's game server technology was designed for high-performance scalable communication between desktop clients and game servers. Many technical aspects concerning clients (fast Windows PCs or consoles), servers (clusters hosted in compute centers), user interface (keyboard/mouse/screen) and networks (broadband) are chosen to match the underlying business model. Performance considerations will often lead to even more inflexibility, making multiplayer middleware solutions specific to a single game genre. For example, "instanced dungeons" in role-playing games (i.e., a copy of the game world exclusively for a small group of collaborating players) are necessary not only for increased player satisfaction but also for cluster load balancing.

Unfortunately, all these restrictions and optimizations make established multiplayer middleware rather unsuitable for developing Augmented Reality applications or games. Instead, it is more fruitful to borrow techniques and practices from research areas such as ubiquitous or wearable computing, who also deal with similar problems. To address the specific needs for AR middleware, a communication framework called Muddleware was developed. It is loosely inspired by Tuplespaces [28] – the name Muddleware hints at the diversity of the unstructured data managed by the framework.

5.1 Muddleware, a middleware for multi-user applications

Muddleware extends the traditional Tuplespace idea with advanced concepts such as publish-subscribe patterns and state-driven application logic. It uses XML to provide a modern and lightweight scriptable application programmer's interface. Although designed as a prototyping tool it provides enough performance and stability for large games with hundreds of clients. To our knowledge, Muddleware is the first multi-user middleware specifically designed for mobile Augmented Reality applications.

In designing Muddleware [114], the author of this thesis strove for creating highly functional software within a short period of time by re-using existing and reliable technology. Another important requirement was a platform-agnostic approach which allows for a wide variety of target devices. However, a large body of existing software in Studierstube research group builds on C++, which was also favoured for performance reasons. This ruled out a pure Java based approach, which was adopted in other platforms such as T-Spaces or the Event Heap.

5.2 Built on XML Technology

A key observation towards our current design was that the most widely established data model for network computing is the Extensible Markup Language, XML[27]. XML elements with named attributes can be seen as a specific representation of a tuple. Of course the most important difference between a conventional Tuplespace and an XML Document Object Model (DOM) is that the latter represents a hierarchical data model. A DOM permits a recursive definition of a tuple, which has child tuples as attributes of parent tuples. This extended definition of a Tuplespace as a DOM for favored for its increased expressive power, its elegant match to many typical data structures such as spatial hierarchies in the real or virtual world, and its use of mature existing technology, namely XML.

Muddleware was thus designed to adopt XML technologies for data storage, addressing and retrieval of data. XML fits our requirements for several reasons:

- XML is a self-documenting format that describes its structure as well as data types and meanings of values.

[27] http://www.w3.org/XML

- XML is simultaneously human- and machine-readable.
- XML is able to present many basic data structures such as lists, trees and records.
- XML's weak type system (when not enforcing a schema or DTD) supports quick changes in data structures and thereby assists rapid prototyping.
- XML structures can be addressed using well known methods such as XPath[28].
- High-quality open source implementations for many XML technologies exist and are freely available.
- High-quality tools for creating, modifying and validating XML documents are available.

In contrast to our approach, many existing servers use SQL for persistence. Muddleware uses XML plus XPath over SQL for several reasons: SQL is ideal for huge amounts of data in flat structures (tables). In the case of mixed reality games, data for all the heterogeneous devices, locations etc. usually requires less space but is highly structured which is a natural strength of XML. Furthermore due to XML's self-describing nature, existing structures can easily be extended with new attributes and child elements without breaking existing code.

Care was taken to keep the system as simple as possible and yet make it modular. The concept of sharing information via a high-performance database that acts as a distribution hub is easy to understand and easy to program: a minimal application requires only three lines of Muddleware specific C++ code.

Figure 5.1 gives a short example on how to manipulate data on the server. The first line of code creates a connection object. The next command connects to the server, in this case on the same machine. Naturally, in practice a developer would check if the connection attempt succeeded. Finally the last command updates an XML attribute of a node addressed by the specified XPath with the new value "13".

XPath addressing is similar to hierarchical file systems, except that qualifiers can be used along the path to select between alternatives. In the example in Figure 5.1, the command addresses an attribute called 'age', which is owned by an element called 'Client'. The 'Client' element is a child element of a root element called 'Application', which is qualified by having an attribute called 'name' with the value "MyApp". If the specified attribute(s) or element(s) can not be found the operation fails returns an error description.

[28] http://www.w3.org/TR/xpath.html

```
MUDDLEWARE::Connection* connection = MUDDLEWARE::Connection::create();
connection->init("localhost");
connection->updateAttribute("/Application[@name='MyApp']/Client/@age", "23");
```

Figure 5.1: Code excerpt to update an attribute on the server using the Muddleware Single-Operation-API

5.3 Database Server

The core of Muddleware is a "real-time" XML database that provides persistence and that can be addressed associatively using XPath. The database server is extended by several extension modules such as an XML based scripting language (Muddleware Script, see section 5.4) and a server-side state machine (Muddleware Controller, see section 5.5) that reacts on specific database changes and adds an independent thread of control to the server. All server components run under Windows as well as Linux. An arbitrary number of clients can connect to the server (see Figure 5.2) by using one of four currently available APIs: Immediate C++, Shared Memory C++, Java and Muddleware Script.

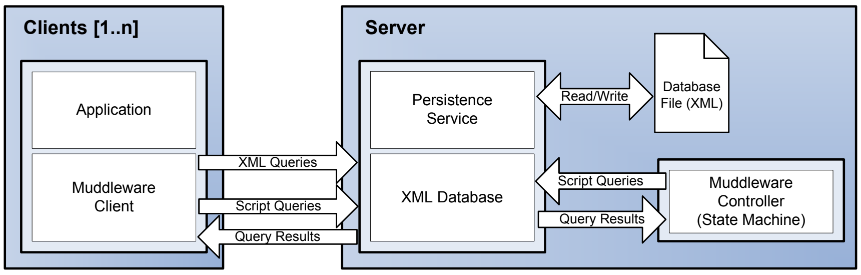


Figure 5.2: Muddleware components

The real-time XML database is the core of Muddleware. All data elements are stored as nodes of an XML DOM, using a modified version of the TinyXML[29] library. Clients can store arbitrary messages in the database in the form of XML fragments. As a query language, clients use XPath to specify query or update operations. No schema is enforced on the database, which facilitates rapid prototyping of communication patterns and improves performance.

[29] http://sourceforge.net/projects/tinyxml

Besides basic update and query operations, Muddleware supports atomic conditional operations: An update can be restricted to be performed only in case the value to be updated is identical to one of a list sent with the update operation. This can be used for locking of specific elements which is required for most non-trivial applications, essentially implementing a lightweight transaction protocol. Furthermore batched operations can be chained: Subsequent operations are only executed if previous ones were positively evaluated.

5.3.1 Update Notifications using Watchdogs

In addition to immediate databases operations, Muddleware allows clients to register Watchdogs (observers) for updates: as soon as an observed node changes, the client is informed about the update. This removes the need for polling and provides a simple, yet powerful publish/subscribe mechanism that can be used to create specific communication channels between clients. Two kinds of watchdogs are available: synchronous and asynchronous watchdogs. When an XML element addressed by an asynchronous watchdog is modified the server immediately sends a notification message to the client which causes a callback invocation. In contrast, synchronous watchdogs piggyback the notification to a regular message exchange between server and client. While the asynchronous variant is ideal for clients that talk to the server sporadically, the synchronous counterpart is suited for setups that rely on continuous frame-by-frame communication.

Naïve use of watchdogs can cause performance problems for the server, since for every update received by the server, all registered watchdogs need to be evaluated. In practice, clients are only interested in the updates concerning a few specific elements or issued by a few specific clients. In order to exploit this limited scope for better scalability, Muddleware allows the creation of so called interest groups [13]. When an element is modified, only those watchdogs that are in the same groups of interest as that client that caused the modification are evaluated. This allows splitting watchdog evaluation into (potentially) overlapping groups of clients and scales with the number of interest groups rather than the number of elements or clients.

Figure 5.3 shows a communication sequence of two clients sharing data via the Muddleware server. Upon start, both clients create an asynchronous watchdog observing the same XML element in the database. Immediately after one client updates that element, the other client receives the updated data.

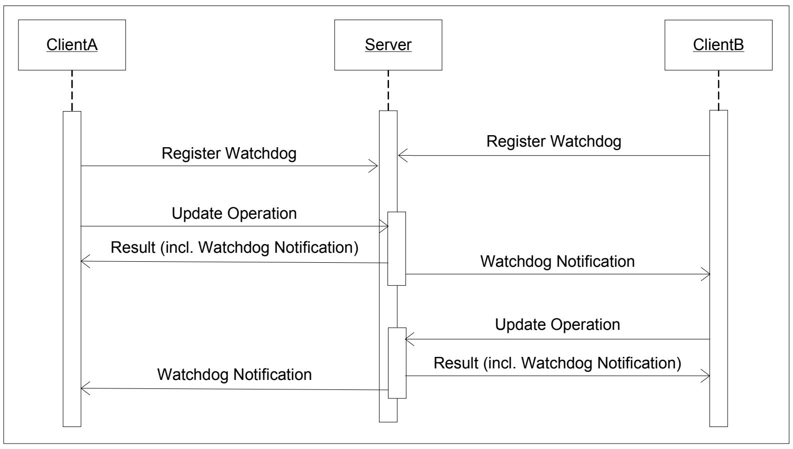


Figure 5.3: Two clients sharing data using an event channel

Muddleware uses the TinyXML DOM as a memory-mapped database. For session handling, which is the performance critical portion of a distributed system the ACE[30] Proactor was chosen. Its implementation handles all I/O operations internally, calling asynchronous system functions (such as for file reading or network communication). The user written handler only reacts to finished I/O operations, effectively allowing a single thread handle hundreds of clients concurrently. ACE provides multiple platform dependent implementations for the Proactor pattern: On each platform it uses highest performance APIs; such as asynchronous I/O and completion ports for Windows XP. The APIs are usually difficult and error prone to program, but effectively hidden behind the Proactor API. Using only a single thread removes the need for thread synchronization and thereby avoids overhead.

At short intervals a background thread dumps the complete database to disk; later versions may use a professional database management system for storage. ACE and TinyXML were chosen for best performance and portability: TinyXML offers only a minimal XML feature set and is therefore easily portable to any platform. ACE on the other hand is a complex framework, but is available on all major platform including Windows, Mac OSX and most UNIX variants. The Muddleware client can optionally run without ACE making it possible to port the client-parts of applications to platforms not supported by ACE, such as Symbian OS.

[30] ACE toolkit: http://www.cs.wustl.edu/~schmidt/ACE.html

5.3.2 Queries with XPath

XPath is a simple language for addressing elements and attributes of an XML document. It was originally designed for usage in XSLT and XPointer, but many independent XPath parsers are available which made it an interesting choice for Muddleware. XPath uses a compact, non-XML syntax which applied to an XML graph results in a list of zero or more elements. XPath is a pure declarative language and does not include constructs such as for-loops, which makes XPath easy and fast to implement.

In contrast to XPath, XQuery[31] allows writing procedural programs that are executed upon an XML document. XQuery includes all standard language constructs such as for-loops and if-then-else clauses. Consequently XQuery is highly complex. Furthermore, XQuery is a very new standard and only a small number of implementations exist so far.

Since Muddleware does not require the full power of XQuery, it was decided to build upon XPath and instead add specific features such as update operations on elements or attributes ourselves.

5.4 Muddleware Script

Muddleware Script is a simple XML dialect for expressing data-driven queries. Clients can create scripts and register these at the server. Each script is precompiled into tokenized form for fast execution and identified by a unique name that is used for execution later on. The scripting language allows invoking all available database actions such as adding, removing and querying elements and attributes. Multiple actions can be hierarchically combined using Boolean operations and functional composition – results of one query can be used as input for another query. Results are passed on from deeper nested actions to their respective parent nodes in the XML graph. An XML schema helps developers to create only valid script files.

Muddleware script exposes only few actions: ActionAnd and ActionOr are used to combine the result of multiple actions into a new single result. ActionEqual can be used to compare a result to a specific value and returns a Boolean value.

The most flexible action is ActionExecute, which executes database operations on the server. Via its ‘operation’ attribute, the developer can define which operation to execute, while the ‘xpath’ attribute specifies the items to operate on. As always in

[31] http://www.w3.org/TR/xquery

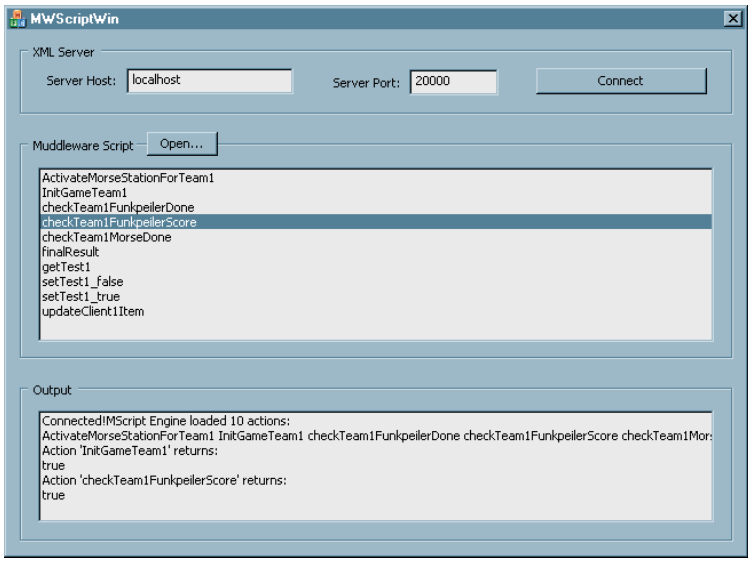
Muddleware, the user is free to address more than one element or attribute at once via the XPath. In such a case the operation is executed on all these items. Since this is not always what the user wants, there's a special attribute 'only-first-xpathitem'. Finally ActionExecute allows specifying a new value or a new name for the target item. The actions that can be executed are: getElement, addElement, addElements, removeElement, getElementExists, getAttribute, updateAttribute, addAttribute and removeAttribute.
Figure 5.4 shows a short sample action that checks if a dragon was defeated. The sample also demonstrates the strength of XPath's associative addressing: Qualifiers for attributes can be specified along the path through the DOM graph down to the target item. In this example the developer wants to check if two attributes hold specific values: only if both attributes hold the correct value the action "checkGameWon" returns true.

To achieve this, the script reads those attributes from the server database using ActionExecute commands. Both these commands return their results to their respective parent elements, which compare these results to the specified values. The Boolean results are then handed over to the ActionAnd element, which only returns true, if both sub-commands return true too.

```
<ActionAnd name="checkGameWon">
  <ActionEqual value="3">
    <ActionExecute operation="getAttribute"
         xpath="/Game[@name='Dungeon']/Level/@id"/>
  </ActionEqual>
  <ActionEqual value="dead">
    <ActionExecute operation="getAttribute"
         xpath="/Game[@name='Dungeon']/Enemy[@name='Dragon']/@state"/>
  </ActionEqual>
</ActionAnd>
```

Figure 5.4: Sample Muddleware script code sequence

A GUI-enhanced client tool (see Figure 5.5) to invoke Muddleware Scripts makes it easy to configure the XML database for testing and run-time re-configuration during the prototyping phase. Muddleware script is also used by the controller (see below) to execute complex commands as reactions to database events.



```
ActivateMorseStationForTeam1
InitGameTeam1
checkTeam1FunkpeilerDone
checkTeam1FunkpeilerScore
checkTeam1MorseDone
finalResult
getTest1
setTest1_false
setTest1_true
updateClient1Item
```

```
Connected!MScript Engine loaded 10 actions:
ActivateMorseStationForTeam1 InitGameTeam1 checkTeam1FunkpeilerDone checkTeam1FunkpeilerScore checkTeam1Mor
Action 'InitGameTeam1' returns:
true
Action 'checkTeam1FunkpeilerScore' returns:
true
```

Figure 5.5: Muddleware Script GUI

5.5 Muddleware Controller

For most real-world Augmented Reality applications a purely passive database server is not sufficient. An AR game needs to advance even if the players are inactive. For example, it may be necessary to enforce time-outs for certain parts of the game. Such overall game or application logic may not depend on the clients that may be unreliable. Instead, this requirement is better addressed by a server-side component with an independent thread of execution that can be loaded with application-specific code to perform all the housekeeping.

The Muddleware controller is such a server component that observes the database and reacts to changes based on internal interpretation of custom logic. The controller is shielded from outside influences in a kind of sandbox since it only communicates with the world by reading from and writing to the XML database (see Figure 5.2). This makes the controller free of side-effects and thus simplifies debugging and behavior analysis.

The controller connects to the XML database as a regular client, which allows isolating controller and database on separate machines. The controller's internal logic is built on a hierarchical finite state machine (FSM) with guards. FSMs are frequently used in nonlinear story-driven applications, but are suitable for a wide variety of application and user interface logic. They are very expressive and can be programmed by the purely data driven approach of specifying the state/transition graph (see Figure 5.6).

The FSM can run multiple paths in parallel. Each path starts at a specific entry state. Actions, expressed as Muddleware scripts, are executed upon entry and exit of states. Each state can have one or more transitions to other states. A transition is only taken if the transition's guard condition holds. Transition guards refer to conditions on elements in the XML database, updated by Muddleware script actions. Likewise all actions triggered by the controller are implemented using Muddleware Script. The controller writes its own state into the XML database, which allows monitoring and influencing the state machine remotely.

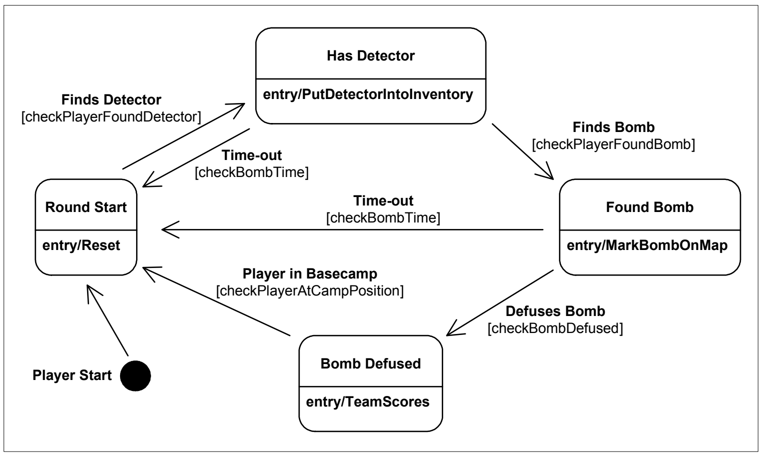


Figure 5.6: Example state graph for the Muddleware controller

The controller's state graph is supplied as an UML state chart, encoded as an XMI file. The XML dialect XMI can easily be parsed in an XML framework and the controller can directly operate on the resulting DOM. This type of state chart was originally designed for software engineering, and high quality graphical editors (e. g., Poseidon for UML[32]) exist for visual programming. Consequently game designers can program the controller without any knowledge in programming languages or even XML.

A simple state graph for a typical AR game scenario is shown in Figure 5.6. States are represented as rectangles, each having an entry action that is executed when the state is activated. Arrows illustrate transitions. Each transition has a name (in bold letters) and a guard (in square brackets) that links to a Muddleware script action. The graph starts with the initial state (marked as black disk) and immediately jumps into

[32] http://www.gentleware.com

the state “Round Start”. The first task for the player is to find the detector which in succession allows finding the bomb. Finally the player has to defuse the bomb to score for her team. If the player is to slow the bomb will explode which automatically restarts the round.

5.6 Muddleware Client

Muddleware provides a choice of four client-side APIs to target a broad audience. While the C++ immediate API allows fine graded communication with the server it also requires most effort in programming. In contrast, the C++ shared memory API sits on top of the immediate API and provides an easy high-level mechanism to access specific elements on the server. Consequently it is the preferred solution in most cases for C++ clients. In many situations, such as games running on mobile phones or web browsers, C++ is not the tool of choice. To target these platforms a pure Java implementation was created with an API similar to the C++ immediate API. Finally, clients can use the aforementioned Muddleware script for communication with the server from any client that is able to send and receive XML strings. The wide choice of client-side APIs enables quick integration of a wide variety of existing applications with Muddleware, which is a necessity to instrument an existing environment for a mixed reality game:

C++ Immediate Client API

The C++ immediate API provides lowest-level access to client-side Muddleware and is implemented as a toolbox of C++ classes. Programmers have the choice between two different approaches:

- With the **single operation API** they can invoke simple methods such as getElement(), updateAttribute(), etc.
  Figure 5.1 shows a minimal code sample of updating an attribute in the XML database. After creating the connection object and connecting to the server, a single line of code is enough to write the new value “Peter” into an existing attribute. The disadvantage of such a simple API is that all calls that invoke database operations use blocking I/O and will halt the thread of execution until the server's reply arrives. While this is sufficient for applications that query the database only upon user actions (e.g. a chat client), most interactive applications would suffer severely from this.

- Alternatively programmers can use the **multi operation API** which differs in that operations are stored in a request object which is then sent to the server. The advantage is that multiple operations can be batched and the creation of operations is uncoupled from the actual sending/receiving process. Furthermore the process of sending and receiving packets is uncoupled from the main thread which allows the client to perform useful work while waiting for the server's reply. Figure 5.7 shows an example of the multi operation API. All four operations are stored in the request object and then sent together to the server. After calling send()the client the time until the reply arrives for other tasks that are independent of the result. Alternatively the client could have used sendAndReceive() which blocks until the reply is available.

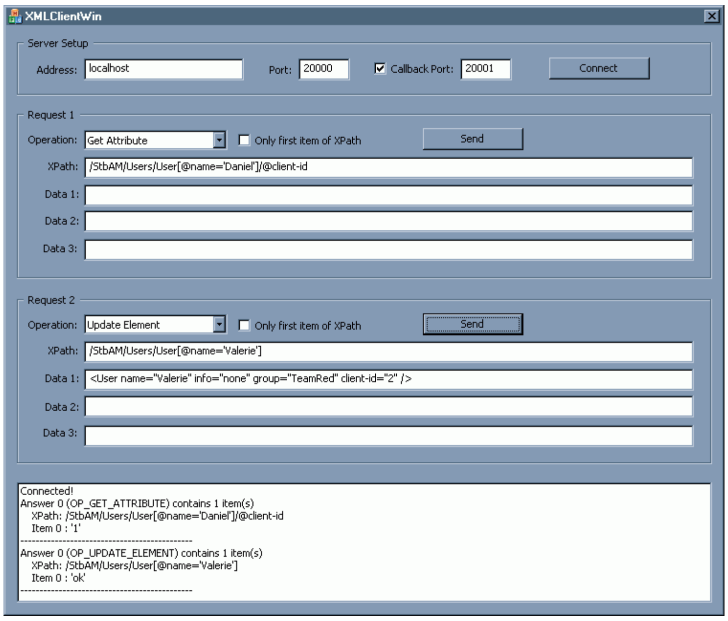
```
MUDDLEWARE::Request* request = MUDDLEWARE::Request::create();
MUDDLEWARE::Reply* reply = MUDDLEWARE::Reply::create();

request->updateAttribute("/MyApp/Owner/@name", "Peter");
request->removeAttribute("/MyApp/Owner/@address");
request->addElement("/MyApp/User", "<User name='John' age='35' />");
request->removeElement("/MyApp/User[name='Fritz']");

connection->send(*request);
//
// ...other application code independent of the reply goes here...
//
connection->receive(*request, *reply);
```

Figure 5.7: Code excerpt using the Multi-Operation API performing 4 operations at once

A simple GUI tool (see Figure 5.8) allows developers to create and test operations interactively before committing the communication statements to actual source code.



Address: localhost
Port: 20000
[✓] Callback Port: 20001

Request 1
Operation: Get Attribute
[ ] Only first item of XPath
XPath: /StbAM/Users/User[@name='Daniel']/@client-id
Data 1:
Data 2:
Data 3:

Request 2
Operation: Update Element
[ ] Only first item of XPath
XPath: /StbAM/Users/User[@name='Valerie']
Data 1: <User name="Valerie" info="none" group="TeamRed" client-id="2" />
Data 2:
Data 3:

```
Connected!
Answer 0 (OP_GET_ATTRIBUTE) contains 1 item(s)
   XPath: /StbAM/Users/User[@name='Daniel']/@client-id
   Item 0 : '1'
---------------------------------------------
Answer 0 (OP_UPDATE_ELEMENT) contains 1 item(s)
   XPath: /StbAM/Users/User[@name='Valerie']
   Item 0 : 'ok'
---------------------------------------------
```

Figure 5.8: Muddleware Client GUI

C++ Shared Memory API

Manually sending updates and receiving query results is often tedious in application level code. Although it provides finest graded control, such a level of control is not always required. Remote object libraries are more convenient since they allow accessing server data via proxy objects that behave like local data objects. The proxies take care of transparently synchronizing all access with the remote database: Whenever the client changes an object's value, this change is forwarded, and changes by other clients are automatically incorporated. Even when sharing data is not the goal such a mechanism can be used for persistence, which is especially interesting for clients that have no storage capabilities or lack required robustness.

```
MW_SHARED_DECLARE_ELEMENT_3_ATTRIBUTES(
BossEnemy, "/MyApp/Boss",
String, name,
Integer, healthpoints,
Boolean, active
);
```

Figure 5.9: Usage of a C++ shared-memory class

The C++ shared memory API implements such a feature set and is therefore the preferred C++ client API for most applications. A programmer can declare a C++ data structure directly in C++ source, which is then automatically implemented by usage of macros and internal tool methods (see Figure 5.9). No interface compiler such as used by CORBA is required.

```
BossEnemy theBoss;
theBoss.setConnectionManager(manager);

std::string boss_name = theBoss.name;
theBoss. healthpoints = 100;
```

Figure 5.10: Usage of a C++ shared-memory class as declared in Figure 5.9

Figure 5.9 shows an example declaration of a structure called 'BossEnemy' that is linked to an XML element addressed with '/MyApp/Boss'. BossEnemy has three member variables: a string called 'name', a numeric value called 'healthpoints' and a Boolean value called 'active'. After declaration, the client programmer can then use the class like a regular C++ structure (see Figure 5.10). The proxy objects have overloaded access operators that transparently forward all updates to the server.

Java Client API

The Java client API is a pure Java implementation of the Immediate API. It was developed for using Muddleware with PC-based Java games and applications, web browser games and Java-enabled mobile phones. Its feature set and API are very similar to that of its C++ counterpart.

XML Client API

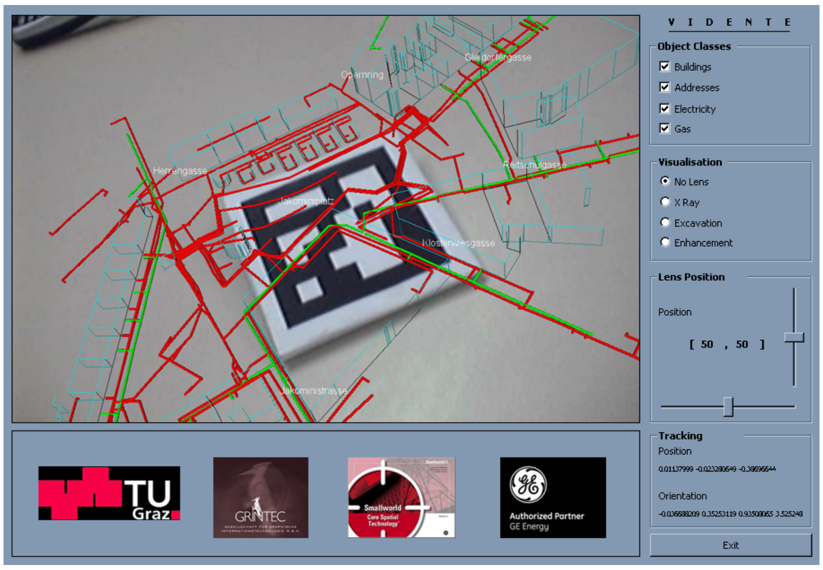
Instead of using the C++ or Java client API to batch database operations, client developers can also write Muddleware script. A script file is loaded by the client API, sent to the server which registers all embedded actions. Clients can then invoke an arbitrary number of operations with a single call to a registered XML action. This technique reduces the network load and provides a more data-driven approach than writing C++ code.
Figure 5.4 shows a short example script excerpt.

5.7 Graphical User Interface Generation

A recent extension to the Muddleware system called Thekla allows the easy generation of graphical user interfaces (GUI). Using the Qt Designer[33], application developers can build user interfaces without any programming. Thekla automatically pushes the states of all GUI elements ("widgets") onto the Muddleware server, which are then forwarded to the application via a Watchdog notification.

A special Thekla module for the OpenInventor scene-graph library allows creating connections between scene-graph attributes ("fields") and Qt widgets without writing a single line of source code. Figure 5.11 shows a simple Qt user interface controlling an OpenInventor application using Thekla. The controls on the right side of the application window directly control data of the scene-graph. In this figure, the Inventor rendering output is embedded into the Qt application's user interface. Alternatively, due to Muddleware's networking capabilities, the controlling GUI and the 3D rendering application could also run on different screens and computers.

Thekla was developed by Christian Pirchheim in the course of his master thesis and is used in combination with Muddleware in various research projects. More information on Thekla can be found in [79].



V I D E N T E

Object Classes
- ☑ Buildings
- ☑ Addresses
- ☑ Electricity
- ☑ Gas

Visualisation
- [✓ No Lens]
- X Ray
- Excavation
- Enhancement

Lens Position
Position [50 , 50]

Tracking
Position
0.01137999 -0.023280549 -0.38696544
Orientation
-0.036683209 0.35253119 0.93508065 3.525248

Figure 5.11: A Qt GUI controlling an OpenInventor application using Thekla

[33] QT: http://www.trolltech.com/products/qt

5.8 Performance

XML encoding and decoding as well as XPath decoding is known to be a computationally intensive task. To prove that the Muddleware server is capable of serving many clients some benchmarks were performed on a Windows XP machine with 1 GB of RAM and a Pentium 4 processor at 3.0 GHz:

A simple benchmark showed that the XML server can easily handle thousands of complex requests per second. To test Muddleware with a more practical example the server's CPU and network load was measured while running 50 instances of a pre-recorded 4-player Virtuoso game (see chapter 7.2) in parallel. The server's CPU load with 200 concurrent clients is ~60% while the 100 Megabit network was operating at a capacity of just 4%.

5.9 Discussion

This chapter presented Muddleware, a solution for creating distributed mobile AR applications. Chapter 7 presents various applications such as Virtuoso and MARQ that were built using Muddleware. In the number of interacting parts and users these applications go beyond previously created mobile AR applications. We actually noticed more collaboration between the users than on PC-based systems as shown in the study on collaborative edutainment in chapter 7.2.2, hence supporting hypothesis H1 of this thesis.

The blackboard mechanism of Muddleware is well known in pervasive computing and makes prototyping easy. It is very forgiving for design flaws in the protocol, which is further improved by the self-describing nature of XML. The decoupling of sender and receiver simplifies changing communication patterns later on.

Other than with directly connected communicating entities which result in up to N^2 relationships, the centralized blackboard mechanism requires only N relationships and therefore scales well with the number of clients (see chapter 5.8 on performance).

Of course the use of Muddleware is not limited to AR on phones, but certainly a contribution to hypothesis H2.

Chapter 6

Software Architecture

This chapter presents the system architecture of our handheld AR client. It also presents the Sphinx engine, a solution specialized for creating multi-player AR treasure hunt games.

6.1 Studierstube ES

Our Handheld AR framework, called Studierstube ES[34] (StbES, ES for embedded subset), was created with portability and lightweight footprint in mind. StbES is the third generation of our software for Augmented Reality on small mobile devices. While previous versions of our handheld AR software aimed at maintaining compatibility with the PC-based Studierstube software, StbES has been rewritten from ground up as a legacy-free design without compromises related to compatibility concerns. It runs cross-platform on Windows CE (the target platform), Windows XP (as a development or high-performance client platform) and also partially on Linux. For the future a Symbian port is planned and prepared, but currently the main focus is on Windows CE.

Our system runs smoothly on smartphones, PDAs, UMPCs and Tablet PCs. In all configurations all processing is done natively on the client, which proved to be more efficient than outsourcing computing tasks to a server. Typical frame rates for the StbES applications running on smartphones are in the order of 10-20 fps, depending on the actual application, tracking setup and target device.

For multi-user message passing, the client maintains a constant connection to the Muddleware server if available, but it can run fully stand-alone too. The

[34] http://studierstube.org/handheld_ar/stbes.php

combination of self-sufficient, low cost clients together with a single server that can easily handle a large number of clients makes this a scalable system.

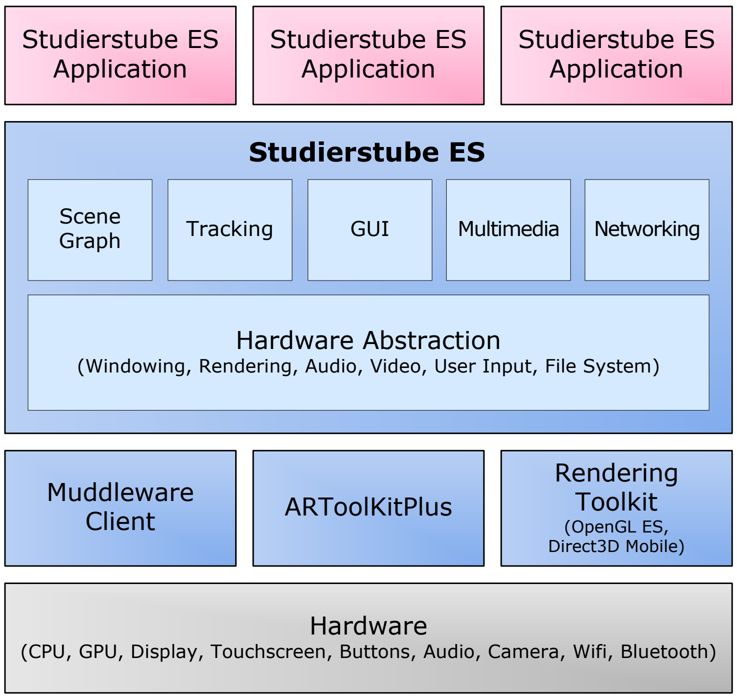

Studierstube ES Application
Studierstube ES Application
Studierstube ES Application
Studierstube ES
Scene Graph
Tracking
GUI
Multimedia
Networking
Hardware Abstraction
(Windowing, Rendering, Audio, Video, User Input, File System)
Muddleware Client
ARToolKitPlus
Rendering Toolkit
(OpenGL ES, Direct3D Mobile)
Hardware
(CPU, GPU, Display, Touchscreen, Buttons, Audio, Camera, Wifi, Bluetooth)

Figure 6.1: Component based-design of the Handheld AR framework

A component based client software architecture (see Figure 6.1) was created to accelerate the task of developing, porting and deploying collaborative applications. The main component is the Studierstube ES framework that allows running multiple concurrent networked applications. Its feature set can be reduced at compile time to minimize the footprint for setups that do not require all available capabilities.

Hardware specifications of mobile devices vary far more than on desktop computers. For example, while there exist basically no desktop camera that delivers video feed in portrait mode, both variants, portrait as well as landscape format are common on mobile phones. Unfortunately the camera's aspect ratio often does not follow the screen's format, which means that on some devices full screen video background is not possible. Consequently configurability is of high concern for a handheld AR system. During the design phase of Studierstube ES care was taken to plan for all circumstances that became apparent after studying the development guidelines of Windows Mobile and Symbian devices.

Boot Strapping

Studierstube ES uses a tiny boot loader (see top of Figure 6.2) that is only a few kilobytes in size and contains only minimal features. The advantage over a single, large executable is that the minimal boot loader can check the runtime conditions before the full framework runs into problems during start-up. For example, a specific issue under Windows CE is that the OS does not report which DLLs are missing, but simply complains that an application could not be started. To work around this problem and show the user meaningful error messages, the boot loader tries to load all required DLLs manually, thereby keeping control which DLLs could be loaded and which failed.

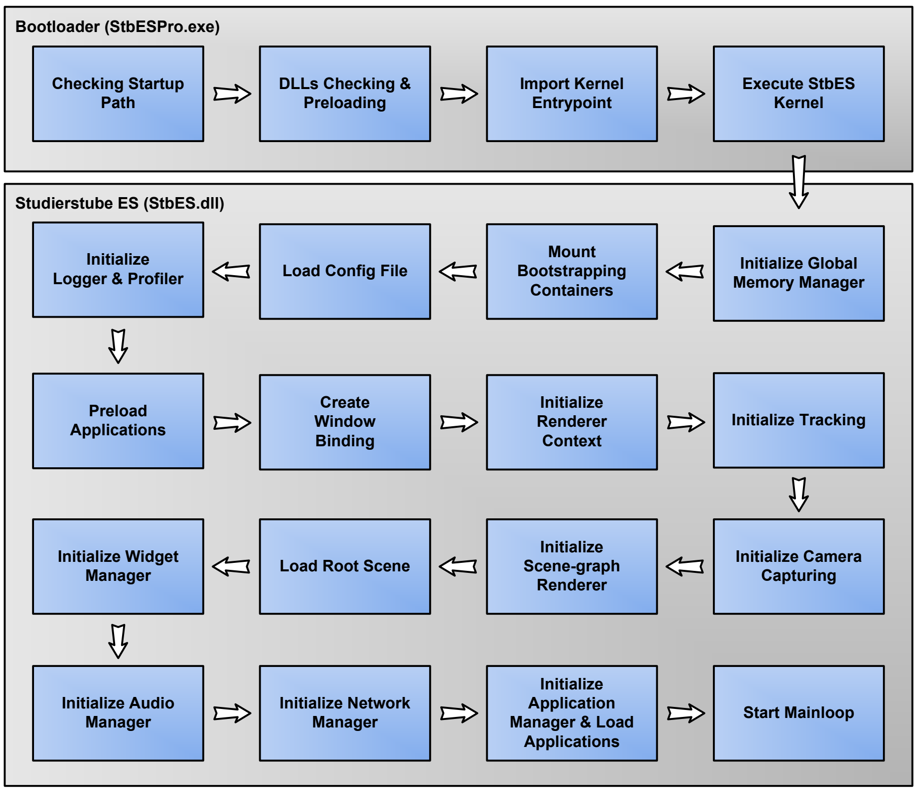

Bootloader (StbESPro.exe)
Checking Startup Path
DLLs Checking & Preloading
Import Kernel Entrypoint
Execute StbES Kernel
Studierstube ES (StbES.dll)
Initialize Global Memory Manager
Mount Bootstrapping Containers
Load Config File
Initialize Logger & Profiler
Preload Applications
Create Window Binding
Initialize Renderer Context
Initialize Tracking
Initialize Camera Capturing
Initialize Scene-graph Renderer
Load Root Scene
Initialize Widget Manager
Initialize Audio Manager
Initialize Network Manager
Initialize Application Manager & Load Applications
Start Mainloop

Figure 6.2: Chain of actions while bootstrapping Studierstube ES.

After all DLLs have been preloaded, the boot loader imports the kernel's entrypoint and executes the kernel. In some version of the framework the boot loader checks for updates over the network. In such a case the boot loader, which rarely changes, can download new versions of the framework before executing them.

The kernel (see bottom of Figure 6.2) begins its start-up by initializing the global memory manager. Many objects that are used system-wide such as the file system or the renderer are implemented as singletons. Some embedded operating systems, such as Symbian, do not allow writing to static global data. StbES therefore uses a global memory manager that holds the internal data of these singletons and stores it in thread local storage instead of static variables.

As a next step StbES mounts bootstrapping containers. Containers can hold files and are implemented as plain or encrypted ZIP files. Using the StbES file reading and writing classes makes working with plain files or files in containers completely transparent. Developers can mount and unmount containers at any time, but in order to put fundamental files such as the configuration file or the root scene into a container (e.g. to protect against modifications), StbES must mount that encrypted container early during start-up. Hence the name and key for this boot-strapping container are hard-wired at compile time.

StbES then loads the configuration file, which determines the rest of the start-up phase. It then initializes the logging and profiling capabilities, which also include memory logging to debug memory bottlenecks. Special conditions which are unique to Windows CE can create problems when loading DLLs at a late point of the process start-up (see chapter 4.2.1 for details). Developers can therefore decide to have their application DLL preload early in the start-up phase.

The kernel then runs the window manager, which creates a render target (window) either for rendering in system or in video memory, as specified in the configuration file (see chapter 4.1.5 on a discussion on software vs. hardware rendering). Next the render manager, which abstracts the basic rendering toolkit (either OpenGL ES or Direct3D Mobile) is initialized and creates the render context.

The tracking manager then creates all tracking subsystems such as the ARToolKitPlus tracker and initialized them. After this the video subsystem connects to the camera and starts retrieving video frames. In the next step the scene-graph renderer is instantiated, which loads the root scene to which applications attach their own scene-graphs to. The widget manager then creates font objects for 2D text output and prepares for creating and rendering widgets. The audio manager initializes and waits for applications to request audio support. After an optional network connection to the Muddleware server is created, the Application Manager starts loading and executing applications. Finally the kernel starts the main loop, which is outlined in the next section.

Main Loop

The StbES main loop is executed as a callback of the windowing system and is therefore independent of the underlying operating system. Figure 6.3 gives an overview of all the steps that Studierstube ES executes for generating a single frame. The following list describes each step in detail

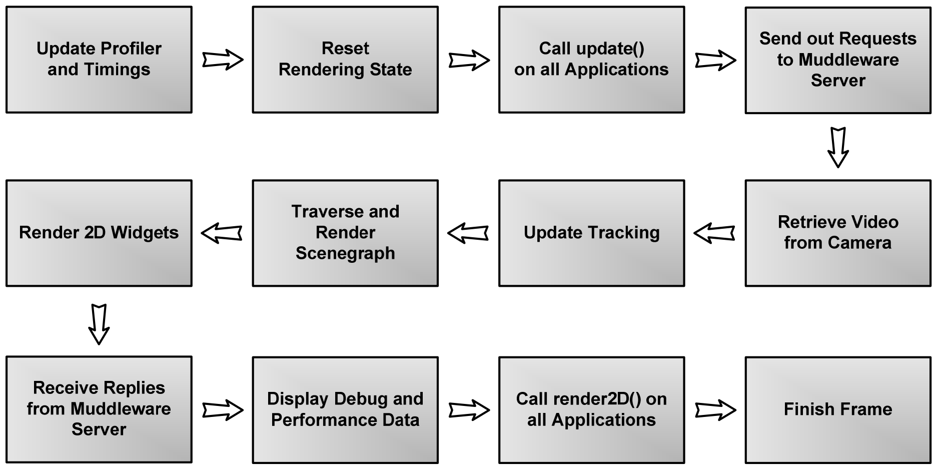

Update Profiler and Timings
Reset Rendering State
Call update() on all Applications
Send out Requests to Muddleware Server
Retrieve Video from Camera
Update Tracking
Traverse and Render Scenegraph
Render 2D Widgets
Receive Replies from Muddleware Server
Display Debug and Performance Data
Call render2D() on all Applications
Finish Frame

Figure 6.3: Workflow for processing a single frame in Studierstube ES.

- **Update Profiler and Timings:** Studierstube ES automatically profiles most of the following steps in order to give developers feedback on where the scarce processing time is spent. Furthermore this step initializes as per-frame timings and thereby creates global, framework-wide consistent timing information, such as used for animations or for interpolation and extrapolation of tracking data.
- **Reset Rendering State:** This action resets the OpenGL ES or Direct3D rendering state by setting it to well defined default values.
- **Call update() on all Applications:** Every active application's update() method is called. Application can use this to invoke actions before rendering starts, which is an optimal point in time to perform updates to the scene-graph. Applications can also queue requests for the Muddleware server, which will be available later in the same frame.
- **Send out Requests to the Muddleware Server:** All requests queued so far our sent out to the Muddleware server. Requests are only sent out, but not required to return instantly, which gives the server enough time to process the messages and send them back without stalling the client.

- **Retrieve Video from Camera:** StbES queries the camera for a new frame. In case a new frame is available, all objects that registered as VideoUsers are notified and handed over the new camera image.
- **Update Tracking:** The tracking subsystem calls all active tracking modules such as the ARToolKitPlus or FTW (a proprietary gyroscope tracker connected via Bluetooth) tracker. The resulting tracking data is stored in fields for later use.
- **Traverse and Render Scene-graph:** The scene-graph renderer updates all field connections and traverses the scene-graph for rendering.
- **Render 2D Widgets:** Studierstube ES' widget manager now draws all widgets such as buttons, images, etc. onto the screen, on top of previous 3D renderings.
- **Receive Replies from the Muddleware Server:** Replies from the Muddleware server are expected to have arrived by now. Otherwise Studierstube ES goes into a blocking wait. Most time consuming tasks such as video retrieval, tracking and rendering are executing in the slot between sending requests and receiving replies. The typical StbES frame duration is ~80 milliseconds, which gives the server enough time to react.
- **Display Debug Information and Performance Data:** StbES optionally displays debug information such as network errors, visible markers and performance data (e.g. frames per second).
- **Call render2D() on all Applications:** Each application is notified that the scene-graph traversal has finished and 2D data can be rendered on top of it now. Furthermore the applications can retrieve results for queries to the Muddleware server that have been queued in the update() method. Since each request was tagged with the application's id, it is guaranteed that every application receives only its own replies.
- **Finish Frame:** In the last step the renderer presents the just created frame buffer on the screen.

Configuration

StbES can be configured in many ways using XML config files. The following we list presents the most important configuration parameters:

- **Logging** is a highly valuable tool for debugging hard crashes. While on a desktop computer usually only the problematic application crashes, the same program can easily take the mobile phone down with it during malfunction. Logging allows checking for errors after rebooting the device. Since file logging can slow down the system considerably, StbES provides several log levels for fine adjustment.

- **Render Target:** Since many devices today do not include graphics chips yet, software rendering is still rather the rule than exception. While both software and hardware rendering produce the same results, they require different strategies (render paths) to achieve optimal performance. StbES therefore creates off-screen render targets ("PixMap" surfaces) for software rendering, which allow direct video memory access, and on-screen render targets for accelerated hardware rendering.
- **Window bindings** define how the application interacts with the operating system's screen and video memory management. StbES support two different window bindings: Windows for PixMap render targets can blit the off-screen image onto the screen. When using EGL window bindings, OpenGL ES handles the buffer management itself. In the case of PixMap windows StbES can rotate also the render target in 90° steps to adapt to users that hold the devices in non-standard ways.
- **Video capturing** is of major importance for handheld AR. StbES supports various video APIs including proprietary ones such as for the Gizmondo device. Developers can choose to use full automatic mode selection: StbES will then use heuristics to find a suitable video mode that best fits the screen's resolution: StbES then tries to scale and crop the video image in case it doesn't fit onto the screen in original size. Naturally, operations that change the original image as little as possible are preferred. Alternatively developers can manually select a specific video mode and configure cropping, zooming and format conversion.
- **Font support** in StbES is currently rudimentary since only pixel fonts are supported. Yet the font type and size can be freely configured to adapt to various screen resolutions.
- **Tracking** is an integral part of any AR application. While ARToolKitPlus is currently the major tracking system used in the handheld AR project, StbES allows integrating arbitrary tracking libraries.
- **Containers and encryption** can be important features when distributing applications to an open, unknown audience. In such a case a developer can put sensitive data into encrypted containers to protect the data as well as the application against misuse. Furthermore assembling lots of small files into fewer larger one simplifies application deployment.

6.2 Sphinx

For multimedia applications content creation, also known as authoring, is a major topic. While small projects can get by with using just a few simple tools, large applications require a carefully orchestrated content development pipeline. This issue is not unique to AR applications, but has also been recognized as a major problem in the area of professional game development.

Content creation for Augmented Reality is different from typical game development processes. While today's content in console games is usually a lot more extensive and complex compared to that of Augmented Reality, developing AR applications requires not only creating and managing virtual content, but physical content too. Typical Augmented Reality demonstrations work with small data sets that have been entered manually and do not require data management. Naturally this approach is not feasible for larger applications.

Even small projects suffer from basic problems such as getting content from graphical editors into the AR software. Larger applications have additional problems that are similar to professional game development: After content has been imported into the AR framework, it must be managed and put to use, which turns out to be a major problem in its own right as soon as the amount of data grows beyond a certain measure.

The Sphinx engine that was developed for a museum treasure hunt game (see chapter 7.4) specifically targets these problems of large AR applications. A large application such as a museum-wide game cannot be developed efficiently by creating custom code for every interaction throughout the game. It was therefore decided to create a custom game engine, Sphinx, on top of Studierstube ES and Muddleware. It serves as a basis for all interactive exhibits called hotspots that players can interact with. This engine must fulfill several requirements to be practical not only during the time frame of the research project, but after the project's end as well.

A primary concern is the efficient creation of new content for the interactive exhibits. The museum game that motivated the development of Sphinx targets a number of roughly forty hotspots, so the amount of development work per hotspot had to be kept at a minimum. Due to ongoing changes in the exhibitions it is furthermore important that existing hotspots can be edited by the museum personnel themselves, i.e., by users with limited technical knowledge. This requires simple to use, graphical tools that can be operated by non-programmers. Since it is out of the scope of the project to develop these tools ourselves it was mandatory relying on existing software instead.

For a multi player game, where teams collaborate in solving a common goal it would be desirable that each player has a unique experience while contributing a small amount to the overall task. In practice though, creating unique content for each player increases work for content creation enormously. E.g. a game by 10 players for 20 minutes of active game play equals 200 minutes of unique game play.

Consequently the game allows reconfiguring the setup to arbitrary numbers and sizes of teams. Each team plays the complete game independent of the other teams. The team that finishes the game first or solves most tasks wins the game. When a small group of users plays the game they can be split into just two teams competing against each other while larger groups can be divided into more teams.

The Sphinx engine is focused on creating and running adventure style augmented reality games. Like in traditional adventure games such as pioneered by Lucasarts[35] in the 1980s, most game logic can be reduced to standard user interface metaphors such as taking or giving items or combining items to create new ones. Consequently the majority of the game content can be scripted, using a simple interpreted control language, requiring custom C++ code only for those parts of the game that go beyond the capabilities of the engine.

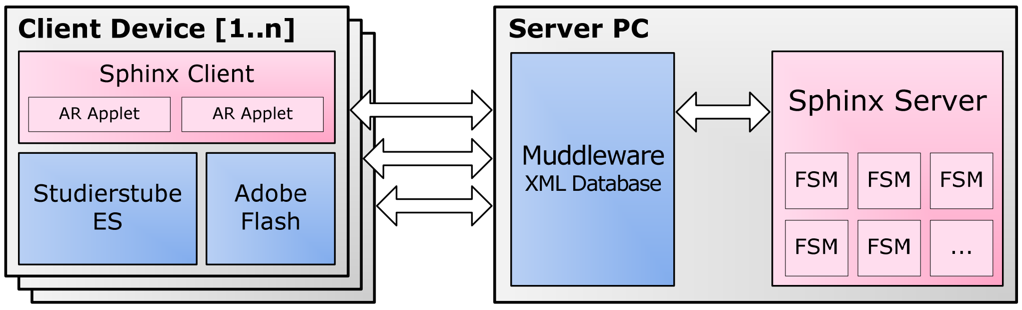


Figure 6.4: Sphinx client/server system layout

Sphinx consists of a server and a client part. The Sphinx server runs on a PC and interfaces only with a Muddleware server, usually running on the same machine (see Figure 6.4). For each team and hotspot the Sphinx server spawns a finite state machine (FSM) in a separate thread. Each FSM listens to specific elements on the XML server, such as waiting for clients to register at the hotspot. After a client registers, the FSM starts sending actions to the client by putting commands or state updates into the Muddleware database and waiting for the client to confirm. These actions range from showing dialog boxes to starting AR applets (see below).

[35] http://www.lucasarts.com

6.2.1 Sphinx Server

All intelligence controlling the hotspots is located in the finite state machines at the Sphinx server whereas the client mostly just reacts to the commands from the server. This restricts hotspots to locations where network connectivity is available. However, it improves robustness since it is primarily the mobile clients that are likely to fail due to malfunction, such as software problems, battery outage or misuse. In case of a client malfunction it is sufficient to restart the handheld for recovery since the complete game state is stored in the XML server.

The Sphinx server is an extended version of the Muddleware controller (see chapter 5.5), but more focused on the task of controlling a multi player adventure game. Both are similar in that they interact only with the Muddleware server directly. Yet, while the Muddleware controller executes Muddleware script commands, the Sphinx server executes C++ code, which allows going beyond the capabilities of Muddleware script.

Command	Description
CLIENT_ShowDialog(DIALOGNAME, AUDIONAME)	Opens a dialog and optionally starts playing an audio file
CLIENT_StartAudio(AUDIONAME)	Starts playing an audio file
CLIENT_StartModule(MODULENAME)	Loads a module (DLL) and starts the AR applet (see next chapter)
CLIENT_StopModule(MODULENAME)	Stops an applet and unloads the module
CLIENT_ModuleCommand(COMMANDNAME, PARAMS)	Sends a command and optional parameters to a module
CLIENT_StartTimer(TIMERNAME, DURATION)	Creates a named timer with a specified time-out value.
CLIENT_ActivatePhotoMode()	Activates the photo mode
CLIENT_DeactivatePhotoMode()	Deactivates the photo mode
CLIENT_ShowActionAR(NAME)	Executes a scripted AR action (animation)
CLIENT_ActivateTools(TOOL1, TOOL2, ...)	Activates one or more tools in the inventory
CLIENT_DeactivateTools(TOOL1, TOOL2, ...)	Deactivates one or more tools in the inventory
CLIENT_LeaveContext(CONTEXTNAME)	Tells the client to leave the context
DB_InventoryStore(ITEMNAME)	Put an item into the player's inventory
DB_CloseContext(CONTEXTNAME)	Closes a context (hotspot)
DB_ValueSet(VARIABLENAME, VALUE)	Sets a new value for variable (used for inter-hotspot communication)

Table 6.1: List of all commands executed on the Sphinx server.

While a Sphinx server purely based on the Muddleware controller would probably be feasible, it was decided to develop a custom tailored server with a more focused API, which in consequence facilitates the development of the hotspot state charts. Therefore, instead of requiring the application developer to write Muddleware scripts that deal with the XML database, the Sphinx server exposes game specific commands.

Table 6.1 gives an overview of all commands executed on the Sphinx server. These commands are executed when entering or leaving a state. All CLIENT_ commands are written into the command node which the client listens to via a watchdog and result in an immediate action executed on the client. The DB_ commands read and write values from the Muddleware database with only indirect influence on the client (for example, DB_InventoryStore will make the client show an icon in the inventory, but only because the client has a watchdog on the complete inventory). All these commands still only interact with the Muddleware database, but the streamlined, focused API of the Sphinx server makes the development and maintenance of hotspots much easier than using the Muddleware controller.

Command	Description
CLIENT_Action(ACTIONNAME)	The client send a message named ACTIONAME
CLIENT_AudioFinished(AUDIONAME)	The client finished playing an audio file name AUDIONAME
CLIENT_TimerFinished(TIMERNAME)	A timer named TIMERNAME ran out
DB_InventoryFull()	The client's inventory is full
DB_InventoryNotFull()	The client's inventory is no full
DB_InventoryIs(ITEMNAME)	The client's inventory contains an item name ITEMNAME
DB_MarkerVisible(MARKERNAME)	The a specific marker is visible for the currently registered client
DB_MarkerNotVisible(MARKERNAME)	The a specific marker is not visible for the currently registered client
DB_ValueIs(VARIABLEBNAME, VALUE)	Checks if a variable has a specific value
DB_PlayerRegistered()	A client is registered at the context

Table 6.2: Guards for transitions between states.
A guard returns true if the described condition is met.

Table 6.2 shows all guards of the Sphinx server. Only when a guard evaluates to true, the state machine takes the corresponding transition into the next state. CLIENT_ guards wait for the client to send specific messages, such as that an audio file finished playing or an action occurred (e.g. the user pressed a button in a dialog). DB_ guards on the other hand, check for states in the database.

Since each state machine runs in a separate thread, every hotspot is generally completely independent of all other hotspots. Only each FSM's graph decides if a hotspot can be played at any time or if it is activated only after specific conditions are met, such as another hotspot being already finished.

State machines are frequently used in nonlinear story-driven games, but are suitable for a wide variety of application logic. A natural strength of state machines is that they can be created and edited without programming skills such as writing source code. Instead game designers can use graphical editors for visual programming and use basic building blocks provided by the engine to modify the state or react onto state changes.

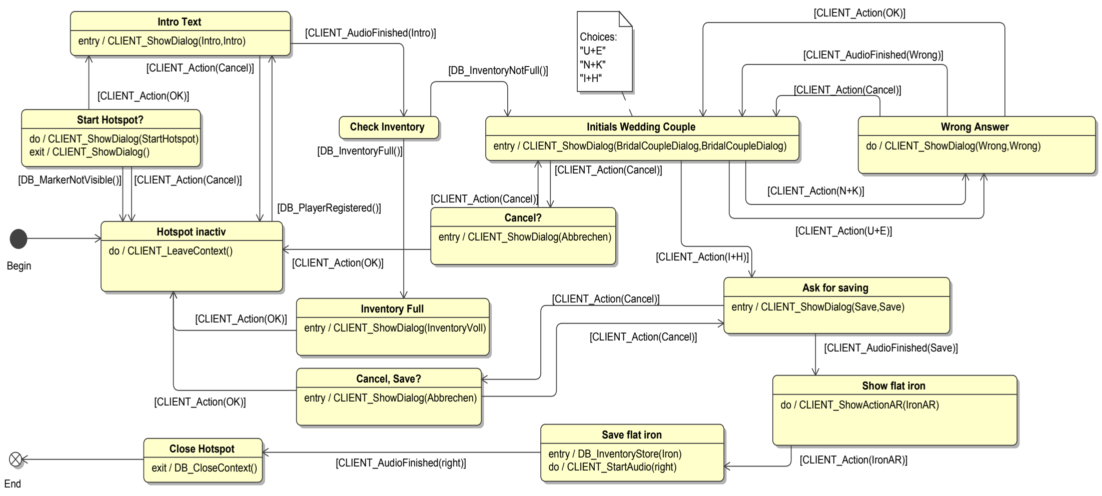
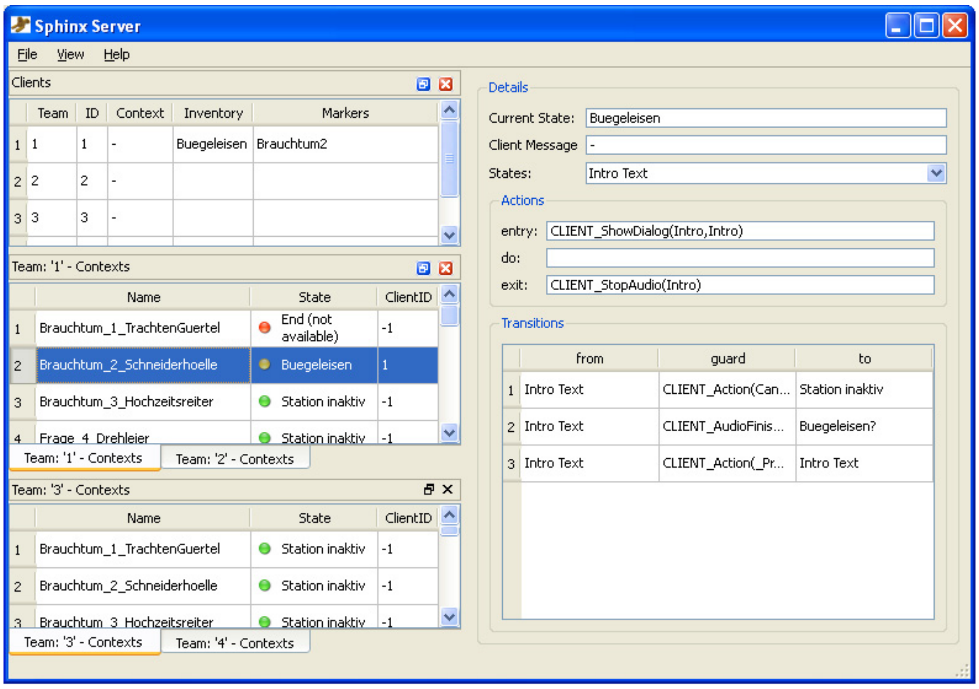


Figure 6.5: Finite state machine of the "Trachtengürtel" (a traditional belt in Carinthia) hotspot

Similar to the Muddleware controller, the Sphinx server's state graph is supplied as an UML state chart (see Figure 6.5) encoded in XMI format. Sphinx parses the open XML-based XMI format using TinyXML and the state machine can directly operate on the resulting Document Object Model (DOM). As can be seen in Figure 6.5, hotspots typically start in an "inactivate" state where the FSM resides until a client registers or jumps back to if the player cancels or something goes wrong. The player is then usually introduced to the hotspot via a dialog box combined with an audio message. At the hotspot in the Figure above the user has to answer a multiple-choice question (state "Initials Wedding Couple"). If the player answers correctly he is rewarded with an item. In this case, a 3D model of a flat iron flies from the real environment (a tailor's workshop) onto the handheld and falls into the player inventory. Finally the FSM closes down the hotspot.

Figure 6.6 shows a screenshot of the Qt-based user interface of the Sphinx server that runs the state machines. The Sphinx server's GUI allows a game designer to browse the full game state at any time. For each hotspot the current state and all its transitions leaving from there can be viewed which is an important tool during development and debugging. Single hotspots or all hotspots can be reset to restart or replay them. The interface also lists all currently connected clients and shows their current position and inventory items. Via the use of the Muddleware scripting interface the corresponding context and client properties in the database can be altered in any way, like placing or removing specific items in the client inventory. This gives the operator the possibility to fulfill prerequisites for playing a hotspot and, therefore, assist players if they require help at a hotspot.



Clients

	Team	ID	Context	Inventory	Markers
1	1	1	-	Buegeleisen	Brauchtum2
2	2	2	-		
3	3	3	-		

Team: '1' - Contexts

	Name	State	ClientID
1	Brauchtum_1_TrachtenGuertel	● End (not available)	-1
2	Brauchtum_2_Schneiderhoelle	● Buegeleisen	1
3	Brauchtum_3_Hochzeitsreiter	● Station inaktiv	-1
4	Frage_4_Drehleier	● Station inaktiv	-1

Team: '1' - Contexts | Team: '2' - Contexts

Team: '3' - Contexts

	Name	State	ClientID
1	Brauchtum_1_TrachtenGuertel	● Station inaktiv	-1
2	Brauchtum_2_Schneiderhoelle	● Station inaktiv	-1
3	Brauchtum_3_Hochzeitsreiter	● Station inaktiv	-1

Team: '3' - Contexts | Team: '4' - Contexts

Details

Current State: Buegeleisen
Client Message: -
States: Intro Text

Actions

entry: CLIENT_ShowDialog(Intro,Intro)
do:
exit: CLIENT_StopAudio(Intro)

Transitions

	from	guard	to
1	Intro Text	CLIENT_Action(Can...	Station inaktiv
2	Intro Text	CLIENT_AudioFinis...	Buegeleisen?
3	Intro Text	CLIENT_Action(_Pr...	Intro Text

Figure 6.6: Sphinx server user interface based on Qt (team specific log history of state transitions and events are hidden)

The user interface can be reconfigured to adapt it to the game designer's needs, such as docking various team lists or log messages together. The number of teams and clients per team are configured in the Muddleware database, beside the definitions of the station-contexts where all of the game content is specified.

6.2.2 Sphinx Client

The Sphinx client runs as an AR application on top of Studierstube ES (see Figure 6.4). During startup it connects to the Muddleware server and retrieves a list of all available hotspots. When the handheld detects a marker it asks the user if he wants to play that hotspot. The client application then registers at the XML server which triggers the aforementioned finite state machine on the Sphinx server. The Sphinx engine currently supports four different types of hotspots (or a combination of them). The list below presents these types in the order of increasing technical complexity:

- **Pure 2D GUI:** At these hotspots, only simple 2D GUI based interaction is available. This can be multiple-choice questions or asking the player to use a specific item already found or still required to find. These hotspots are very simple to create as they typically only require a short script and a few images to be shown in a dialog box.
- **Flash-based applets:** For these hotspots the Sphinx engine starts an Adobe Flash movie that can take over full control of the device and communicate with the Sphinx server. Using Flash allows incorporating existing high-quality design tools which graphics and web designer are used to. Via ActionScript, Flash applets can talk to the Sphinx server such as waiting for new events or sending results.
- **Scripted AR hotspots:** These hotspots make use of the AR interaction features of the Sphinx engine. They typically require the player to interact with the environment in a simple way such as using a game related virtual tool (e.g. the lens for virtually looking in more detail at an exhibition item) or applying a previously found item to the current hotspot.
- **AR applets:** By programming custom C++ code, more complex and unique interactions that go beyond the scripting capabilities of Sphinx can be created. Multiple AR applets can be grouped into DLLs that are loaded automatically on request. An example for such a hotspot is the "Silent Piano" (see chapter 7.4) where the user has to watch the handheld virtually playing a piece of music on the real piano and then repeat it.

6.2.3 Offline State Machine

The Sphinx engine was originally planned to run in an area that is mostly covered via a wireless network that includes all the locations of interactive exhibits. After running first tests in the museum and presenting the results to the museum staff, the museum management decided that it would be too expensive to outfit all required areas with permanent wireless connectivity.

Hence, the Sphinx engine was extended to also run the state machines "offline" on the client. The local state machine was designed to expose the very same API as on the Sphinx server so that existing state machines designed for the server remain unchanged. In offline mode network communication with the client is not mediated via the network but directly from the state machine to the client's communication management class.

Since the games based on Sphinx are still multiplayer games, network connectivity is still required at some point. The client searches permanently for network connections. As soon at is succeeds in connecting to the server both exchange data to provide updates into both directions. As a result the server is able to integrate the latest actions of the player into the overall game state and the client gets instantly receives this up to date overall game state. Chapter 7.4.1 presents the actual setup of the redesigned of team base that shows an overview map.

6.3 Performance

To prove that the system developed in this thesis is highly suitable for AR on phones, several benchmarks were run that reproduce typical situations on a series of devices ranging from a low-end smartphone to a brand new device with hardware accelerated graphics. The following list gives an overview of the devices used in the benchmarks (see Figure 6.7):

- **HTC Tornado:** The Tornado is a small, low end smartphone, with a 240x320 pixels screen in a typical phone form factor with a T9 keypad. Its Texas Instruments OMAP 850 CPU running at 200MHz is common for this device class. The camera delivers images in YUV12 format at 320x240 pixels landscape format and 15Hz. Since the screen is in portrait format, only the overlapping 240x240 pixels can be used as video background AR.
  This phone is branded under the names: O2 XDA IQ, O2 XDA Orion, Swisscom XPA v1240, T-Mobile SDA US, Vodafone VDA II, Vodafone v1240, Qtek 8300, Qtek 8310, Dopod 586W, Dopod 577W, I-Mate SP5
- **HTC Excalibur:** The Excalibur is a typical smartphone with a landscape screen of 320x240 and a full QWERTZ keyboard below the screen – a form factor that became popular due to the success of the Blackberry devices. It is based on the

same Texas Instruments OMAP 850 200MHz CPU as the Tornado, hence similar performance measurements are to be expected. Its camera delivers images in 320x240 in YUV12 format at 15Hz.
This phone is branded under the names: Orange SPV E600, T-Mobile MDA Mail, T-Mobile Dash, Dopod C720W, O2 XDA Cosmo, HTC S620

- **Palm Treo700W:** The Treo was the first Windows Mobile powered device released by Palm. Its screen is square at a resolution of 240x240 pixels. The camera delivers images at 320x240 pixels in YUV12 at unusual 28Hz. It possesses a full QWERTZ keyboard. The Treo uses an Intel XScale CPU that runs at 312MHz, but can be overclocked to 520MHz.
- **Motorola Q:** The "MotoQ" has the same form factor as the HTC Excalibur: A full QWERTZ keyboard is available below a 320x240 pixels screen. The camera delivers images in 320x240 at 15Hz in YUV12 format. Its Intel XScale CPU runs as 312MHz.
- **Motorola Q9:** The "Q9" is the successor to the MotoQ. Although it has the same form factor, it incorporates completely different hardware: Its Texas Instruments OMAP 2420 CPU has hardware 3D support for the vertex as well as pixel stage and even includes a floating point co-processor. The camera delivers images at 320x240 pixels in RGB565 format at 15Hz. Unfortunately, the Q9 does not come with an OpenGL ES driver, but only a Direct3D Mobile driver. Furthermore the floating point unit (FPU) is deactivated and requires low-level programming to be accessible. To futher compare the influence of hardware accelerated rendering, the Q9 was benchmarked with and without GPU support.

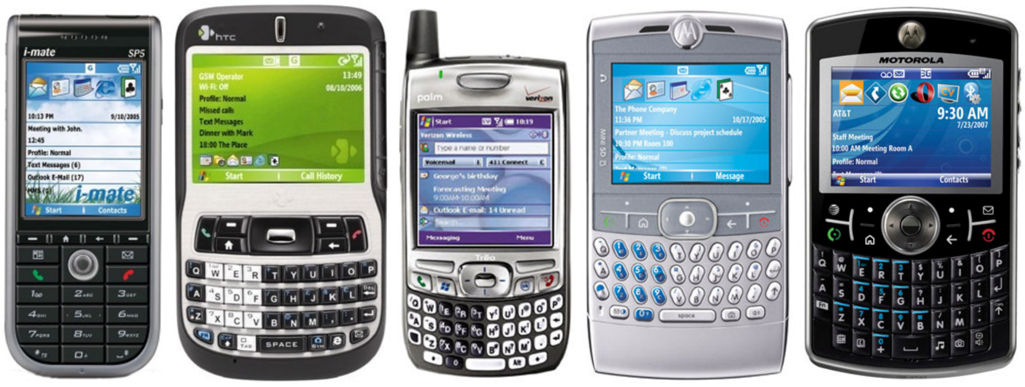


Figure 6.7: Mobile phones used for benchmarking StbES. Left to right: HTC Tornado, HTC Excalibur, Palm Treo700W, Motorola Q, Motorola Q9. (images property of www.pdadb.net)

Benchmarks

On each phone several tests were run to benchmark the various parts of the StbES mainloop. Every subsequent test adds more features of the AR pipeline:

- **Empty Frame:** This test runs a mostly disabled pipeline to benchmark the overhead of StbES when not performing any AR related duties. All features listed below are activated.
- **Video Capture:** This benchmark adds video capturing to the previous test. Video is read from the DSVideoCE DirectShow wrapper library and converted into the RGB565 format, if required, but not rendered.
- **Video Render:** This test adds video background rendering. In the cases of software only rendering, this results in a simple memory copy operation, while for hardware rendering a texture has to be loaded and rendered.
- **Tracking:** In this test tracking of a single ARToolKitPlus marker is added to the pipeline. The mainloop performs all common actions, except that the scene-graph does not contain drawable objects. Hence, this test shows the top performance a device can deliver, if 3D rendering is not taking into account.
- **Cube:** This benchmark renders a lit cube with a 256x256 pixel sized texture on top of the marker (see left image in Figure 6.8). Due to the extremely low triangle count, it mainly tests the performance of the pixel stage.
- **Venus:** This test renders a detailed, textured model of the Venus of Willendorf on top of a marker (see middle image in Figure 6.8). The 3D model consists of 870 vertices in 2625 triangles (stored as a single triangle strips) and a 256x256 pixel texture.
- **Car:** This test rendered a highly detailed, lit, untextered model of a car on top of a marker (see right image in Figure 6.8). The model consists of 12 meshes and materials. In sum the model contains 25652 vertices and 27219 triangles.

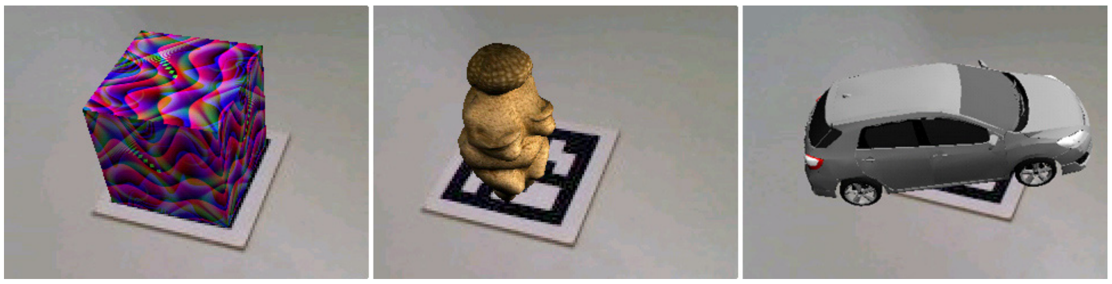


Figure 6.8: Test models rendered for benchmarking. Left: Textured cube; Middle: Venus of Willendorf; Right: Model of a car.

All tests were performed on all devices listed above. The Treo700W was benchmarked at 312MHz, as well as 520MHz to estimate the influence of pure clock rate. Table 6.3 shows the results of the tests. The table lists only frames per second (fps) values. For comparing the effect of specific stages, measurements in milliseconds would be preferred, but the tests showed that several stages influence each other: On all devices, capturing the video from the camera runs in a separate thread that competes for processing time with the main thread. On hardware accelerated devices the GPU runs in parallel too, making it extremely hard to estimate exact timings. Therefore only timings of the complete mainloop are reported.

	Empty	Video Capture	Video Render	Tracking	Cube	Venus	Car
Tornado	221,6 fps	62,4 fps	49,8 fps	27,3 fps	16,3 fps	13,2 fps	3,7 fps
Excalibur	147,0 fps	37,3 fps	29,6 fps	17,7 fps	11,2 fps	9,2 fps	2,6 fps
Treo @ 312Mhz	318,8 fps	106,6 fps	79,4 fps	40,2 fps	23,6 fps	19,9 fps	5,7 fps
Treo @ 520MHz	326,4 fps	128,2 fps	92,8 fps	48,6 fps	30,6 fps	26,5 fps	8,0 fps
MotoQ	167,3 fps	69,7 fps	53,3 fps	33,2 fps	19,4 fps	16,6 fps	4,9 fps
Q9 H/W	239,1 fps	201,6 fps	72,4 fps	45,2 fps	40,4 fps	36,4 fps	14,3 fps
Q9 S/W	145,6 fps	114,7 fps	82,5 fps	31,7 fps	19,5 fps	15,7 fps	4,2 fps

Table 6.3: Benchmarks of Studierstube ES on different mobile phones. Higher fps (frames per second) values are better.

Discussion

As can be seen in the first row, every device performed extremely well with the "empty" mainloop that renders an empty (black) rendering on the screen, but does not include video capture, video background, tracking and 3D rendering. As can be seen, the overhead of StbES is negligible: even on the slowest devices it sums up to less than 7 milliseconds.

Video capture introduces highly different workload to the phones: While the Q9 hardly looses any performance, all other devices suffer enormously from this task. The main reason for this is that the Q9 delivers images in RGB565 directly, while the other devices have to convert from YUV12 to RGB565. Although StbES contains optimized code for the format conversion, it still slows down the overall performance considerably. Yet, some devices such as the Excalibur loose more performance than others.

Rendering the video background creates inversed results to the retrieval of the video images: Here the Q9 looses most performance, which is to be expected, since it has to upload the video into a texture and then render that texture. The other devices, running software rendering only, can directly copy the camera image into the frame buffer. Hence, the Treo outperforms the Q9, even when not overclocked.

The next task added to the mainloop is tracking. Again the Q9 and Treo are far above the rest. Yet, even the Excalibur still performs at 17,7 frames per second. Taking the difference in timing between this and the previous test, one can estimate that tracking takes 22,3 milliseconds on the slow Excalibur and only 9,8 milliseconds on the overclocked Treo. As already outlined in chapter 3.4 the tracking performance scales mostly linear with the clock rate, independent of the CPU manufacturer.

The remaining three benchmarks run the full mainloop including rendering of 3D objects. All devices perform similar on the cube as well as on the Venus, despite the considerably larger polygon count of the Venus model. Obviously the 870 vertices, grouped efficiently in a single triangle strip create no big bottleneck for the software-implemented vertex stage.

Finally the car model creates a noticeable burden on all devices, including the hardware accelerated Q9. While the Excalibur and Tornado run at frame rates which are below of what is considered as real-time in AR, the other devices keep up interactive rates at least. Only the Q9 runs at almost 15 fps, which is maximum frame rate the camera can deliver.

The general performance trends become clearer when plotting all results into a chart (see left chart in Figure 6.9). As can be seen, the overall performance goes down quickly as more and more features of the AR pipeline are added. For a better comparison between the various devices, the right chart in Figure 6.9 shows the data in logarithmic scale.

All phones have enough processing power to blit the screen at 150 frames per second or more. Hence, other than a few years ago this step does not pose a noticeable bottleneck anymore on smartphones. For most devices the first big performance drop occurs when capturing video capturing to the pipeline. Yet, as already mentioned above, converting pixel formats from YUV12 to RGB565 demands lots of processing power which results in a clear performance drop on all devices that have to perform this task.

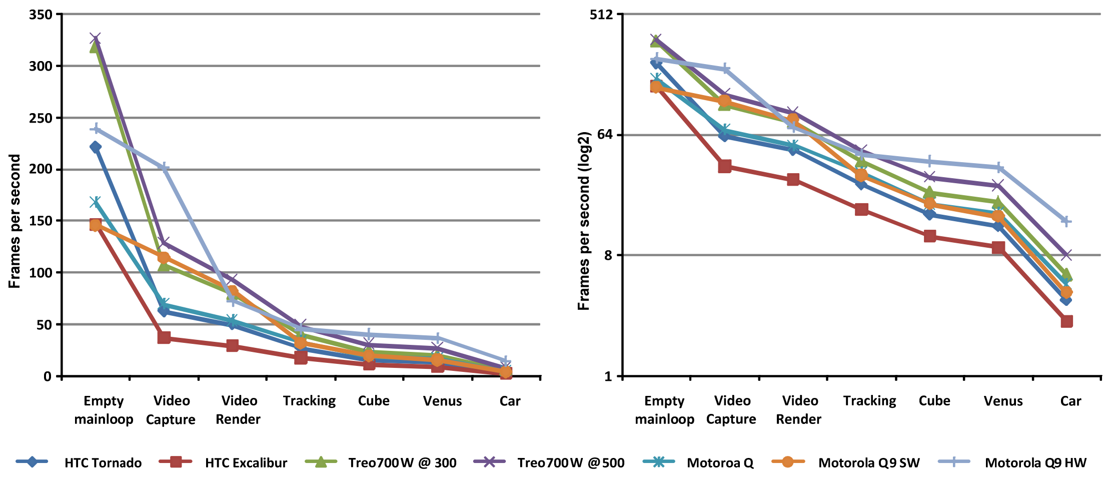


Figure 6.9: Absolute performance results. Left: Linear scale; Right: Logarithmic scale.

While the hardware accelerated Q9 could keep up very well so far it drops heavily when video background rendering is added. Here, the hardware accelerated version performs worse then using software rendering on the same device. Adding tracking costs most devices similar performance as when rendering a simple model. Only the H/W accelerated Q9 hardly looses performance with rendering, except for the car model with its extremely high polygon count.

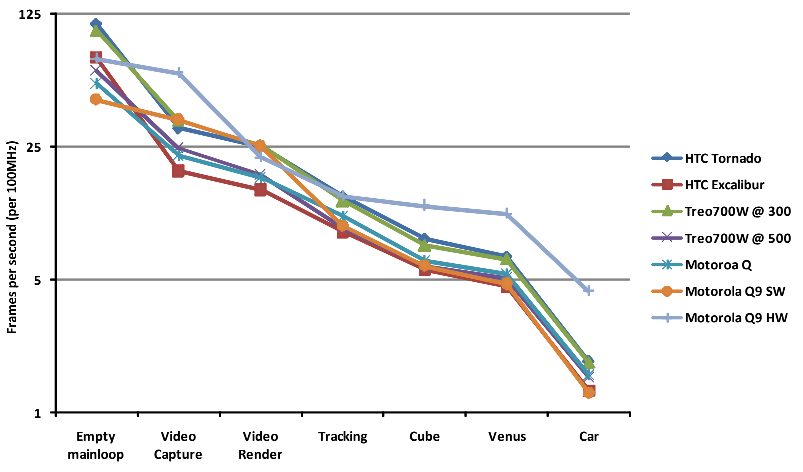


Figure 6.10: Performance results normalized to per 100Mhz CPU clock rate.

Figure 6.10 shows the same chart as Figure 6.9, except that this time all measurements are normalized to 100MHz. It is interesting to notice that some devices have a lot faster video memory access than other. Especially the Treo and Tornado devices perform far beyond all others in this category. As soon as other tasks are added, all devices perform very similar with respect to a normalized clock rate – except for the Q9 which benefits from the RGB565 video format as well as from its GPU. The chart shows clearly that without these two advantages, the Q9's CPU possesses no performance gain over the other, considerably older GPU designs.

Our tests show that even devices with similar specs can expose highly different performance characteristics. Unlike when using synthetic benchmarks that cover only a single task as reported in chapter 3.4, the performance of real life applications varies heavily. The Q9 draws a lot of performance from its hardware accelerated rendering and the 330MHz CPU. Both the MotoQ and the Treo possess an XScale processor which is well known for good performance. Yet, the Treo surpasses the MotoQ considerably in all tests. The smaller screen size of the Treo alone can not explain this difference in performance. A similar situation happens with the Tornado and the Excalibur phones. The older Tornado is almost 50% faster in all tests.

Probably the most interesting device of these benchmarks is the Motorola Q9, since it is the only one with hardware 3D acceleration, a feature that is expected to be available in most new phones released in 2008 and later. The tests show that although the video background via texturing does cost a lot of performance, it is countervailed by the enormous power when rendering high detailed 3D models.

6.4 Discussion

This chapter described Studierstube ES, a framework that combines the basic building blocks ARToolKitPlus, StbSG and Muddleware that were presented in the previous chapters of this thesis. Studierstube ES is optimized for running on mobile phones, but also works on PC-based setups too. This enables developers doing most of their work on powerful PCs, leaving only final tests to be performed on the actual client device, which speeds up development cycles enormously.

Furthermore, the Sphinx game engine for mobile AR adventure style games was introduced, which is the basis for the MARQ game, presented in the next chapter among other applications.

The Studierstube ES framework is equally powerful for developers as Studierstube (PC version), DWARF and Tinmith, etc. in the kind of applications that can be built. While this is a subjective measure, it is supported empirically by the applications presented in the next chapter and therefore a proof for hypothesis H1 and H2.

Chapter 7

Results and Evaluation

This chapter presents applications that have been developed and evaluated over the course of the handheld AR project, ranging from classical single user applications such as Signpost to content-rich multi-user programs such as the Schatzsuche game.

To demonstrate the applicability of the handheld AR approach and to evaluate the statements postulated in the beginning of the thesis, several applications have been developed and evaluated in various user studies. While the Invisible Train game was studied only informally, the feedback from its roughly 5000 users makes it as important as the formal studies run on the Virtuoso and Signpost2007 applications. The test runs of the "Schatzsuche" game, performed at the Landesmuseum Kärnten, was led by professional pedagogues who have many years of experience in creating and evaluating museum exhibitions.

7.1 The Invisible Train

The Invisible Train, developed 2003, is one of the first applications in the handheld AR project. Hence, our main concern was in evaluating the suitability of handheld AR for mass users. We therefore designed a simple multi-player game, in which players steer virtual trains on a real wooden miniature railroad track (see Figure 7.1). These virtual trains are only visible to players through their handheld's video see-through display, since they do not exist in the physical world.

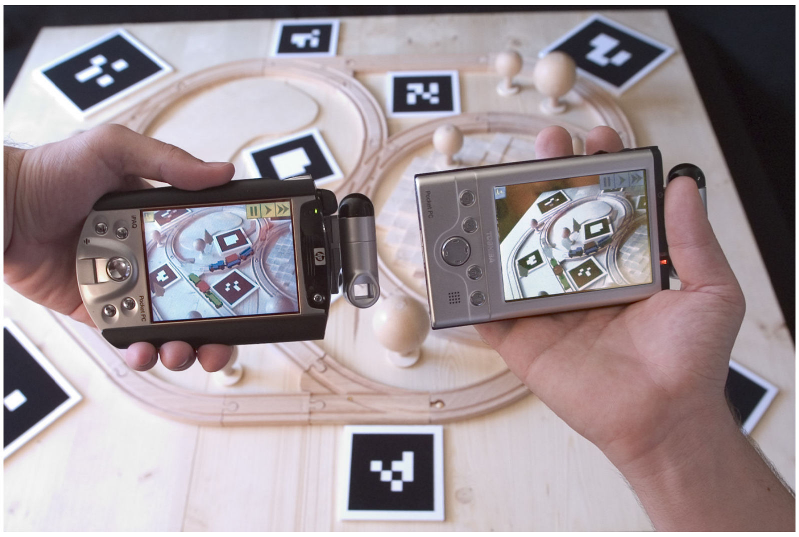


Figure 7.1: Invisible Train game board.

While it would have been possible to draw inspiration from a number of marker-based augmented reality applications published by other researchers, none of those met all of our requirements or made use of the unique possibilities gained by bringing marker-based AR to handheld devices. Naturally, we were looking for an interactive application that would allow its users to participate in a collaborative or concurrent task. The application should be distributed, synchronizing state between multiple clients through wireless networking. Many marker-based AR applications that have so far been presented make heavy use of fiducials as tangible interface components, allowing their users to flip through the pages of marker-enhanced “magic books” [17], to use markers as cards in an augmented memory game [111], or to use markers for positioning various objects such as design elements or video surfaces in the user’s workspace [16][7].

In contrast to these applications, which focus on the use of fiducial markers as moveable, dynamic parts of the application, we decided to employ the handheld’s tracked display itself as the tangible, dynamic interaction element. Therefore, we decided to focus on pen-based touch-screen input as the main interaction technique.

An important requirement was that the application should be sufficiently spatially distributed to give an impression of the properties of our tracked display surface with respect to panning and zooming interactions — users should be required to move in closely to the environment to discover important details, and to move the perspective away from the setting in order to gain an overview of the scene. This differs from other applications such as the magic book, which are designed to be fully visible within the field of view of the user, and therefore require no navigational actions from the user.

We specifically chose the game genre because we expected its playful nature of engaging in cooperative tasks would encourage users to participate in our evaluation. We deliberately left the decision whether the game should be collaborative or competitive open. As a result, the game can be played either collaboratively (trying to avoid a collision between trains for as long as possible) or competitively (attempting to crash into the other player's train). Since we anticipated people would use the application for about a minute each, we omitted a scoring mechanism and left the decision whether to cooperate or compete to the players.

Figure 7.2 shows the game's user interface elements, as seen from a player's perspective. Users are offered two types of actions: operating track switches and adjusting the speed of their virtual trains, both of which are triggered in response to a tap on the PDA's touch screen. There are two different kinds of track junctions: three-way (Y-shaped) and four-way (X-shaped) interconnections. Both are visualized through semi-transparent graphical icons floating above the physical junction element. These track switch icons serve as clickable buttons and indicate their current state and effect on train routes by their visual appearance (see Figure 7.3).

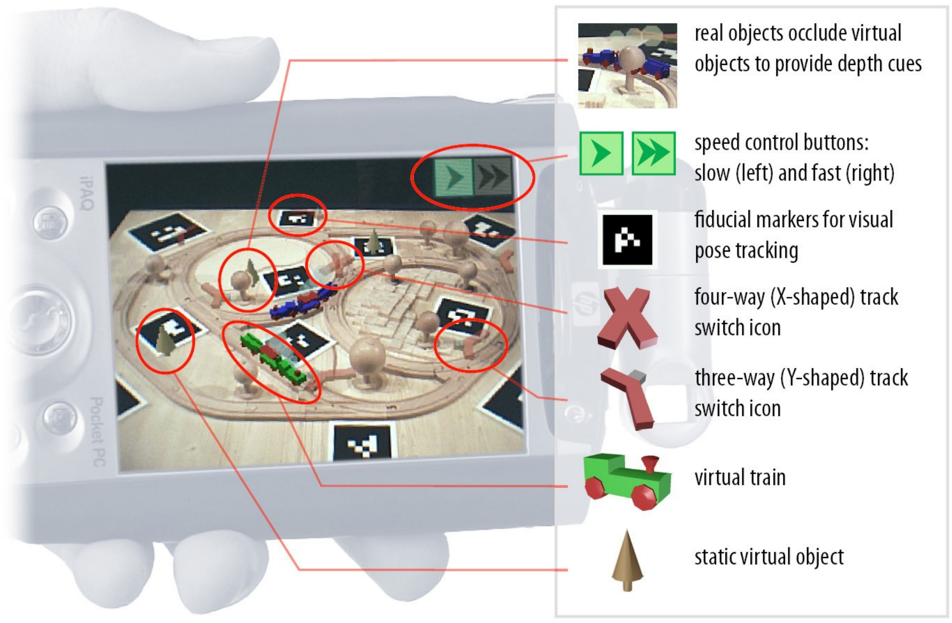

real objects occlude virtual objects to provide depth cues
speed control buttons: slow (left) and fast (right)
fiducial markers for visual pose tracking
four-way (X-shaped) track switch icon
three-way (Y-shaped) track switch icon
virtual train
static virtual object
iPAQ
Pocket PC

Figure 7.2: User interface and graphical features.

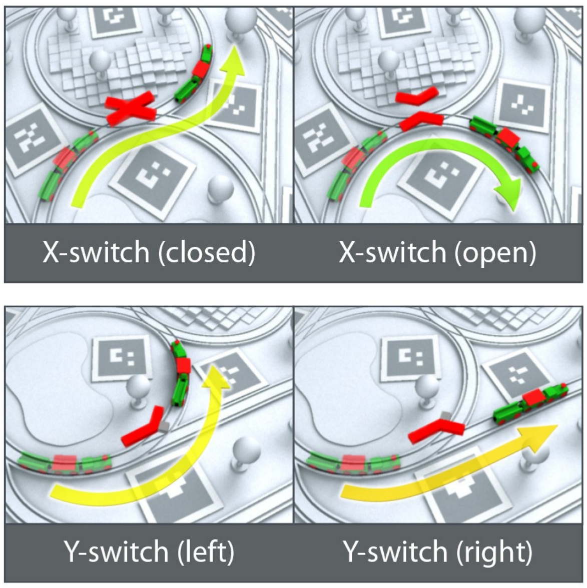


Figure 7.3: Track switch icons and their effect on train routes.

Whenever users activate a track switch, its icon turns fully opaque for one second, during which other track switch buttons become unclickable. This mechanism was primarily intended to provide users with visual feedback, but will also prevent "race conditions" where multiple users rapidly try to operate the same track switch. Users need not exercise great precision when aiming at their touch-screens: a ray-casting algorithm automatically selects the appropriate track switch depending on the closest virtual track being pointed at. Virtual trains can ride at two different speeds, which can be controlled via two dedicated buttons in the upper right screen corner. The active button is shown in color while the inactive button is grayed out. During the game, application state is constantly synchronized between all participants via wireless networking. Whenever a collision occurs, the game ends.

7.1.1 Evaluation

In order to assess the practical deployability and usability of our framework, we considered it imperative to conduct a field test in which as large a number of users as possible would be asked to try their hand at a PDA-based AR application. Ideally, most participants would not have had prior experience with AR interfaces. The application that we presented to end users in the evaluation was chosen according to several criteria: first and foremost, the application should expose our framework's major features and key properties to the end user while simultaneously allowing us to draw early conclusions about the practical value of our developments. We consecutively deployed the Invisible Train at ten different locations:

- SIGGRAPH 2004 computer graphics convention in Los Angeles (USA).
- An orientation day for incoming freshmen at Vienna University of Technology (Austria).
- A career information event for secondary school students (Austria).
- Inside the Ars Electronica Center's (AEC) "Museum of the Future" in Linz (Austria).
- ISMAR 2004 conference in Arlington (USA).
- Imagina 2005 Trade Show in Monte Carlo (Monaco).
- LEARNTEC 2005 (Germany).
- Virtual Reality 2005 conference in Bonn (Germany).
- Pervasive 2005 in Munich (Germany).
- Wired NextFest 2005 in Chicago (USA).

Over the course of these exhibitions, we gradually moved from expert audiences, who were familiar with AR technology, to a general public (see Figure 7.4) with little or no previous exposure to AR. An estimated five to six thousand visitors have engaged in playing the Invisible Train game during the four evaluation cycles, one of which lasted for over four weeks and was partially unsupervised (with occasional maintenance work done by AEC museum staff). To our knowledge, these quantities lie at least an order of magnitude above comparable informal field tests of mobile AR system, denoting the first time a mobile AR application has successfully withstood a field-test of sizeable proportions.

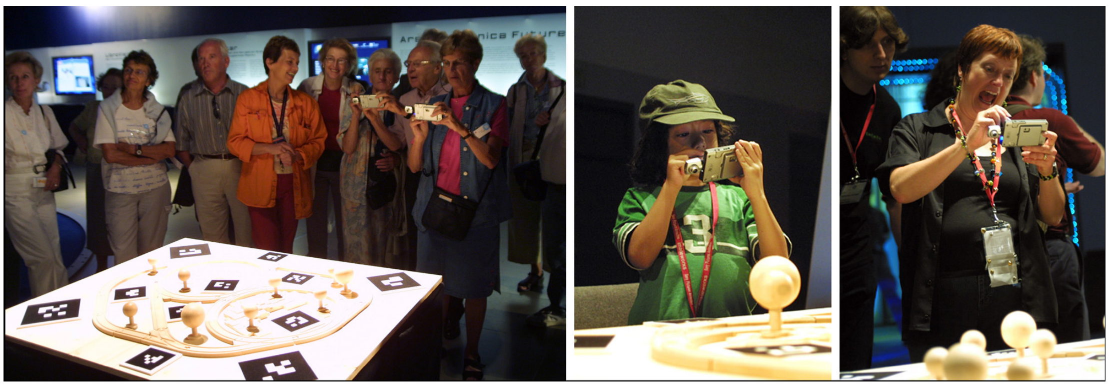


Figure 7.4: Visitors playing the Invisible Train.

7.1.2 Results

Although we did not perform a formative user study, we solicited user feedback through informal, unstructured (i.e. no specific or predetermined sets of questions were asked) interviews and conducted a summative evaluation of user performance and behavior, which led to small iterative refinements of the game's user interface. More importantly, however, we successfully completed a rigorous stress-test of our system architecture's overall robustness. Several of our empirical observations, some of which were directly comparable to our past experience involving HMD-equipped "backpack"-style setups, confirmed our assumption that handheld devices are generally more accessible to a general public, and exhibit better learning curves than traditional mobile AR systems: We found that visitors had little to no reservations towards using our system. Several participants figured out how to play the Invisible Train on their own by simple trial and error, others would learn the gameplay by looking over another player's shoulder while awaiting their turns — some children would intuitively grasp the concept and outperformed even seasoned computer science professionals.

Consequently, our supervision overhead was considerably lower than administrators of traditional mobile AR application would normally experience. On many occasions, we could observe unsupervised user experience in which visitors would pass around the PDAs while explaining the game to each other. Most participants would play at least a single game (averaging roughly 60 seconds) before handing their PDA to the next visitor. In contrast to our past experiences with "backpack" setups, we experienced almost no hardware-related failures, with the exception of a small number of application crashes, whenever users removed the add-on camera from its SDIO slot. These incidents have only further confirmed our observation that wearable devices intended for public deployment must resemble robust monolithic units without any loosely attached parts. According to user feedback, our application was considered sufficiently responsive for the intended type of spatial interaction: only a negligibly small fraction of players felt their PDA's display update rate and delay would impair their ability to play the game. We measured our system's average performance at 7 frames per second (on devices equipped with Intel's XScale PXA263 processor clocked at 400MHz, and an add-on SDIO camera from Spectec Computer Ltd), while wireless network latency was measured at about 40-50ms. Camera blur caused loss of registration during rapid movements, but was not considered a major problem.

7.2 The Virtuoso Arts History Game

Using a predecessor of our Studierstube ES handheld AR platform, we implemented Virtuoso, a collaborative educational game for up to four players. The players' objective is to sort a collection of artworks according to their date of creation along a timeline drawn on a wall-mounted billboard (see left picture in Figure 7.5).

Every marker on the timeline carries one of the artworks, which are only visible through the player's AR PDA (see right picture in Figure 7.5). Initially the artworks are in random order. A player can pick up any artwork with his or her PDA, by clicking on the artwork on the display and drop it on a free position by clicking on an empty marker on the display. While an item is located on the PDA, the player can access explanations about it.

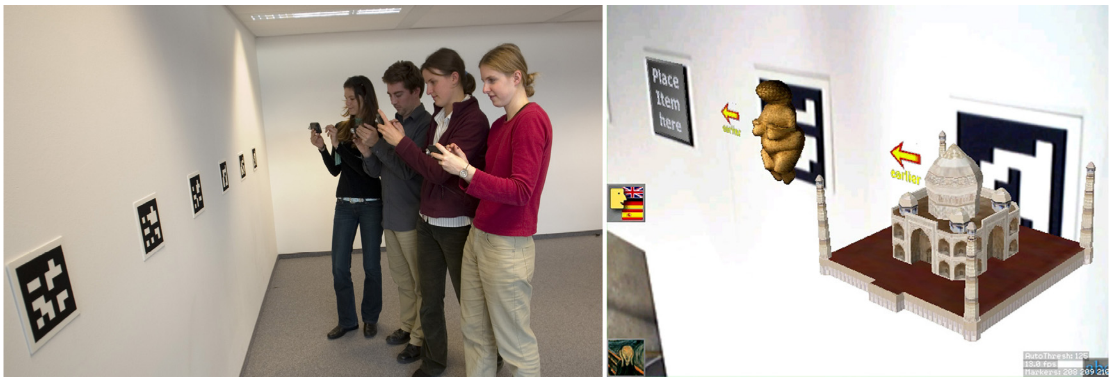


Figure 7.5: Timeline of the Virtuoso game.
Left: players using their PDAs; Right: screenshot of a player's device.

A virtual animated character called Mr. Virtuoso (see Figure 7.6) can provide help for players that are stuck. By placing the artwork on Mr. Virtuoso's desk, Mr. Virtuoso will then be prompted to provide his expertise on the subject through the use of text, audio playback and animation. The user can then take the item back, hopefully knowing by then, where to put it onto the timeline. After an artwork is placed onto its correct position on the timeline, it cannot be moved again.

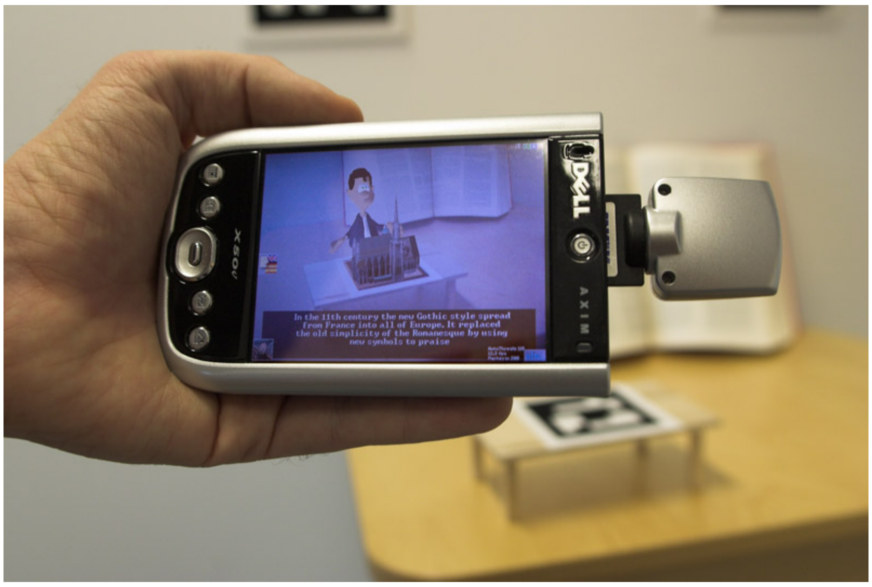

X50v
DELL
AXIM
In the 11th century the new Gothic style spread from France into all of Europe. It replaced the old sinplicity of the Romanesque by using new symbols to praise

Figure 7.6: Mr. Virtuoso giving details about an historical object.

Besides the hints from Mr. Virtuoso, the game engine can provide even more help in several ways: It can show arrows pointing "left" and "right" next to the artwork if it should be placed earlier or later on the timeline (see right picture in Figure 7.5). Furthermore the game engine can display an item's date of creation when the item is placed on the timeline. There are three configurable levels of difficulty by choosing where and when the exact date of creation of an artwork will be revealed:

- **Beginner's level**: all dates are always shown.
- **Intermediate level**: date is shown when an artwork is put into ist final position.
- **Advanced level**: dates are only revealed when all artworks are correctly placed

If the timeline is very long (more than 10 items) players can easily loose oversight. To prevent that, the game can display the timeline as a series of icons on the bottom of the PDAs' screens. The game master can enable and disable any of these options at any time during the game.

The art history application features an overall selection of 20 artworks from which the game master can select a subset for play. The game features textured, animated 3D models, multimedia background material and pre-recorded audio narration in three languages (English, German and Spanish). A graphical user interface for the game master allows runtime configuration of all game features.

7.2.1 Virtual Characters in Handheld AR

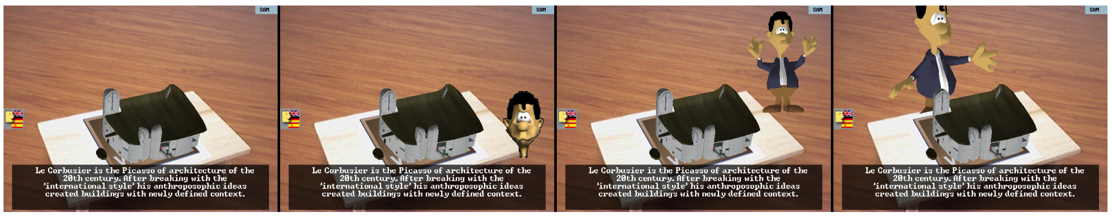
We tested different virtual character representations in the Virtuoso game to evaluate the influence of a humanoid avatar on the personal experience and learning effect of the user [110]. The key research question we were exploring was how realistic does a virtual character need to be for the user to feel engaged with it and enjoy the application, and what benefit can be derived from using AR characters.

The original art history game can be played collaboratively by up to four players. Since we wanted the subjects to focus on the learning part and the virtual character rather than on collaboration with other players, we created a modified version that was played by a single player. All help options such as arrows pointing into the direction of an item's correct spot on the timeline were turned off.

In order to explore the effect of the virtual character representation we conducted an experimental evaluation with five experimental conditions:

- **A** Text only: The virtual character is just represented by text windows appearing on the screen (Figure 7.7a).
- **B** Text and Audio: As in condition A, but in addition an audio voice over was played (Figure 7.7a).
- **C** 2D Image: As in condition B, but in addition a 2D image representing the character was shown on the screen (Figure 7.7b)
- **D** 3D Character: As in condition B, but in addition a 3D animated virtual character was shown on the screen. The character was fixed to the screen as a TV moderator (Figure 7.7c)
- **E** AR Character: As in condition D, but the 3D virtual animated character appears fixed in space in the real world (Figure 7.7d).

Condition A includes an absolute minimum presence of the virtual character. Conditions add progressively more and more realistic cues, while condition E (right-most picture in Figure 7.7) is the only case where the virtual character is seen as part of the user's physical environment, and so it is the only true AR condition. It should also be noted that the virtual head shown in condition C is not animated, unlike conditions D and E.


Le Corbusier is the Picasso of architecture of the 20th century. After breaking with the 'international style' his anthroposophic ideas created buildings with newly defined context.

Le Corbusier is the Picasso of architecture of the 20th century. After breaking with the 'international style' his anthroposophic ideas created buildings with newly defined context.

Le Corbusier is the Picasso of architecture of the 20th century. After breaking with the 'international style' his anthroposophic ideas created buildings with newly defined context.

Le Corbusier is the Picasso of architecture of the 20th century. After breaking with the 'international style' his anthroposophic ideas created buildings with newly defined context.

Figure 7.7: The five cases of Mr.Virtuoso: left-most picture (A/B): text and/or audio only; picture 2 (C): 2D image of Mr.Virtuoso's head; picture 3 (D): screen aligned and 3D animated; right-most picture (E): fully 3D registered

For the experiment, users played the game once in each condition, trying to correctly arrange four artworks each time. They were told to ask Mr. Virtuoso to learn as much as possible about the art works as they would be tested on their knowledge after each condition. The emphasis was on learning about the art rather than correctly arranging the artworks in the shortest amount of time. The order of presentation of the conditions and the artworks for each condition were changed to prevent order effects in the results.

For each condition we measured the time taken to complete the task, and we asked the users a number of questions relating to how much they enjoyed playing the game, how real they thought the character was and how much they learnt. Subjects were also asked a number of multi-choice questions about the artworks, asked to rearrange pictures of the artworks and interviewed about the experience.

The key questions we were looking to answer include:

- Is there a relationship between the character representation and perceived realism?
- Is there a relationship between character representation and enjoyment of the experience?
- Is there a relationship between character representation and how much people felt they learnt?

Knowing the answers to these questions may help developers create more effective virtual character based entertainment experiences in the future and also better understand how AR technology can be used to develop new types of characters. Chapter 9.3.1 includes the original questionnaires that where handed out to the participants.

Results

There were 13 participants of which 9 were male and 4 were female, aged 20 to 33 years. Most of the participants were native English speakers. None of them knew the game before. The experiment lasted about 40 minutes per subject including a short concluding discussion. Data analysis was performed using SPSS version 13. The main effect was tested with repeated ANOVA. If a main effect was found, pair-wise post-hoc comparisons using the Bonferroni adjustment for multiple comparisons were performed.

Subjects used Mr. Virtuoso heavily, Table 7.1 showing the average number of times used per condition. There were four artworks shown per condition, so the agent was used almost once per artwork to discover more information about art.

Cond.	A	B	C	D	E
Asked #	3.3	3.6	3.5	3.6	3.9

Table 7.1: Average Number of Times Mr. Virtuoso used

ANOVA shows no significant difference (F(4,60) = 0.26, P = 0.90) between conditions in the time taken to arrange the art works in the correct order with. Table 7.2 shows the time players played each condition. There was no significant difference across these measurements.

Cond.	A	B	C	D	E
Time (s)	248.5	230.0	236.2	237.3	252.7

Table 7.2: Time to perform task

Table 7.3 shows the average results that users got right on the four multi-choice test questions on the artwork after each condition. No significant differences were found (F(4,45) = 1.01, P = 0.41).′

Cond.	A	B	C	D	E
Score	3.0	2.8	2.5	2.3	2.6

Table 7.3: Average Number of Questions Correct

Subjects were asked to mark on a Likert scale of 1 to 7 how much they agreed or disagreed with a number of statements, where 1 = Strongly Disagree and 7 = Strongly Agree. There were a number of questions about the game and ease of use of the interface, including (see Chapter 9.3.1 for the original questionnaires):

Q1: I enjoyed playing the art history game
Q2: The PDA interface was easy to use
Q3: The task was easy to solve
Q4: I felt I learned new facts about art items from the game

Table 7.4 shows the average results for each of these questions. As can be seen there is little difference between conditions for these results. ANOVA tests found no significant differences for the user survey scores for these questions.

Condition	A	B	C	D	E
Q1	5.46	5.85	5.92	5.85	5.85
Q2	6.08	5.92	5.92	6.08	6.15
Q3	5.69	5.69	5.84	6.00	5.92
Q4	5.39	5.85	5.92	5.84	6.15

Table 7.4: Subjective Survey Scores

A second set of questions related to the virtual character:

Q5: Mr. Virtuoso seemed real to me
Q6: Mr. Virtuoso was helpful for completing the task
Q7: Mr. Virtuoso improved the overall experience
Q8: I found Mr. Virtuoso to be friendly
Q9: Mr. Virtuoso seemed to be part of the real world

There was a significant difference between the results for all of these questions. Figure 7.8 shows the average results for Q5: *Mr. Virtuoso seemed real to me*. As the virtual character exhibits more visual and audio cues the subjects felt that it was real. An analysis of variance was conducted with type of virtual character (A - E) as the within-subjects factor. Doing this we found a significant difference between conditions (F(4,48) = 11.18, P < 0.001). Post-hoc found that Mr. Virtuoso in conditions E (P < 0.001) and D (P < 0.001) was rated as significantly more real that in condition A. Condition D was also rated significantly higher than condition A (P<0.01). There were no other significant differences between conditions.

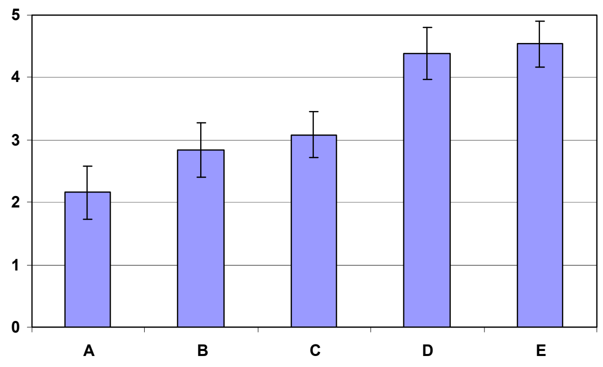


Figure 7.8: How real Mr. Virtuoso was (Q5)

Table 7.5 shows the average results for Q6, Q7 and Q8. An ANOVA on Q6: *Mr. Virtuoso was helpful for completing the task* showed a significant difference across conditions (F(4,48) = 8.186, P < 0.001). Post-hoc comparisons between conditions found a significant difference between conditions D and E and condition A (P<0.05). Similarly, an ANOVA on Q7: *Mr. Virtuoso improved the overall experience* produced a significant difference across conditions (F(4,48) = 5.22, P < 0.01). Finally an ANOVA on Q8: *I found Mr. Virtuoso to be friendly* produced a significant difference across conditions (F(4,36) = 12.322, P < 0.001). Post-hoc comparisons found a significant difference between conditions D and E and condition A (P<0.01), and conditions B and A (P<0.05). In all cases the score of the condition without audio (condition A) was lower than the other conditions, while the two conditions with 3D graphics (conditions D and E) were the highest.

Cond.	A	B	C	D	E
Q6	3.46	4.92	4.85	5.69	5.85
Q7	3.23	4.69	4.31	5.15	5.39
Q8	2.10	4.50	3.90	5.20	5.20

Table 7.5: Helpfulness, Experience, and Friendliness

As the quality of the character representation increased it also seemed to be more part of the real world. Figure 7.9 shows a graph of the average response to question 9: Mr. Virtuoso seemed to be part of the real world. An ANOVA showed a significant difference across conditions (F(4,36) = 6.46, P < 0.001). Post-hoc comparisons found a significant difference between conditions C and A (P<0.05), conditions D and A (P<0.05), and conditions E and A (P<0.05). As before the 3D virtual characters (D, E) are significantly different from the text-only condition (A).

In addition to providing subjective survey responses, subjects were also asked to rank each of the conditions in order according to the following criteria. For each criteria 1 = lowest, 5 = highest.

R1: How real Mr. Virtuoso seemed
R2: How much fun it was
R3: How much you learnt
R4: How helpful was Mr. Virtuoso

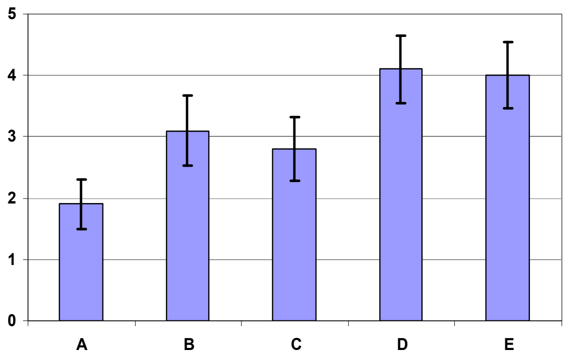


Figure 7.9: How much Mr. Virtuoso seemed part of the real world.

The rankings for R1 and R2 are significantly different across conditions. An ANOVA for R1 finds (F(4,44) = 67.42, P < 0.001). Post-hoc comparisons show significant difference (P<0.02) between all conditions except B and C. In fact, all of the subjects except one ranked the AR condition as most real. Similarly, an ANOVA for R2 found F(4,48) = 30.25, P < 0.001. All but two of the subjects ranked the AR condition either highest or second on how fun it was. Figure 7.10 shows the results for rankings R1 and R2. In this case, when users where forced to chose, as the virtual character had more realistic characteristics they thought it was more real and correspondingly more fun.

However there was no significant difference between rankings on R3: How much you learnt. An ANOVA finds (F(4,32) = 0.80, P = 0.53). This is consistent with the survey results for Q4 and the multi-choice question results. Table 7.6 shows the average rankings.

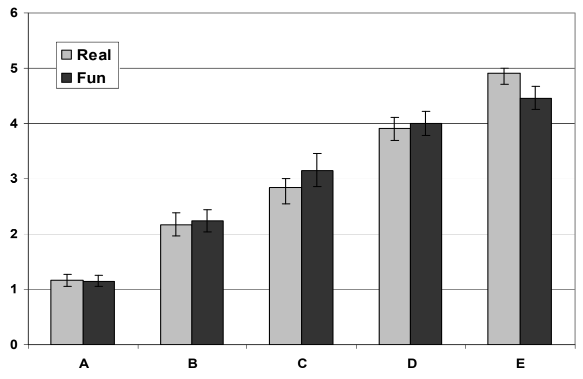


Figure 7.10: Average ranking on realism and fun

For ranking R4: How helpful Mr. Virtuoso was, an ANOVA produced a significant difference across conditions (F(4,36) = 3.78, P < 0.01). Post-hoc comparisons confirm that this is because of the difference between the condition with no audio (Condition A) and the other conditions.

Cond.	A	B	C	D	E
R3	2.54	3.67	3.19	2.55	2.91
R4	1.93	3.15	3.50	3.23	3.23

Table 7.6: Ranking of Learning and Helpfulness

Interviews

In the interviews with the subjects several consistent themes emerged. Although we emphasized before and during the game that time was of no importance, 4 out of 12 users complained afterwards that they felt slowed down by Mr. Virtuoso, particularly in condition E where they needed to wait for his walking animation to finish. The main reason for this that Mr. Virtuoso presented information in his own speed and could not be interrupted.

Most subjects pointed out that they were very aware of the fact that the more feature-rich versions of Mr. Virtuoso did not provide more information then the other versions. Still they usually liked the AR version more because they felt it looked more natural and realistic. However many subjects pointed out the importance of Mr. Virtuoso's voice feature as being critical.

Three subjects complained that Mr. Virtuoso did not make eye contact with them while speaking which made them feel uncomfortable or even offended them. Condition D (screen-aligned) did not have this problem since the screen-aligned character implicitly looked in the player's direction. Those three subjects generally preferred condition D over E.

One subject did not like the AR version because he had the feeling that he was interrupting Mr. Virtuoso from his other actions when asking him for assistance. This was not an issue with the screen-aligned version because Mr. Virtuoso in this situation walks into the screen from the right side and is therefore not visible before.

Finally, two players felt that the animated character distracted them from reading the text and listening to the voice.

Discussion

The main focus of our study has been to explore the effect of different virtual character representations on user engagement, enjoyment and educational outcomes in a learning task. The objective measures of time to complete the task and number of correct test questions did not vary significantly across conditions, showing that the various character representations did not have any effect on the educational outcomes. Similarly when the users were asked their subjective opinion of the different conditions there was no difference in how much they enjoyed each condition or how difficult they found the task and how much they felt they learnt. However, when forced to rank the conditions in order, as the virtual character became more realistic the users felt the condition was more fun (Figure 7.10).

This is not too surprising since users were engaged in a relatively short task and as the virtual character representation becomes increasingly realistic it did not provide additional educational content, such as gesturing to specific parts of the virtual artwork. If the task had a greater spatial learning element (such as learning how an engine is taken apart) the more realistic characters could have an impact on the learning outcomes.

The increasing enjoyment ranking when forced to make a choice may be due to the novelty factor; AR characters are more novel than a disembodied voice over and so relatively more fun.

The subjective results in response to the character did show significant differences across conditions. When asked how real Mr. Virtuoso seemed and how much he seemed to be part of the real world, the main differences was between 3D and non-3D representations of the character and audio vs. non-audio. In all cases the non-audio condition rated lowest while the 3D character was the highest, but there was no difference between AR and screen aligned characters.

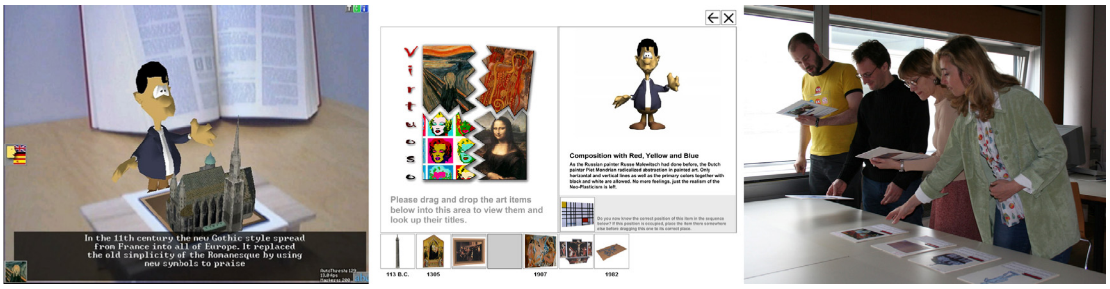
It is interesting to note the effect that adding audio can have on the user's perception, causing a large jump in average scores between conditions A and B, while there was no additional benefit of adding a 2D representation to the audio. If the 2D character had been animated this may have had a greater effect.

Players did not rate the AR version more realistic than the screen-aligned 3D virtual version and it was not perceived to be more helpful or friendly than the 3D virtual version. Only when forced to choose did subjects rank the AR version more realistic and more fun.

One of the reasons for this could be that the AR character did not exhibit any more communication cues than the screen-aligned 3D virtual character that would make it seem more real. Although he walks around the real table, the AR version of Mr. Virtuoso did not give any spatially related information on the objects such as pointing to specific spots on items as a real person would do while explaining. Furthermore a realistic person is expected to behave politely and to look at the people he is talking too.

7.2.2 Collaborative Edutainment

Handheld Augmented Reality is expected to provide ergonomic, intuitive user interfaces for untrained users. Yet until now only few comparative studies have evaluated these assumptions against more traditional user interfaces. We therefore compared the suitability of our Virtuoso arts history learning game against its non-AR variants on paper and on a desktop PC [116].


In the 11th century the new Gothic style spread from France into all of Europe. It replaced the old simplicity of the Romanesque by using new symbols to praise

Please drag and drop the art items below into this area to view them and look up their titles.

Composition with Red, Yellow and Blue

As the Russian painter Russe Malewitsch had done before, the Dutch painter Piet Mondrian radicalized abstraction in painted art. Only horizontal and vertical lines as well as the primary colors together with black and white are allowed. No more feelings, just the realism of the Neo-Plasticism is left.

Do you now know the correct position of this item in the sequence below? If this position is occupied, place the item there somewhere else before dragging this one to its correct place.

Figure 7.11: Three variants of the Virtuoso game: Augmented Reality, desktop and paper

The desktop PC version was programmed in Macromedia Flash (see middle image in Figure 7.11). Players can move artworks on the timeline displayed on the screen using drag and drop operations with the mouse. Consequently the PC version can only be operated by one player at the time. While this restricts the user interface of the game, it provides a realistic example of typical PC-based edutainment applications or museum installations. As usual, when an item is located at its final position its date of creation is displayed below of it and the item can no longer be moved. Moving an item to the left-top pane provides basic information, while the animated Mr. Virtuoso on the top-right pane provides detailed explanations in text and audio. Items can be directly dragged from one slot on the timeline onto another.

For the paper version of the game we printed pictures of the artworks on playing cards (see right image in Figure 7.11). On the front, the playing cards show the introductory text, while on the back more detailed descriptions by Mr. Virtuoso are provided. Players are only allowed to hold one playing card at a time and must only put it back into a free position. While the computer can enforce the game rules for the AR and desktop version we introduced a human game master to the paper version who takes care that the game's rules are not violated. The game master will also reveal an item's date of creation when an item is located on its correct position on the timeline. Consequently this version of the game has the same functional characteristics as the handheld AR and desktop PC versions.

Experimental Evaluation

We compared our collaborative AR application with the two non-AR variants of the game (the desktop PC and paper-based version). In the experiment participants were grouped into teams of four. In each game, players had to sort seven items on the timeline. After a team finished its task they filled out a questionnaire about the game they just played including detailed art history questions about the items they just arranged, and how they felt about the game interface. Then all teams moved on to another version of the game. After the participants played all three versions of the game they filled out another questionnaire asking to rank the conditions in several categories. The introductory instructions to the participants emphasized the focus on collaboration and the need to learn about the artwork items. Users should learn rather than completing the task as fast as possible. To further motivate cooperation between players, players were told, that the goal of the game was to get a high team score, rather than personal scores on the arts history questions they had to answer.

There were 48 participants 26 female and 22 male, aged from 20 to 43. 25 people stated that they had never used a PDA before. The experiment lasted about one hour for each subject including introduction and a short finishing discussion. Data analysis was performed by using SPSS version 13 and the main effect was tested with repeated ANOVA. If a main effect was found, pair-wise post-hoc comparisons using the Bonferroni adjustment for multiple comparisons were performed. The questions the participants had to answer after each game can be grouped into four main categories: collaboration, easiness of the task, learning effect and fun factor. Subjects were asked to mark on a Likert scale of 1 to 7 how much they agreed or disagreed with a number of statements (1 = Strongly Disagree and 7 = Strongly Agree. Chapter 9.3.2 includes the original questionnaires that were handed out to the participants.

Results

We asked two questions about the way people collaborated:

- Q1: I collaborated intensively with my team members.
- Q2: I knew exactly what the others were doing.

Table 7.7 shows the average results for each of these questions. Subjects felt that they collaborated more in the Paper and PDA versions; an ANOVA test found a significant difference for Q1 (F(2,94)=3.94, P<0.023) and a post-hoc comparison found a significant difference between the PC game and the other two variants. Similarly, an ANOVA for Q2 found a significant difference between how well subjects felt they knew what the others were doing: F(2,94)=6,13, P<0.003. A post-hoc comparison found a significant difference between the PDA condition and the PC and paper versions of the game.

Condition	Paper	PC	PDA
Q1	5.71	5.00	5.61
Q2	5.67	5.75	4.73

Table 7.7: Average results on collaboration

In the category ease of the task we asked the following five questions:

- Q3: I always had a good overview of the timeline
- Q4: I had sometimes problems with the user interface
- Q5: The game was sometimes confusing
- Q6: The user interface was easy to use
- Q7: The task was easy to solve

Table 7.8 shows the average results. As can be seen, there is little difference for the conditions of the questions Q3, Q4 and Q5. An ANOVA test found significant differences for Q6: *The user interface was easy to use* (F(2,94)=5.27, P<0.007). A post-hoc comparison showed that the paper variant was rated significantly lower than the PC version and there was no difference between the PC and PDA conditions. For Q7: *The task was easy to solve*, ANOVA found significant differences (F(2,94)=3.52, P<0.034), and a post-hoc comparison showed that the PC version was rated significantly easier than the PDA version, but there was no significant difference between the other conditions.

Condition	Paper	PC	PDA
Q3	5.10	5.27	4.81
Q4	2.27	2.21	2.69
Q5	2.56	1.98	2.65
Q6	5.38	6.27	5.86
Q7	5.60	5.94	5.44

Table 7.8: Average results for ease of use.

To measure if people felt a learning effect we asked the question:

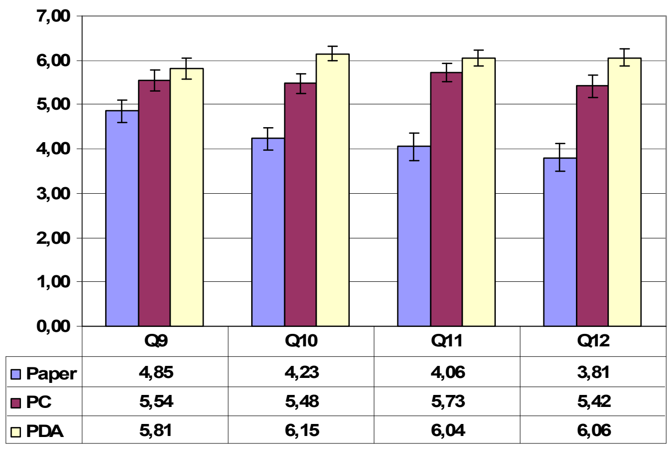
- Q8: I believe I learned something about those artworks

Performing an ANOVA on Q8 did not find any significant differences. The last group of questions we asked after each game was about how much people liked the game and how much it would fit into a museum setting:

- Q9: I enjoyed playing the game
- Q10: Playing the game was a great experience
- Q11: This game would fit well into a museum exhibition
- Q12: I would like to play this game in a museum

Figure 7.12 shows the average results for each of these questions. There were significant differences between the results for all of these questions. As can be seen for every question the PDA version scored highest while the paper version was rated lowest. An ANOVA was conducted and Q9: *I enjoyed playing the game* resulted in F(2,94)=5.472, P<0.006. Post-hoc analysis found that the PDA version was rated significantly higher than the paper version. An ANOVA for Q10: *Playing the game was a great experience* resulted in F(2,94)=32.916, P<0.001. Post-hoc analysis showed that the results for all three conditions were significantly different. For question Q11: *This game would fit well into a museum exhibition* the PC and PDA version got very similar ratings. An ANOVA (F(2,94)=25.713, P<0,001) including a post-hoc analysis showed significant differences between the paper and the other two conditions but no differences between the PC and PDA conditions. Finally for Q12: *I would like to play this game in a museum*, an ANOVA resulted in F(2,94)=30.716, P<0.001. Post-hoc analysis found all three versions of the game were significantly different.

In general subjects thought the PDA version provided a greater experience than the other two conditions and they would like to play this in a museum more than the other two games.



	Q9	Q10	Q11	Q12
▢ Paper	4,85	4,23	4,06	3,81
■ PC	5,54	5,48	5,73	5,42
▢ PDA	5,81	6,15	6,04	6,06

Figure 7.12: Average results for Questions Q9 - Q12

In addition, subjects were also asked to rank each of the conditions in order according to the following criteria. For each criteria 1 = lowest, 3 = highest.

- R1: How easy the game was to play
- R2: How much you learnt
- R3: How good the overview of the timeline was
- R4: How much you collaborated
- R5: How much fun the game was
- R6: How much the game would improve a museum exhibition

Condition	Paper	PC	PDA
R1	1.89	2.25	1.86
R2	1.82	2.09	2.09

Table 7.9: Average results for rankings R1 and R2

The rankings for R1 and R2 did not produce significantly different results. As can be seen in Table 7.9, all conditions were ranked very closely. However, for ranking R3: *How good the overview of the timeline was*, an ANOVA found F(2,90)=4.723, P<0.011. A post-hoc analysis showed that the PC and PDA conditions were significantly different giving the PC version the best score of overview of the timeline (see Table 7.10). Interestingly, ranking R4: *How much you collaborated*, resulted in exactly the opposite ratings. Here the PC version scored significantly lower, while the paper and PDA conditions were rated almost equally well. An ANOVA plus post-hoc comparisons resulted in F(2,88)=4.006, P<0.022 and found significant differences between the PC condition and the other two conditions.

Condition	Paper	PC	PDA
R3	1.89	2.35	1.76
R4	2.15	1.67	2.13

Table 7.10: Average results for rankings R3 and R4

Finally, R5: *How much fun the game was* and R6: *How much the game would improve a museum exhibition* produced the results shown in Table 7.11. An ANOVA found F(2,92)=43.607, P<0.001 for R5 and F(2,88)=31.253, P<0.001 for R6. For both R5 and R6 post-hoc comparisons showed that all results were significantly different resulting in the PDA version being ranked as the most fun and most appropriate for a museum exhibition.

Condition	Paper	PC	PDA
R5	1.36	1.89	2.75
R6	1.36	2.00	2.64

Table 7.11: Average results for rankings R5 and R6

Interviews

We interviewed the participants after each condition and several consistent themes emerged. For the paper version, subjects felt that they needed to organize themselves to prevent chaos, which was not a problem in the electronic versions of the game where strict rules were implicit. While many players said that it felt good to have a physical object in their hands they also added that the paper version was very "old school". In general subjects felt the paper version was less appealing.

Although most participants rated the PC version as providing the best overview of the timeline in the questionnaires, some participants complained that too many items crowded the screen which confused them. Interestingly, the same audio recordings for Mr. Virtuoso's voice-over was used in the PC and PDA version, but several users commented that the PC version's virtual character sounded more pleasing. We assume the reason for this is the low quality of the PDAs' built-in speakers. Subjects told us that collaboration was most difficult with the PC version because there was only one mouse to use and every action had to be first discussed with the other players. Players sitting more distant from the PC screen usually participated the least.

Some subjects said that the PDAs' touch-screens were more difficult to use than the mouse interface which is expected since most subjects had never used a PDA before, and people were afraid to break the PDA, especially due to the attached camera. All participants complained that Mr. Virtuoso should speak louder which is a well known problem with PDA speakers. Participants noted that the small screens could not be seen by the other players so collaboration was more difficult than with the paper version. Mr. Virtuoso was identified as a bottleneck for the game progress because other players would have to wait until Mr. Virtuoso had finished describing the artwork. People thought the user interface and the graphics in general were very appealing although some participants argued that it was difficult to explore the 3D artworks on the small screen. As most users had only minimal computer science experience, they were very excited due to the high-tech feeling of the PDA game and commented that the handheld AR concept was "innovative" and "ingenious".

Discussion

Although we tested three different game conditions, there was no difference in the educational outcomes. This could be because the learning task was essentially a memory task that wasn't dependent on effective collaboration or the ease of use of the interface. However, there were significantly different user subjective results as a consequence of the different characteristics of each condition.

One of the most obvious differences between the conditions is in how space was used. In both the AR PDA case and the paper interface the art pieces were spread out in physical space allowing the four users to work on the game in parallel. This shows one of the advantages of AR, namely that it allows virtual content to move from the screen and into the real world. In contrast, with the PC interface the users are working on a much smaller screen with only a single input device. In this case it was impossible for users to manipulate objects at the same time. Thus users felt that both the AR PDA interface and the paper version allowed them to collaborate more effectively than the PC interface.

Another key difference between the interfaces was in how much awareness they provided of what the others were doing. In the PC and paper versions all of the users could see all of the art pieces on the timeline at the same time. When a player moves a piece of artwork, everyone is aware of it. In contrast, in the AR PDA application, each of the users had a personal view of the virtual content, and unless they shared their PDA, they were not aware of which players had picked up which art pieces. One of the challenges of designing collaborative AR systems is providing independent views of virtual data while at the same time creating shared group awareness.

Despite the different interface conditions, there was not a significant difference in usability. Although the users had never used an AR PDA interface before, in general they found it relatively easy to use; as easy as using the mouse-based PC interface and manipulating real cardboard pictures. This is unusual for first time users of a novel interface, but is due to using an intuitive interface metaphor. In this case a magic lens metaphor in which the AR PDA becomes a virtual window on the real world. Users are able to view the virtual scene as easily as if they were using a real handheld lens.

Users ranked the PDA interface as the most enjoyable of the three conditions and the one that they would most like to see in a museum. The fun factor may be due to both the novelty and visually appealing nature of the AR interface. The AR condition provided 3D virtual imagery, animation, sound and text.

From these results we can infer several design guidelines for handheld AR interfaces that can inform future applications:

- Allow the user to experience the virtual content in space.
- Use an appropriate interface metaphor, such as a lens input metaphor.
- Seamlessly integrate 3D virtual imagery with animation, 2D images and text to create a multi-sensory experience.

In general, in a face to face collaborative AR interface, key elements of normal human face to face must be considered. This includes providing a mechanism for sharing user views to establish shared understanding, enabling users to work in parallel, and preserving the ability to share verbal and non-verbal face to face communication cues.

7.3 Museum Augmented Reality Quest at Technisches Museum Wien

Before we created the "Schatzsuche" game at the Landesmuseum Kärnten (see chapter 7.4) we developed a prototype that was situated in the permanent exhibition *medien.welten* at Technisches Museum Wien (TMW). The theme of this exhibition is the history of media storage and transmission. The exhibition illustrates how different media were preserved and transmitted since antiquity.

Museum Augmented Reality Quest (MARQ) encompasses a selection of exhibits and links them into an exciting story. The objective of the game is to solve a quest composed of puzzles and other tasks associated with the exhibits. The target groups are classes of teenagers at the age of 12 to 16.

Tasks vary widely depending on the nature of the exhibit and the kind of knowledge to be mediated. Interactions with mobile AR applications are supplemented by simpler techniques, like displaying classical 2D interfaces on the PDA. This allows integrating classical e-learning methods such as multiple choice questions, which are more rapidly produced.

A noteworthy type of mixed reality task is interaction with the instrumented hands-on exhibits. These exhibits are tangible interfaces, specifically designed to explain certain technologies: For example, the Morse exhibit (see left and middle pictures in Figure 7.15) allows a user to input a character using an old-style push button, and displays the corresponding letter if one is recognized. These hands-on exhibits are computer controlled and can be set to present a certain task or operating mode when approached by a MARQ player. In that way, the environment is responsible to the player in ways beyond the through-the-lens AR experience.

A first version of the game focused on three exhibits which have been embedded in a small espionage story set in World War II. The exhibits have to be visited by the players in a certain sequence to achieve the game objectives. They start from the checkpoint, where the quest is introduced and the PDAs are handed out (see left image in Figure 7.13). A group of players with one PDA usually consists of two or three students.

The screens of the checkpoint and PDAs show a map of the exhibition, highlighting the relevant task locations. The map also indicates the current position of the players and lists already solved and remaining tasks.

7.3.1 MARQ Hotspots

The first task is a radio direction finder used at wartime to detect and record radio messages from mobile transceivers. The operator had to manually turn the antenna to home in on the signal and then follow it to record it. A special electronic guide was aiding this task by producing characteristic sounds when turning the antennas near the exact signal direction.

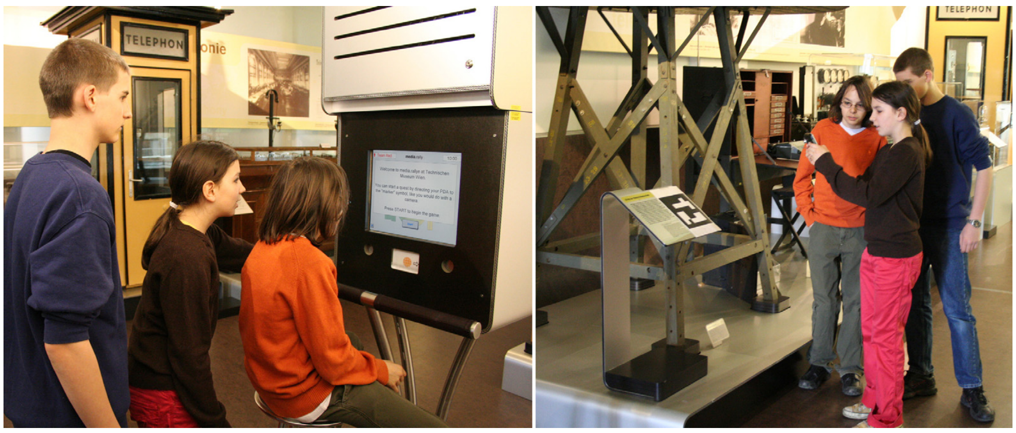


Figure 7.13: Left: Checkpoint, where the quest starts;
Right: Recording the message at the radio finder

The radio finder game (see right picture in Figure 7.13) developed for MARQ uses an AR application running on a PDA. The PDA has to be physically moved around the exhibit to find and hold the exact signal direction. Looking through the PDA the players see a virtual compass superimposed onto the lower platform of the real exhibit (see left image in Figure 7.14). The PDA also plays sounds indicating the deviation from the exact signal direction. The sound depends on the direction and the angle of deviation. The interval between beeps gets shorter when the PDA gets closer to the signal location. When entering a small angular interval around the exact signal direction the characteristic beeps of a Morse message are played, that are easily distinguishable form the homing sounds. Once the players have found this window the PDA must stay in it until the entire message has been received. A progress bar on the PDA shows the percentage of recorded data.

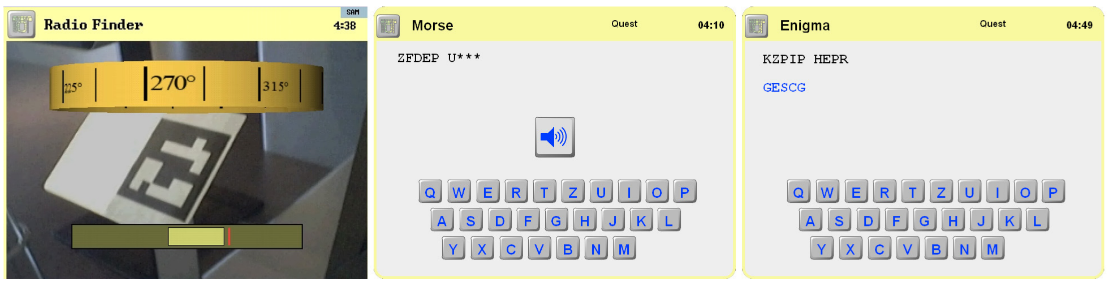


Figure 7.14: In-game screenshots. Left: Radio Finder hotspot; Middle: Morse hotspot; Right: Translating the Morse code at the Enigma hotspot.

The game story explains that the message was sent in Morse code. Thus for the next task the players must visit the hands-on exhibit presenting the history of Morse codes. It features a tangible interface - a replica of an old-fashioned telegraph pushbutton for typing Morse codes (see left and middle pictures in Figure 7.15).

The players are asked to translate the Morse code received at the radio direction finder into text. For that purpose the Morse exhibit is automatically switched to input mode: Every character input via the pushbutton is immediately translated into the corresponding letter on the terminal's screen (see middle image in Figure 7.14).

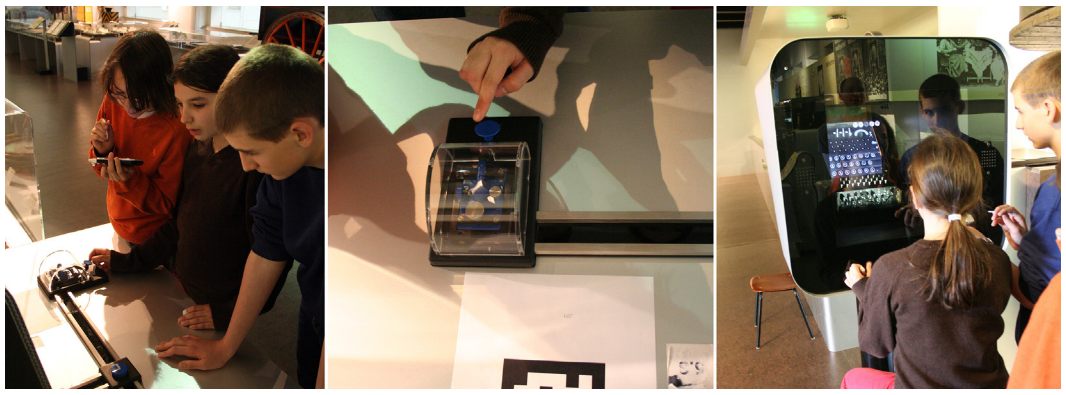


Figure 7.15: Left: Translating the Morse code at the Morse station, an interactive exhibit of the museum; Middle: The tangible interface of the Morse station, an old-style push button; Right: The virtual show case containing the Enigma

A Flash application on the PDA shows the previously recorded Morse code. The interface also contains a play button replaying the Morse code, and a virtual keyboard for entering the translation. The message has to be translated letter by letter. The player can listen several times to the sound of the actual Morse code. Once they have identified the combination of short and long beeps, one student types them into the interactive exhibit with the pushbutton interface and then reads the translated letter from the display. The person operating the PDA uses the virtual keyboard to enter the letter, and then the next Morse code is presented. This process is intended to encourage the collaboration between the players.

When the players have completed this task, they learn that the translated message makes no sense at all but looks like a random gibberish of letters. This is because the message was encrypted by the Enigma machine.

They have to move to the Enigma hands-on exhibit (see right picture in Figure 7.15) to decrypt the message. The Enigma was used by the Germans during World War II to encrypt messages prior to transmission. The exhibit shows a real Enigma embedded in a Virtual Showcase [18], a mixed reality display combining real artefacts (in this case, a real Enigma machine) with projected imagery through mirror optics

Using a trackball interface, the visitors can operate the virtual overlaid Enigma without touching the real one.

When the team arrives at this exhibit the Enigma exhibit switches to free decryption mode (normal visitors have to run through a story, where fixed messages have to be decrypted). The players first have to set the day key on the Enigma as instructed by the PDA. Then they decrypt the message letter by letter. One player operates the Enigma while another types in the plaintext into the PDA using a virtual keyboard (see right image in Figure 7.14). The two user operation corresponds to the way an Enigma was actually operated in the field.

After solving all assigned tasks or running out of time, the quest game is over and the players return to the checkpoint, where the results of their performance are displayed. The screen shows where mistakes occurred and the percentage of the message that was revealed.

7.3.2 Integrating of Hands-on Exhibits

Figure 7.16 shows the integration of hands-on exhibits and embedded terminal PCs ("Checkpoint Terminal") into the MARQ game at the Technisches Museum Wien. Flash is used intensive on various parts of the distributed application: On the mobile clients it drives the 2D parts of the game (see next chapter), while on the terminal PCs it shows the team's overview map and game state.

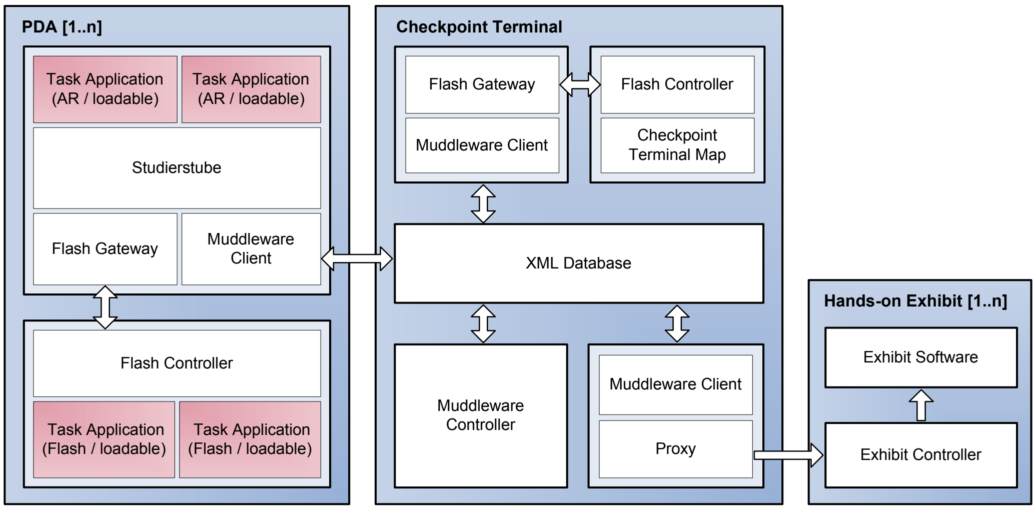


Figure 7.16: Integration of terminal PCs and hands-on exhibits into the MARQ prototype developed for the Technisches Museum Wien

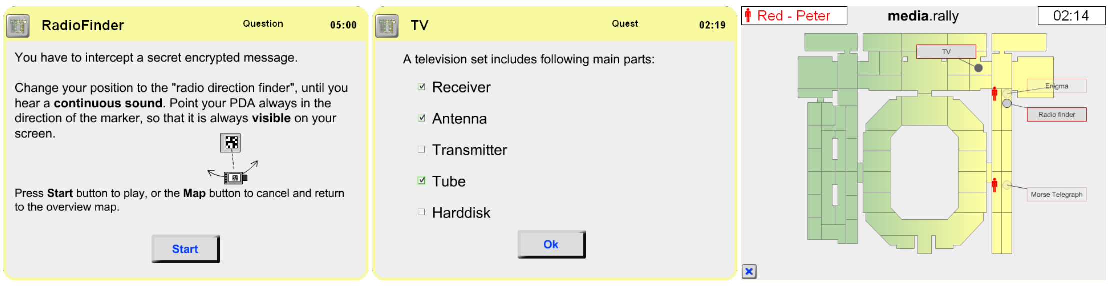
On the handhelds it is integrated into Studierstube. It communicates with Studierstube using an the “Flash Controller” an ActionScript class that talks to the “Flash Gateway”, implemented as an application inside Studierstube, which directly executes local commands and forwards the remaining operations to the Muddleware server.

The “Checkpoint Terminals” are PCs integrated into the museum that usually run the Museum’s information browser. For the game the terminals were modified to run the server as well as another Flash application showing the team’s overview map and game state. Again this Flash application uses the Flash Controller and Flash Gateway to talk to the Muddleware server.

Hands-on exhibits are installations that are controlled by PCs and allow users to interact with the exhibition items. Other than the terminal PCs, the PCs of the hands-on exhibits are already prepared to put the exhibits into a special “free” mode. It was therefore sufficient to develop a Muddleware client that acts as a bridge between the game server and these PCs.

7.3.3 2D User Interface in MARQ

Besides the story-driven tasks, MARQ also features a number of multiple choice questions related to real exhibits or AR exhibits (see Figure 7.17), which can be included in the game, but are mostly unrelated to the espionage story. Multiple choice questions are implemented in Flash and consist of three parts, introduction, question and evaluation. The multiple choice application is data driven and can be configured purely by entering text associated with an exhibit in the XML database. A marker next to an exhibit triggers the display of the question once observed by the user's PDA.


RadioFinder Question 05:00

You have to intercept a secret encrypted message.

Change your position to the "radio direction finder", until you hear a **continuous sound**. Point your PDA always in the direction of the marker, so that it is always **visible** on your screen.

Press **Start** button to play, or the **Map** button to cancel and return to the overview map.

TV Quest 02:19

A television set includes following main parts:

- ☑ Receiver
- ☑ Antenna
- ☐ Transmitter
- ☑ Tube
- ☐ Harddisk

Figure 7.17: Left: Radio finder task introduction; Middle: Multiple choice question; Right: Overview map showing current position of team members

The right picture in Figure 7.17 shows the overview map on the player's client device. Furthermore, users can see which hotspots have already been played by the team and which are still available.

7.3.4 Evaluation

The evaluation of MARQ consisted of an front-end evaluation, to find out about user interests and expectations, and a survey of the acceptance of the game, as well as a formative evaluation to test the system and gain feedback for the game design and development.

MARQ was tested by eight groups from different schools (encompassing 19 persons, aged 12-15) and has been evaluated by carrying out observation studies, supplemented by quantitative logging data and analyzing semi-structured interviews concerning handling and experience. The result of the survey has been qualitatively analyzed and the feedback was used for ongoing improvements in user interface and the conceptual formulation of the task descriptions. The changes were made after evaluation of group 4. See chapter 9.3.3 for the original questionnaires.

Interview Question	Assessment [1..5]
Experience with PDA	1,29
Motivation by use of PDA	4,29
Mark for the game	4,29
Extend game whole exhibition	4,2

Table 7.12: Average values of interviews rating from 1 to 5(=best)

Almost none of the subjects stated to have any experience with a PDA before. Only one person had seen and held a PDA once at least in his hands before (see Table 7.12). Cross-checking with a nominal question (yes/no) about experience with other mobile devices showed, that all of them were familiar with mobile telephones and some with portable game consoles.

The duration of the quest game was limited to 15 minutes, beginning from the moment the game has started. A single task was limited to 5 minutes absolute. If the task was interrupted or cancelled and chosen from another group, only the remaining time was available for solving it. The average task duration time (see Table 7.13) analyzed from logging data showed, that it was below the 5 minute time limit.

Quest Application	Average duration	Average Score
Radio Detection Finder	1:44 min	9 of 9
Morse Telegraph	3:01 min	7 of 9
Enigma	3:33 min	9 of 9
All quests	2:44 min	-

Table 7.13: Result of statistical analysis from the logging data

The game time limit was not exceeded except for the first group, which had severe difficulties in navigating the exhibition space without help. However, observation left the impression that the total game time was almost too short, because of the distances between task locations and the time needed to comprehend the task.

The interviews revealed that subjects demand having "more PDAs", "more action", "longer messages to decode" and "more adventure" in the game. Only in one case students were not convinced that the game should be extended to the whole exhibition. As their main reason they mentioned the complexity of the tasks, which turned out to be caused by insufficient instructions. As a consequence the descriptions and the user interface have been improved.

The first tasks were found to be the most demanding for inexperienced users, who had no previous experience with marker tracking and therefore needed a certain time for familiarisation. We consider introducing a tutorial task at the beginning, which just demonstrates how to use AR interaction with markers.

The quest which incorporates the Morse exhibit seems to be more difficult to solve correctly than the other two. This is emphasized by the high average duration time on this exhibit and the least correct answers. Interestingly the Morse quest was found to be most enjoyable. The two main reasons given by the subjects were: using the tangible interface in combination with the PDA and playing the sounds of real Morse signals.

Qualitative analysis of the survey data assumes that the overall acceptance of the game was fairly good, and it was perceived better by male students than female students, probably due to their acquaintance with portable game consoles. In contrast to the technical affinity of male students, female students left the impression that they had faster insight in understanding the task.

One of the main results of the observation shows that playing the quest must not be interrupted by interface complexity. Motivation is high, if the technology and the interaction works out fine, but frustration can quickly arise, if the user feels uncomfortable or experience malfunctions.

7.4 Expedition Schatzsuche

Based on the Sphinx engine described in detail in Chapter 6, we created a large situated game called "Expedition Schatzsuche" (Expedition Treasure Hunt). Although it is a direct successor to the quest developed for Technisches Museum Wien presented above, it is based on completely redeveloped software designs and game concepts, making best use of the lessons that we learned during creation of its predecessor.

The game was developed together with Imagination GmbH and Landesmuseum Kärnten and is hosted in the very same museum. For the first version that we evaluated, we developed 16 hotspots. All hotspots have been selected and described by the pedagogical staff of the museum, who also wrote all the texts and designed the tasks the players have to master. At each hotspot a player can experience what the exhibited item was used for. The museum staff used the chance to focus on items that look especially unspectacular in the showcase, items that are likely to be missed when walking through the museum without a guided tour.

Most hotspots are connected into sequences of events that require them to be played in a specific order. At the end of each sequence the player receives a piece of the treasure map. Early user tests revealed that especially children do not enjoy reading text on the small screen. Consequently we took care that all written texts in the game are just a few words long. Instead, there are multiple voice recordings for every hotspot, spoken by professional actors. While some of the sequences described below might appear unconnected, the well designed story and professional audio recordings convincingly link the hotspots together. In the following we describe two complete sequences, the photo tasks and the teams' base camps. The game starts at the base stations where the players can deposit map pieces or retrieve special tasks such as searching for an answer in the museum or photographing a specific exhibition item.

Due to the nature of the museum, the context of many exhibits is difficult to understand without participating in a guided tour. During the design phase the game, the museum staff decided explicitly to make use of AR for these exhibits:

- Placement of the "Stimmbogen" at the "Wiener Horn": The virtual crook is positioned at the horn (where it is missing in reality), to illustrate where to connect it to adjust the pitch of the horn.
- Illustration of the heating process of a flat iron: At the tailors' workshop the player's attention is attracted to the heating action - without electricity - of a

flat iron by putting it into an oven. This principle could be easily missed by visitors, because only the wooden grasp remains visible of the plugged flat iron.

- Playing the "silent piano": Playing a key with no audible sound illustrates the mode of operation of this training device. Electrified headphone systems have since then replaced the need for dummy pianos.
- Lifting bellows of an organ: The two flat bellows on the back side of the museum's organ are easily missed by visitors and it is difficult to imagine their inflated state during operation. Inflation is necessary to blow air into the organ which causes the typical sound of the instrument. The aim is to explain the working principle: One of the bellows has to always go down blowing air into the organ allowing the pianist to play.

Chain of hotspots: "Brauchtum"

This sequence about old traditions in Carinthia incorporates three hotspots. The story of this sequence is about a wedding and customs related to this topic. At the first hotspot (see left picture in Figure 7.18) the player has to look at a traditional belt and find initials of a wedding couple on it. If he succeeds he retrieves a flat iron of that time.

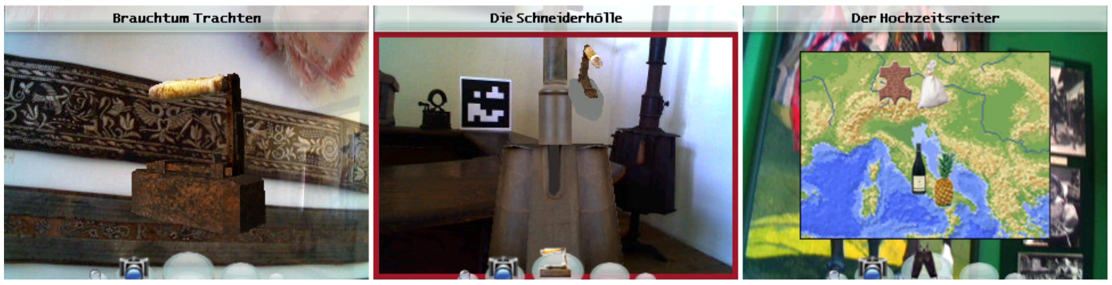


Figure 7.18: Hotspots of sequence "Brauchtum. Left: the player retrieves the flat iron for finding out the initials of the wedding couple; Middle: the player puts the iron into the tailors' oven; Right: the player learns about the wedding rider and finishes the sequence.

The flat iron serves as a hint to go to the old tailor's workroom where he learns how a tailor's everyday work life looked like (second image in Figure 7.18). Since the tailors are currently missing an iron the player can help out by providing the flat iron he just got. Gratefully the tailors reward the player with a wedding suit. In the next room the player finds a picture of a wedding rider (right picture in Figure 7.18). The handheld device continues with the story, telling that the wedding rider is missing his wedding suit without which he cannot invite all the families to the wedding. After the player delivers the wedding suit, the rider thankfully presents a piece of the treasure map.

Chain of hotspots: “Music”

This sequence contains four hotspots (see Figure 7.19) of which two - the silent piano and the organ - are implemented as AR applets. The music sequence starts with a silent piano that was used by novice pianists to practice without disturbing others. The handheld invites the player to learn a short piece of music. First only one note is played by highlighting it on the real keyboard, which the user has then to play too. Consecutively more and more keys have to be pressed until the user can play a complete short sequence. As a result the player receives a sheet of music. At the Mandora (a bass lute) showcase the player is asked to present some notes to hear a composition played on the instrument. In return the user receives a "Stimmbogen" (crook), which is used to modify the resonance characteristic of a horn.

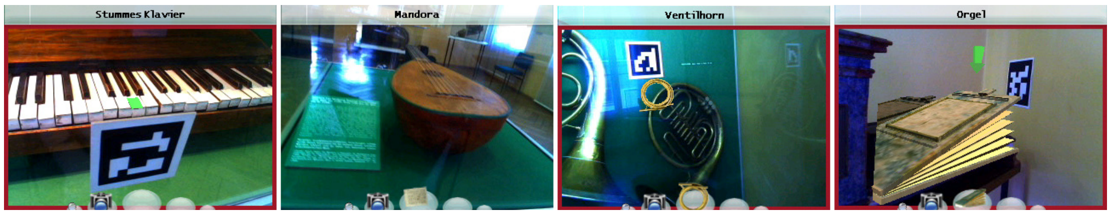


Figure 7.19: Hotspots of sequence "Music": Left: playing the silent piano; Second: listening to the Mandora; Third applying the "Stimmbogen" to the horn; Right: keeping the organ playing by pumping up the bellows.

At the "Wiener Horn" the player learns that the horn can play a large range of notes when applying different Stimmbögen. After applying the Stimmbogen the horn starts playing a song and the user receives a bellow. The bellow belongs to an organ in the very same room. To play the organ a second person always had to keep one of the two bellows moving or the music would stop. At this hotspot the player takes over the role of this second person in an arcade style mini game: He has to keep the virtual bellows blowing or the music will stop. To do this the player has to quickly select one bellow and press the “play button” of the Gizmondo continuously as fast as possible to pump it up. After providing air pressure for a short piece of music he receives a part of the treasure map and thereby finishes the sequence.

Answering questions using the Photo mode

Several hotspots require the players to answer questions (see second picture in Figure 7.20). Alternatively to presenting a list of possible choices, some of these hotspots put the mobile device into photo mode (see third picture in Figure 7.20). To solve the task the players have to take pictures of specific items they have to find in the museum. The players can take up to three pictures and must then return to the base camp for reward. After taking pictures of the correct item, an audio narration explains the story of the object. In case of questions concerning music instruments, an original piece of music is played as well.

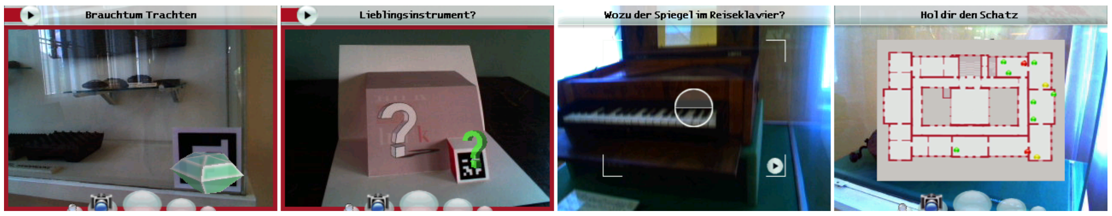


Figure 7.20: Left: Virtual diamonds telling the state of a hotspot (green: free, yellow: in use, red: already solved); Second: virtual questions marks use the same color codes as the diamonds; Third: Solving a task by taking a picture of the wanted item; Right: map of the museum showing all hotspots and their availability.

The Team's Base Camp

These hotspots serve as base camps and meeting points for the expedition teams in the museum. The players are told to bring all pieces of the treasure map to this station. On a big screen at a central location in the museum (in front of the stairway) the progress of the game is shown to the teams. A map displays the current state of the hotspots (see right picture in Figure 7.20): Available, currently in-use or already solved. After solving the quest and delivering the last piece of the map, the client device displays another map that reveals the location of the secret treasure in the museum.

7.4.1 Museum Integration

We selected the Gizmondo gaming console (see Figure 7.21) as the mobile device for our first prototype. The Gizmondo is a Windows CE 4.2 based gaming console with a 400MHz ARM CPU, 64MB of RAM, GoForce 4500 GPU, a screen size of 320x240 pixels, Bluetooth and GPS. Via its SD Card slot the Gizmondo's storage capabilities can be extended far beyond our requirements. The robust and well known form factor of a game console makes the Gizmondo an ideal device for our purposes.

Figure 7.21 shows a picture of the mobile client device running the Sphinx client engine Since the device itself has been designed for games the large number of physical buttons compensates for the lack of a touch screen. On top of the screen the name of the currently active hotspot is displayed. Even before a player activates a hotspot he can see the hotspots state by a colored diamond (see left picture in Figure 7.20) rotating on top of the marker: green means that the hotspot is free to play, while yellow denotes a hotspot already being played by another member of the team.

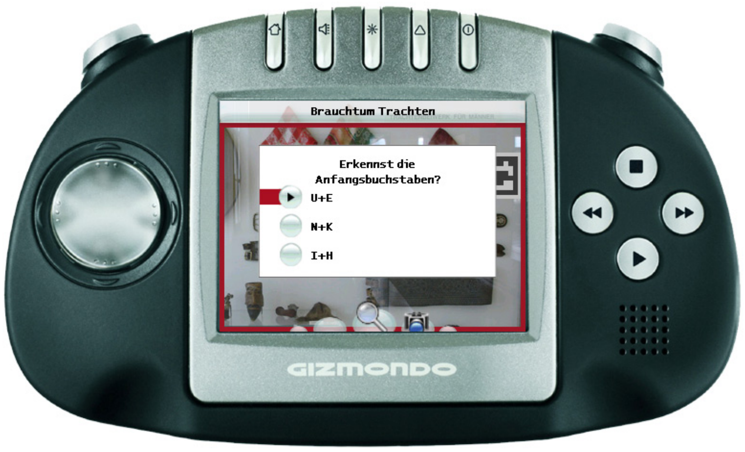


Figure 7.21: Client user interface

Finally a finished and closed hotspot is marked with a red diamond. Since there is network connectivity at hotspots this information is always up to date. The state and position of all hotspots can also be explored by every player and at any time by bringing up the 2D map (see right picture in Figure 7.20) using the home button, the left most button in the "button bar" at the top of the device.

At the bottom of the device's screen a ring menu lets the user browse all tools and items he is currently carrying. Each entry of the ring menu is symbolized with a white bubble that can hold one item. If the user wants to take a closer look at a scene he can activate the lens, or if he wants to switch the device into photo-mode, he can activate the camera tool. Early in-house user tests revealed that players wanted to have visual feedback when the device detected a marker, which was implemented using a red frame around the screen.

Figure 7.21 shows the mobile device asking the player to identify initials on a traditional belt. The question is posed using an audio message accompanied by a short on-screen text. All dialog boxes are defined and stored as templates on the client device, which allows re-layouting the GUI on-the-fly for different clients devices. To show a dialog, the state machine on the server asks the client to load a specific template and then fills the template's items (buttons, text fields and images) with content. Furthermore dialogs can be animated by updating the items in a timed sequence allowing "PowerPoint"-like presentations, which are typically used to introduce the player to the story of a hotspot.

A wireless network was installed in the museum to support the mobile devices. Our mobile devices only have Bluetooth, so we distributed Linux based WiFi base stations throughout the museum that can be extended with Bluetooth capabilities. These base stations act as bridges relaying incoming Bluetooth signals to the WiFi network providing a cost effective solution for an untethered, building-wide Bluetooth network (see Figure 7.22). One station can typically serve a Bluetooth cloud for two or three rooms, depending on the layout of the rooms and the mounting of the Bluetooth emitter.

We created a graph structure that models the neighborhood of Bluetooth access points and allows clients to quickly roam between Bluetooth connections. Since the MAC addresses of all Bluetooth stations are known at start-up the lengthy Bluetooth device discovery procedure can be skipped and the mobile clients can directly connect to new hotspots, which usually takes only a fraction of a second.

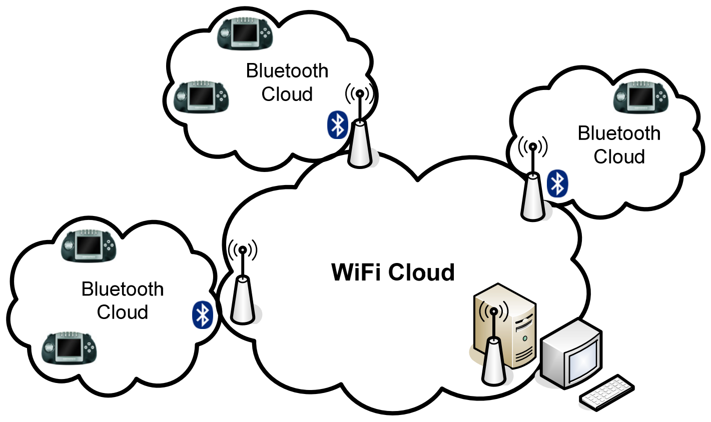


Figure 7.22: Museum network structure

Yet, after first test runs the museum management decided to save costs by not outfitting the museum with wireless network, which required extending the Sphinx engine to run the already designed state machine graphs on the client devices too (see chapter 6.2.2).

The network topology was therefore redesigned to only include a single Bluetooth cloud at a central meeting point in the staircase. A large high-resolution TV screen permanently shows the overview map of the game as well as the list of all players. Icon show how many map of pieces each player already found.

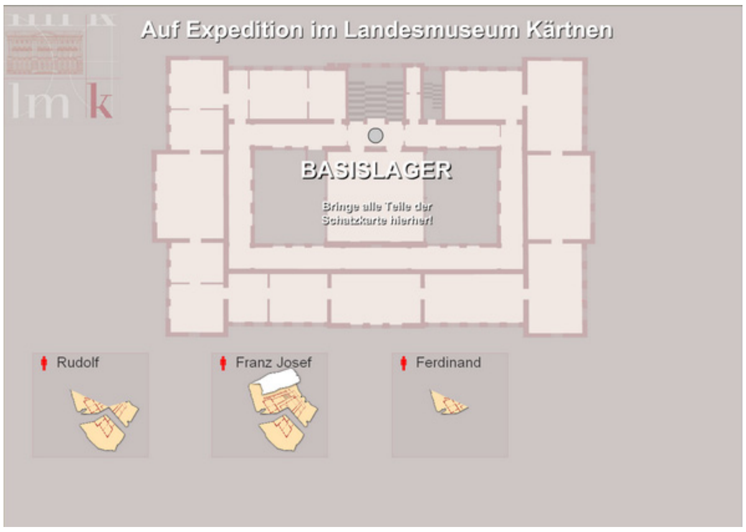

Auf Expedition im Landesmuseum Kärtnen
lm k
BASISLAGER
Bringe alle Teile der Schatzkarte hierher!
Rudolf
Franz Josef
Ferdinand

Figure 7.23: Overview map and list of player on the big screen at the central meeting point.

7.4.2 Evaluation

The field evaluation was conducted as a participant observation and half-structured interviews afterwards, with support of the education department of the museum. Two runs of six simultaneous players (12 high school pupils, six male and six female, at the age of 12 years) have been carried out. Each of the test persons has been given their own Gizmondo to play the game.

While the interaction with the game interface and the comprehension of the game tasks posed no problems for the test subjects, the aspects of tracking and augmentation were new to them. The mechanism of activating a hotspot by aiming at it and pressing a button while the hotspot is being detected by the tracking system was intuitively accepted by the users.

Observation and analysis of video recordings showed that the users understood and used the visual feedback of the red frame that appears on the screen when the tracking subsystem detects a marker.

In the interviews the users rated playing the AR mini-games as more difficult than the conventional tasks, which is backed up by our observations. Players stated that they had no problem with playing the "silent piano" after finding out 'how to' and 'what to do'. The time critical task of operating the organ's bellows on the other hand demanded their full attention. Consequently some players stopped aiming the camera at a marker while playing (which had no negative effect except for incorrect augmentation). To improve this, the users should be given more time for getting acquainted to AR interaction.

Following, we summarize our key observations of the test run.

What went right:

- AR animations were experienced as exciting and interesting: "the station with the flat iron was cool"
- The mechanism of tracking was easy to understand: "that was quick to comprehend" and "has been all illustrated"
- Collaboration and cooperative use of the client device: Three of the users were sharing two devices, after one device crashed due to hardware problems.
- The on-device overview map was used extensively for localization of the hotspots: "look we are here - we have to go there".
- The players spontaneously formed groups playing together as well as split up and played individually.
- The users rated playing is highly motivating: "Not only answering questions, but also playing … like the piano and the organ." and "to learn and to play [at the same time]"
- Playing the game was perceived as fun and learning at the same time: "…although one is learning" and "I haven't known that the music instruments are so old." (referring to the oldest conserved Mandora in the world).
- Solving questions by taking pictures of the answers has been perceived as intuitively and easy to handle. When the players found the right answers, audio explanations are given.
- Although the first hotspot of a sequence was not specially marked and therefore took some time to figure out, the test persons stated the concept of solving one station to make the next station playable as very motivating: "the combination 'thing' was very good" and "… that you have to move around and search … like a detective"

What went wrong:

- Augmentations were intuitively comprehended, but players had sometimes difficulties with aiming the camera at markers, especially when distracted by interactions.

- When players formed groups they often operated all client devices simultaneously, which resulted in playing multiple audios at the same time. Especially voices were hard to understand in such a case.
- One device failed due to problems with a slow memory card containing the client-side game content. One of the groups had to restart their game several times until we removed the device from the game.
- The extremely strong walls of the ancient museum building requires a lot more WiFi access points than anticipated and resulted in changes of the overall game play (see chapter 7.4.1).
- Showcases that are situated opposite to windows can suffer from strong reflections. While the human brain can easily compensate this effect, the tracking quality can be reduced considerably.

7.4.3 Lessons learned

Using the ARToolKitPlus markers for tracking is a non-trivial task in a museum. Placement of markers is restricted, since they cannot be directly attached to historical artifacts, yet need to be of reasonable size and in view of the camera. Understandably, the museum staff would like to place markers in a way so that they are not noted by the visitor, which is in contrast though to the technical requirements of marker tracking.

More tracking problems are caused by the dim light. Many exhibits are very old and may only be presented under severely restricted lighting conditions. Some interesting areas in the exhibition could not be integrated due to improper illumination or the inability of proper market placement.

The fact that typically only teachers will be available for supervision puts extraordinary requirements on the robustness and ease of use of the client devices. We aimed at making the user interface as intuitive as possible, and provide detailed task descriptions at any time. Still it turned out that some tasks need to be explained in more detail to be fully understood - especially those that go beyond the usual interaction mechanisms, notably the AR applets.

The task of keeping track of markers with the video camera works well while consuming AR enriched animations, but turned out being too difficult for some users while playing fast-paced AR games. It is therefore essential to integrate better tracking technologies. A preferred solution would natural feature tracking, which sounds reasonable, considering that in a museum application typically only small and well know areas with constant lighting conditions around exhibition items need to be dealt with.

7.5 Signpost

There is a long history of the Signpost applications in the group around Prof. Dieter Schmalstieg. The idea of the Signpost application is to guide a user through an unfamiliar building: The user selects a target location and the system shows the way until the user reaches this destination.

The first incarnation was created as an undergraduate project by Michael Kalkusch et. al [56] in 2002 (see left and middle image in Figure 7.24). The project mainly focused on wide area tracking using the ARToolKit library [57] and required distributing dozens of markers on the walls. Using the application, a member of the faculty would be guided to the institute's library and directed to a previously selected book. Naturally, a back-pack setup is not a practical means for selecting a book from a library. Yet, the project set a milestone in wide area, low-cost indoor tracking.

The concept was later improved by Gerhard Reitmayr [87] who extended the system to work outdoors. In this application tourists equipped with a back-pack AR system are guided through a city. When they approach an object of interest, the system augments spatially registered graphical information onto the view of the user. Targeting outdoor usage, Reitmayr extended the tracking system with GPS and inertial tracking.

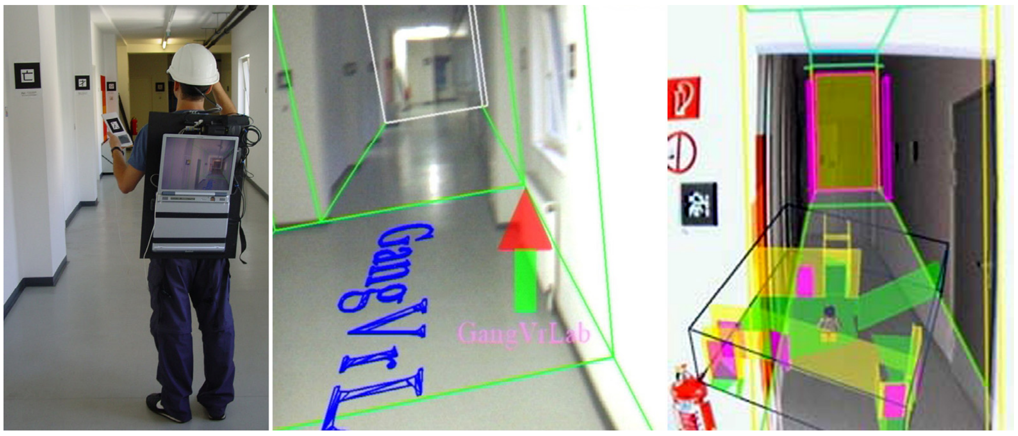


Figure 7.24: Signpost on the PC. Left and middle: Signpost 2002 for indoor tracking; Right: The World In Miniature (WIM) interface.

In 2003 Michael Knapp improved the indoor application with his work on Signpost II, which introduced the BAUML (Building AUgmentation Markup Language) XML dialect for creating, storing and retrieving building data. The WIM (World in Miniature Model, see right image in Figure 7.24), allows a user to get a better overview of his current location. A marker-allocator application systematically targeted the problem of repeating the limited number of markers in a large area.

From 2005 on Schall worked on self-surveying [91] in order to reduce the deployment times for Signpost-like systems in large buildings. The software developed in this project imports measurement data from a Leica Total Station, a tool commonly used for high precision indoor and outdoor 3D measurements. Based on the multiple sets of measured data, a graph of inter-relationships is created, optimized for minimal error and finally converted into a BAUML representation.

The author of this thesis contributed to this series of applications with the development of Signpost 2003 and Signpost 2007. Although the 2007 version is the successor to the Signpost 2003 application, the original application had several features that were not reimplemented in the new version. Both applications mark the very first and the last developments performed during this thesis. It is therefore of interest to first introduce the original application before presenting its successor.

7.5.1 Signpost 2003

Development of Signpost 2003 started in end of 2002 with the beginning of the handheld AR project. Due to the early stage of the handheld AR project and our minimal experience with PDAs at that time no high quality content was available. Signpost 2003 used a custom renderer, which later evolved into the Klimt library (see chapter 4.1.3) and displayed most 3D graphics in wireframe (see Figure 7.25).

The complete application, including user interface was optimized to use permanent tracking due to the vast amount of markers deployed at the institute at the Vienna University of Technology. While such a setup is unrealistic for practical applications and led to significant design changes for the 2007 version, it was easily justified for an early research proof-of-concept demonstration.

Signpost 2003 acts as a guide that routes a visitor through an unfamiliar building. At startup the user selects a target room. The application uses a cell and portal system to guide the user room by room to the target location. A 3D arrow shows the user the direction to go: It points to the next door to take (see left image in Figure 7.25) and thereby guides the user from room to room to the final destination. Since the complete building was available as mesh, the 3D model served as overview map as well (see right image in Figure 7.25).

Unlike its PC-based variant, the PDA-based Signpost 2003 could not rely on Studierstube, Coin3D and OpenGL. The PC version uses preprocessing to convert the XML document that encodes the measured building via an XSLT processor into an OpenInventor scene-graph. The PDA version instead imports all data directly from the XML file, building a 3D mesh optimized for the custom software renderer on the fly.

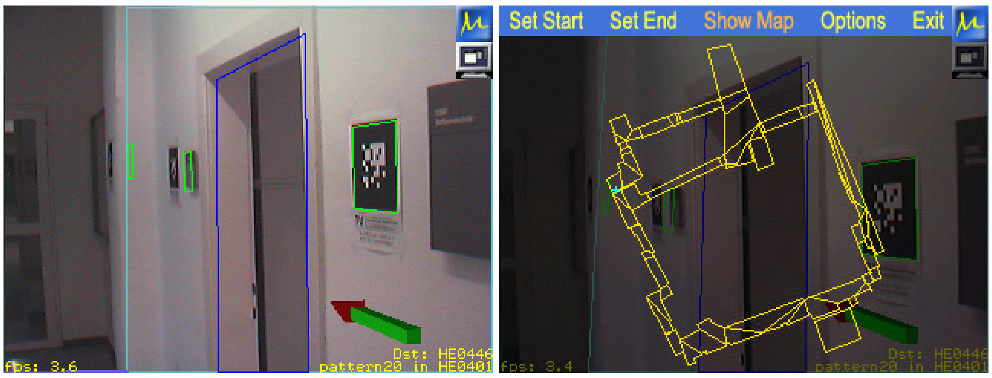


Figure 7.25: Signpost 2003.
Left: Wireframe overlay of building data; Right: Overview map

The application creates a search graph from the portals and cells inside the XML. At every frame the system selects the cell the user is currently in and calculates the shortest path to the target. Knowing the portals (doors) that connect the cells, the arrow can guide the user to the door that leads to the next room.

7.5.2 Signpost 2007

Although both the 2003 and the 2007 version of Signpost serve the same general purpose, the design of the 2007 version is completely different. The 2007 version was created from ground up to work in coarsely instrumented and sparsely tracked environment. For the 2003 version an accurate 3D model of the building plus markers at every 2 meters on the walls were available. Signpost 2007 instead targets deployment at conferences, where such infrastructure requirements are unfeasible.

Signpost 2007 uses bitmap maps of buildings, because accurate 3D models of conference locations usually do not exist or are not available. Another important aspect to transform the original version into a practical application was reducing the amount of markers required to run the application. The original version made use of about 200 markers deployed in an area of 20 by 30 meters, while conference organizers would typically install about 40-50 markers for an area about two orders of magnitude larger (see Figure 7.26). We therefore redesigned the application's workflow to only sporadically require markers for operation. While 3D building data can not be meaningfully overlaid anymore in such a case, it is sufficient for displaying detailed 2D maps.

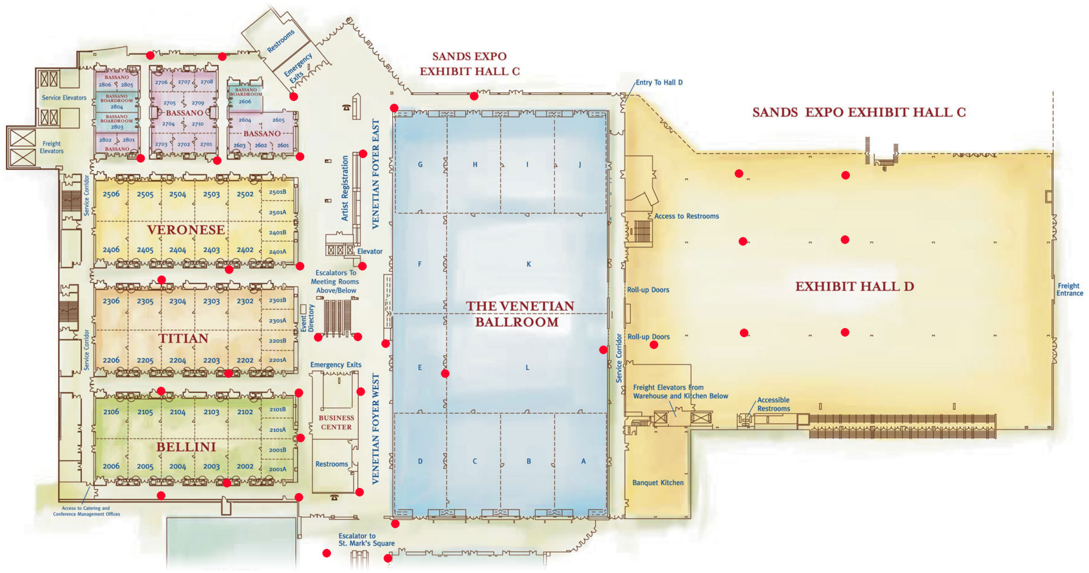


Figure 7.26: Marker placement for Signpost 2007 at the MEDC 2007 conference in an area of roughly 100x200 meters. Red dots mark locations of posters with instructions and markers.

Using 2D map tracking only further reduces requirements on accurate marker placement accuracy. While the markers for the Signpost 2003 application were measured at millimeter accuracy for precise overlays, the 2007 version requires only coarse marker placement: Markers can be stuck onto posters stands that are deployed quickly on-site. While deploying markers and creating 3D models at the Graz and Vienna universities took months, setting up the markers for Signpost 2007 at a conference site usually takes just one or two hours. Signpost can update marker pose data via an on-site wireless network to allow changing positions and orientations during the event, which turned out to be a very welcomed feature.

Hence, Signpost 2007 is centered about operating map positions rather than 3D overlays. From a research point of view this meant a step back from the previous version, creating a less AR intensive application. From a usability point of view these changes transformed a mostly unusable research prototype into a practical and highly successful tool for conference attendees that is easy to deploy and manage.

The most distinguishing new feature in the 2007 version is special support for data browsing. Targeted as a conference guide, Signpost 2007 includes a complete schedule that can be browsed using various filters such as per day, per session or full text indexing. Like the marker positions, the schedule can be updated via WiFi to reflect latest changes of the event's schedule. Users can freely navigate maps (see Figure 7.27) by panning, rotating and zooming in a UI style similar to regular navigation systems such as TomTom[36] or Navigon[37].

To operate the map manually (without tracking) the user has to first select between the different naviation modes (panning, rotating and zooming) and can then use the cursor cross or stylus to modify the view onto the map. Furthermore the user can switch between maps, which is necessary for multi-level buildings or events with multiple non-connected sites.

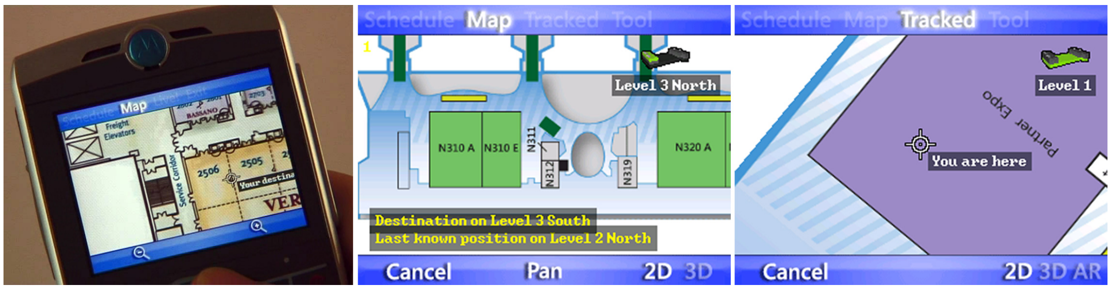

Schedule Map Tracked Tool
Level 3 North
N310 A
N310 E
N311
N312
N319
N320 A
Destination on Level 3 South
Last known position on Level 2 North
Cancel
Pan
2D 3D
Schedule Map Tracked Tool
Level 1
Partner Expo
You are here
Cancel
2D 3D AR

Figure 7.27: Navigating a conference map. Left and middle: Using the cursor cross (joystick) to pan a map Right: Currently tracked position

2D User Interface

The numerous features and the focus on 2D interaction required designing a suitable 2D user interface. While this poses no problem on devices with touch screens, the project also targeted smartphones with button only interfaces.

Most users today are highly familiar in operating devices with touch screens style user interfaces. Many ATM or point of sales machines today use touch screens, and the general UI method is very similar to that of using a desktop mouse. Creating user interfaces for devices that are operated with buttons only is quite hard by comparison. Instead of just drawing a button on the screen and letting the user touch it, the UI has to make clear which virtual button on the screen relates to which physical buttons on the device. Devices that are designed for a single purpose can take advantage of this

[36] http://www.tomtom.com
[37] http://www.navigon.com

fact by introducing special purpose buttons. For example, players know which buttons to press on mobile game consoles since the button layout follows well known conventions. Similarly, users know very well how to operate a phone for the purpose of calling other people. Applications, which are unusual for a specific device class though, cannot take advantage of this method.

The Signpost 2007 application targets an open range of Windows Mobile smartphones with and without touch screens. Although all these devices are optimized for using them as mobile phones or personal information managers (PIMs) their physical design varies significantly. Most mobile phones today include many special purpose buttons such as camera buttons, scroll wheels and applications launchers. Yet, most of these special purpose buttons are not standardized. Due to the very different button layouts and mapping of various mobile phones only few buttons can actually be used on an application that targets unknown and untested devices.

Signpost 2007 therefore only uses the cursor cross, the two screen buttons, the zero to nine button and character keys in case text input is required (such as for text search boxes). Although Signpost is a fully graphical, skinned application it tries to duplicate the standard Windows Mobile GUI elements to give the user a familiar feeling. Hence, it shows the mappings of the two screen buttons on a bar on the bottom of the screen (see Figure 7.29). These buttons are always mapped in a way that the left button cancels or exits an operation, while the right button confirms a selection.

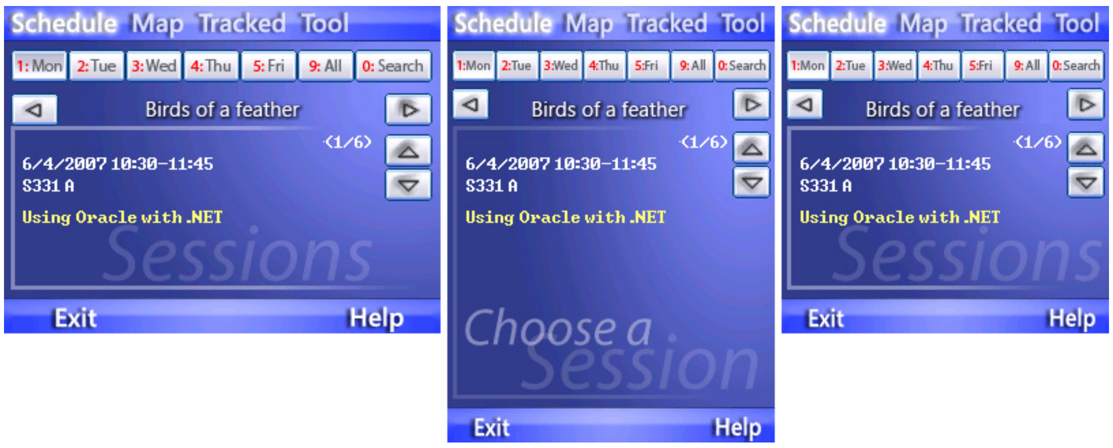


Figure 7.28: Three different resolutions supported in Signpost 2007. Left: 320x240, Middle: 240x320; Right: 240x240

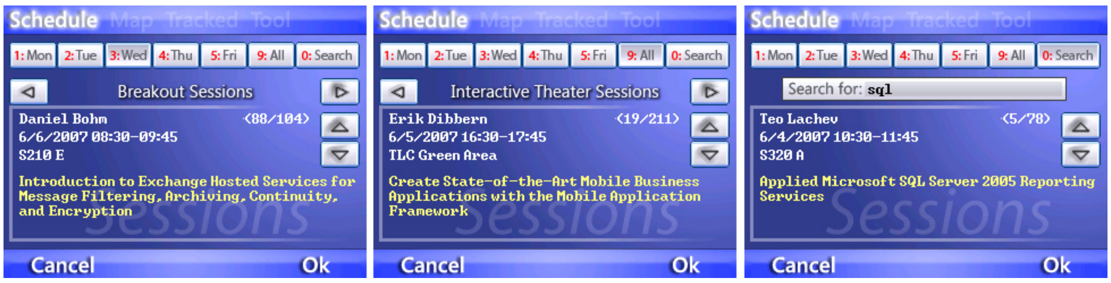
All buttons on the screen can be pressed with the stylus on touch screen enabled devices. On devices without touch screen support the on-screen buttons are mapped to physical buttons and keys. Text or symbols on the buttons serve as hint, which physical button to press: Filtering selection is done using the keys from 0-9, which assures that even devices that have no touch screen and only a T9 keypad can operate these functions. The user can then switch between sessions using the left and right directions of the cursor cross. Similarly iterating through talks is done by pressing up and down. On the bottom bar, two screen areas are mapped to the physical screen-buttons which are always placed near the bottom left and right corners of the screen.

When deploying mobile phone software to a large number of devices support for different hardware specifications becomes a major issue. While 3D applications can usually easily adapt to different screen resolutions, 2D graphical user interfaces require more work. Figure 7.28 shows the three different resolutions that are currently supported in Signpost 2007. For each resolution we specifically created a complete application layout that is specified via an XML file. At start up the application automatically detects which layout to use.

Manually providing layouts for specific resolutions achieves optimal results for those resolutions that are supported, but creates scalability problems to the amount of different devices that can be supported. Currently about ten different screen sizes ranging from 176x220 to 800x480 are in use, where the highest resolution can display ten times as much data as the smallest one. More screen sizes are sure to come. The most promising option in such a case is probably automatic layouting, which we also plan for the next version of Signpost 2007.

Application Features

Figure 7.29 shows the "Schedule" screen of Signpost 2007 that allows a user to select from a large number of items. Obviously the amount of data is too large to iterate to the list item by item. Filtering allows creating a more effective selection, but it complicates the user interface.



Breakout Sessions
Daniel Bohm <88/104>
6/6/2007 08:30-09:45
S210 E
Introduction to Exchange Hosted Services for Message Filtering, Archiving, Continuity, and Encryption

Interactive Theater Sessions
Erik Dibbern <19/211>
6/5/2007 16:30-17:45
TLC Green Area
Create State-of-the-Art Mobile Business Applications with the Mobile Application Framework

Search for: sql
Teo Lachev <5/78>
6/4/2007 10:30-11:45
S320 A
Applied Microsoft SQL Server 2005 Reporting Services

Figure 7.29: Schedule screen of the Signpost 2007 application. Left: Filtering by day; Middle: No filtering; Right: Filtering using full text indexing.

For large conferences with lots of presentations even the aforementioned two-step filtering via days and sessions can be too difficult since attendees often don't know which session a talk is held in. In Signpost, pressing the '0'-button activates the full text filter that shows only those entries that contain the specified string (see right picture in Figure 7.29).

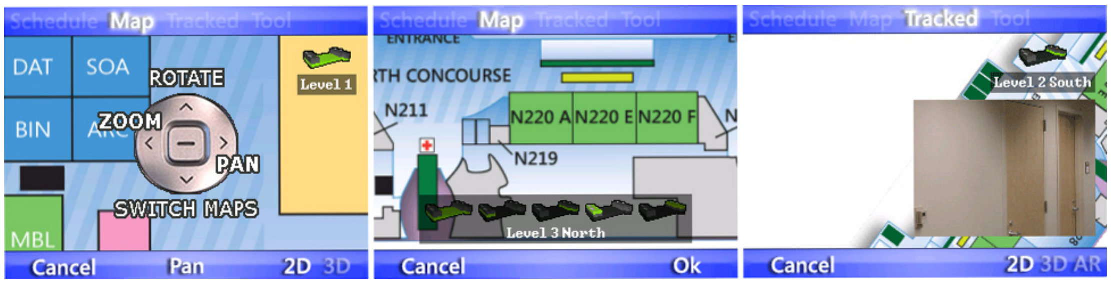


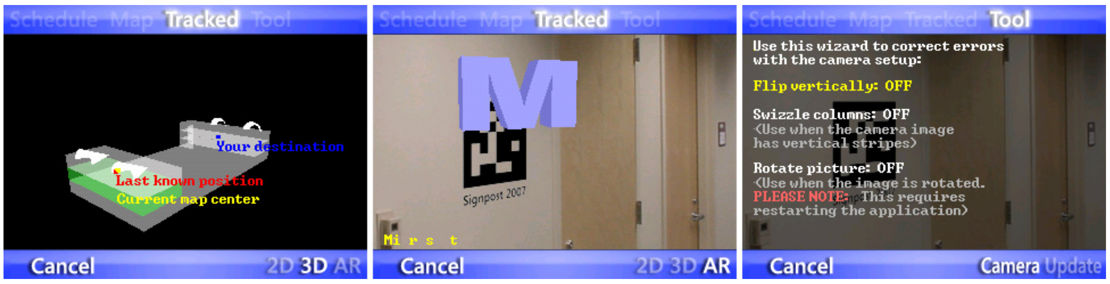
Figure 7.30: Navigating maps in Signpost 2007.
Left: selecting an action with the in-place menu; Middle: Switching maps;
Right: View finder while no marker is visible in tracking mode.

Navigating a 2D map is a common UI concept for mobile navigation systems. These devices usually rely on touch screens or special purpose buttons. Signpost 2007 demands more operations that the typical zoom, pan and rotate map as well as switch between maps (see left image in Figure 7.30).

Tracking of fiducial markers only works, when markers are visible in the view of the camera. Aiming at markers with the mobile device's camera is easy as long the camera's image is visible on the screen, such as the video background in video see-through AR setups. While Signpost 2007 does include a "regular" AR mode, users mostly operate it in the "tracked map mode" that shows a correctly panned and rotated map, but no video background (see right image in Figure 7.27). We therefore automatically blend in the camera image (see right image in Figure 7.30) as a view finder, when the camera looses sight of markers to support the user in finding a marker again. To keep the user interface consistent the view finder image is smoothly faded in and out instead of just popping up.

While the first version of Signpost 2007 allowed only a single map, with the introduction of support for multi-level buildings the problem arose of how to make people aware of their current location within the building. Showing their 2D location on the map was not enough anymore. We therefore added 3D building models that are simple enough to be quickly created, yet sufficient to show the user's current location in a building.

Other than in the 2003 version, these 3D models are not meant to be accurately overlaid on the real environment, yet in tracked mode the building model is correctly rotated help the user find her target location. The left image in Figure 7.31 shows a screenshot of the tracked 3D building. On this screen the user is informed about her current location and the target location, which other than in 2D map mode can span multiple building levels. The currently active map is highlighted.



Use this wizard to correct errors with the camera setup:

Flip vertically: OFF

Swizzle columns: OFF
(Use when the camera image has vertical stripes)

Rotate picture: OFF
(Use when the image is rotated. PLEASE NOTE: This requires restarting the application)

Figure 7.31: Left: Tracked 3D building; Middle: AR objects (Easter eggs) in Signpost 2007; Right: Configuration screen to solve camera driver bugs

To go beyond a pure data browsing and mapping application we also added a small AR treasure hunt game that can be used for marketing. Each marker holds an arbitrary virtual object ("Easter egg", see middle image in Figure 7.31). To find these items, the user has to switch into the game mode of the application which brings up the video background in full screen. Whenever the user points the device to a marker it shows the attached Easter egg. After the user collected all different objects, he can register to win a price.

The Signpost 2007 application targets any Windows Mobile 2005 or later devices. Unfortunately, typical problems with these devices include camera drivers bugs. Devices tend to report wrong video modes or return camera images in wrong orientations. To overcome this issue, we added a camera setup tool to Signpost 2007 that allows the user to override the camera's reported settings in order to adapt the application to driver bugs (see right picture in Figure 7.31).

7.5.3 Evaluation Results

We performed an evaluation on Signpost 2007 at the MEDC 2007 conference. Although about 150 attendees tried the application, only 34 participated in the evaluation. The reason for this is that we required the users to find all Easter eggs, which – in the first version of the application – turned out to be a too demanding task in terms of time required finishing the game. For later versions we changed the game mode to not require visiting all markers.

Participants were mostly native English speakers. The vast majority was male and no subject had seen the application before. All participants of the study answered the following nine questions (see Chapter 9.3.4 for the original questionnaire) by marking on a Likert scale from 1 (I strongly disagree) to 7 (I strongly agree):

- Q1: Signpost was easy to use
- Q2: Signpost was more useful than a conventional map
- Q3: Those black-and-white markers disturbed me
- Q4: I'd like to see the other users' positions on my device too
- Q5: I think Signpost2007 can be used by novice PDA or Smartphone users
- Q6: I was able to quickly access and understand the information (schedule and map) I searched for
- Q7: I enjoyed using Signpost2007
- Q8: Signpost2007 improved my location awareness
- Q9: Signpost2007 should be used on other events too

As can be seen in Figure 7.32, all answers were very consistent with only minimal standard deviation.

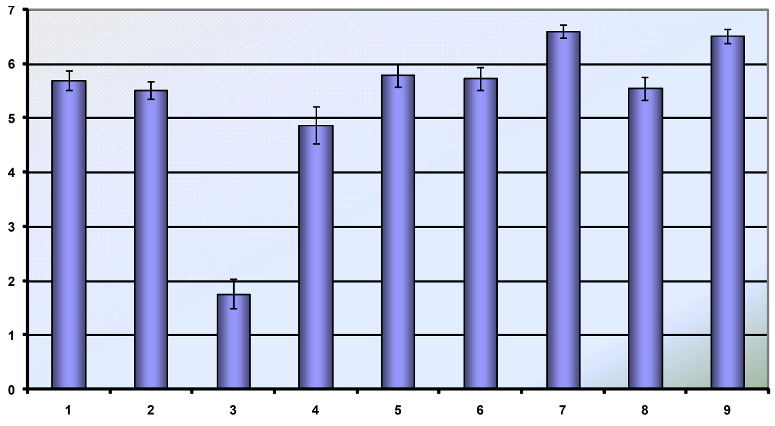


Figure 7.32: Results of the questions Q1 to Q9 of Signpost 2007 evaluation at MEDC 2007

Attendees found Signpost consistently easy to use (Q1) and more useful than the conventional conference map (Q2) that was part of the printed conference guide. Since MEDC is a conference targeting expert users and developers we specifically asked for their opinion about suitability to novice users. Even though we received reports that Signpost does not follow the standard UI conventions enough, attendees rated it suitable for novice users (Q5). Most users were able to quickly access the information presented on their device (Q6) and experienced improved location awareness (Q8).

Users consistently enjoyed the application (Q7) giving it an average score of 6.6 out of 7. Furthermore the attendees strongly believed that Signpost should be used at other conferences too (Q9) giving a score of 6.5 out of 7 for this question.

In Question Q4, which asked about seeing other users on their screen created we noticed the highest diversity among the answers. It was also only question that received answers from both extremes “I strongly agree” to “I strongly disagree”. Talking to users confirmed our expected concerns about privacy issues that many users have with such a feature.

Using fiducial markers in natural environments outside research labs always raises concern for acceptance of the introduced visual clutter. Yet, most attendees did not complain (Q3) about the deployed ARToolKitPlus markers giving it a score of only 1.7 at a range from 1 to 7, thereby disagreeing to the statements that they felt disturbed by the markers. A probable reason this is that conference areas are generally visually polluted with posters and projection screens, making the fiducial markers less eye-catching than in other environments.

From the results above we can conclude that Signpost 2007 is highly accepted – at least by technology savvy audiences. Most of the problems we experienced at the MEDC conference could be solved for the later TechEd conference, though some new issues were introduced with those changes. At MEDC it turned out that marker locations as well as the event’s schedule could not be fixed before conference start. We therefore created new distributions on a daily basis, which allowed new users to work with the latest settings, but users that previously downloaded Signpost did usually not update. Fortunately, only minor changes aroused. This issue was solved for the TechEd event making Signpost able to download updates instead of requiring updating the whole application.

For the TechEd conference we added many new features such as multiple map support, full text search, downloading updates and configuring the built-in camera, which created a rather complex user interface cluttered with features and introduced problems to new users. We therefore plan to redesign the complete user interface and make it more standard conform as well. Due to the enormous success of the Signpost application at the MEDC and TechEd conferences, Microsoft asked for deployment on a number of more conferences.

The example of the Signpost 2007 application shows that AR technology can be used effectively for ubiquitous non-AR applications. The system can be easily adapted to new environments, requiring only a map of the target site for preparation.

Tracking based on computer vision further reduces the costs since it only requires placing a few posters with marker at the site. It is therefore preferable over alternative tracking methods such as infrared or Bluetooth beacons, which require active devices and give less precise localization. Furthermore allowing users to experience the application on their own mobile phones improves acceptance from the attendees and at the same time reduces costs for client devices to zero.

7.6 Discussion

The applications presented in this chapter demonstrate the practical usability and deployability of phone-based Augmented Reality applications. While first prototypes such as Signpost2003 and the Invisible Train were created mostly for demonstration purposes and have been shown only inside our labs or at research conferences, later developments such as Signpost2007 and MARQ are practical applications that have been deployed into "real" target environments and used by many inexperienced users including children.

The evaluations described in this chapter show that Augmented Reality on phones is well accepted and allows creating user interfaces that are easy to understand and navigate. A more focused discussion on the importance of these results on the hypothesis (especially hypothesis H3) as stated in chapter 1.3 is postponed to the next chapter.

Chapter 8

Conclusions and Guidelines

In this chapter we analyze how the hypothesis stated in the introduction chapter was fulfilled. We discuss how each requirement was met; give guidelines on creating handheld AR applications and finish with an outlook to future work.

In the previous chapters solutions developed in the handheld AR project were presented that resulted in a fully working and practical framework for Augmented Reality applications on mobile phones. While it is unique in being the only complete AR solution for mobile phones today, it also goes beyond many other AR frameworks currently available in terms of supported features. Applications based on the presented framework have been used by thousands of users. While first creations such as the Invisible Train required providing specific hardware units for demonstration purposes, the latest applications were used by hundreds of users on their own devices. Following, we discuss how each statement of the hypothesis in chapter 1.3 was met:

Hypothesis H1 *Augmented Reality on phones can work as well as on personal computers, despite the fact that phones are less powerful, have smaller screens and inferior input capabilities*:

The solutions presented in the chapters 3-6 resulted in a framework that allows the rapid creation of practical AR application on mobile phones. Processing capabilities of mobile phones are an order of magnitude or two lower than on PCs. Yet, benchmarks presented in this thesis demonstrate that performance and features similar to that of average PC-based AR setups can be achieved. Typical frame rates are in the range of 10-20Hz, which is below the cinematic standard of 24 Hz, but empirically sufficient for interactive applications, especially on small screens.

The small screen size and reduced input capabilities of mobile phones raise concerns for their suitability for applications that go beyond typical PIM (personal information management) tools such as calendar and contact management. Yet, despite these restrictions, the feedback on the applications and evaluations presented in chapter 7 shows that practical AR applications and games can be created.

Most classic game concepts can not be effectively transported to the mobile phone platform due to its tiny screen size and bad input capabilities. Hence, despite the enormous amount of deployed client units, a large and profitable game market has not developed yet for this platform. In contrast, AR offers a “point and shoot” style interaction, thus using the phone and its position in space as an input capability. It thereby overcomes the limitations of T9 and similar input concepts. By using the Magic Lens metaphor, the phone’s screen is virtually extended and can easily be navigated.

Hypothesis H2 *Using phone based AR, larger mobile Augmented Reality systems than previously shown can be built at reasonable costs*:

The extremely low cost per unit of AR on mobile phones (especially if pre-owned by the user) is a unique selling point. While any application or game presented in this thesis could be created using traditional hardware such as Tablet PCs or backpack setups, their prohibitive costs, increased weights, etc. prevented the development of any mobile AR system with more than three users so far.

Furthermore, when selecting the user’s private mobile phone as target device, the per-unit costs go down to zero as demonstrated with the Signpost 2007 application: The costs for the event organizer were restricted to providing a few files for download on a web server. This allowed hundreds of concurrent users to experience the AR conference guide, which is about two orders of magnitude beyond any other AR application deployed so far.

Hypothesis H3 *The phone’s form factor is more suitable for untrained users than HMD-based setups*:

Backpack setups with HMDs have the advantage of providing high processing power and immersion. While HMDs have clear advantages in application areas that require stereoscopic augmentations or hands-free interaction, they did not succeed in mass markets. The high costs HMDs themselves prevent a commercial success in a mass market, but is only one factor that reduces their suitability for untrained, private users. The aim of HMD producers to reduce the HMDs’ weight, while still keeping the costs at a reasonable level resulted in highly fragile devices that easily break, even when handled carefully by expert users. Before usage, an HMD needs to be put on and calibrated at least coarsely by the user, which are both complex tasks and therefore difficult for inexperienced users.

In contrast to HMDs, most people today know very well how to operate a mobile phone. The usage of phones as Magic Lenses is highly intuitive, requires no calibration and is easily understood even by small children as our experiences show. We therefore conclude that despite the higher potential of HMDs in certain (expert-) domains, phones are more suitable AR devices for untrained users. Consequently HMD- and phone-based systems are rather complementary than competitive.

We now summarize how the system developed in this thesis fulfills the requirements on a practical AR setup as stated in chapter 1.5:

- **Low cost:** The Signpost 2007 application was used by hundreds of conferences attendees on their own mobile devices. Users were able to install the applications on their Windows Mobile phones reducing the costs for conference organizers to hosting a few files for download on already deployed the intranet web server.
- **Robust and fool-proof:** In the last year of this thesis we developed several demo applications that were sent out by email to users. Untrained users were able to install, understand and use the applications on their own in natural unsupervised conditions.
- **Self contained operation and networking support:** The way how an application uses the networking and communication capabilities strongly depends on the actual application. Based on Studierstube ES we developed applications that run fully self-contained and require no networking at all (such as Signpost2007 or various small demos), applications that sporadically use networking (such as the MARQ treasure hunt game) as well as applications that require networking at all time (e.g. the Invisible Train or Virtuoso games).
- **Tracking support:** As outlined in Chapter 3, tracking with fiducial markers performs well in various lighting conditions even on low-end smartphones.
- **Rapid prototyping:** All applications presented in this thesis were developed with only one or two developers and graphics artists, which is a minimal staff compared to typical commercial productions. Details on how applications can be developed with Studierstube ES are given in Chapter 9.
- **Content creation:** The Schatzsuche and Virtuoso games are both heavy on content and benefited a lot from a clear content creation pipeline that supports direct import of well known and supported multimedia file formats.

We therefore conclude that handheld Augmented Reality is suitable for mass market audiences. The increasing interest we receive from commercial entities such as from industry or marketing companies in AR on mobile phones supports this claim.

Developing for mobile devices such as cell phones is fundamentally different from creating applications for desktop computers. For good reasons creators of operating systems for these platforms publish style guides on how to design, develop and deploy applications for these devices. Yet, these documents usually cover only aspects common for typical mobile phone applications such as organizers or date books.

The following chapter summarizes the experiences gathered in the last five years in the handheld AR project. These guidelines are grouped into two areas: the chapter on applicability lists typical pitfalls that researchers are confronted with when creating commercial handheld Augmented Reality applications. The guidelines on performance offer advice how to make optimal use of those target devices that are always short on resources.

8.1 Guidelines on Applicability

There is a strong difference between research prototypes and practical applications with commercial grade quality. Often it is not possible to transform a prototype into a practically deployable application. Even if it is, it usually takes longer than creating the prototype itself. While it is enough for a research prototype to just run "somehow" on the target device, applications for end users must run always and be intuitive to use and behave "nicely", which are requirements that researchers are not are used to. In the following we list our experiences, partially learned the way in creating commercial grade AR applications for end users:

- **Deployment:** Developers have to provide fool-proof methods for distributing and installing an application. This usually includes writing multiple installers including those that can run directly on the mobile device and those that can be run from a desktop computer.
- **Resources:** Mobile phones and PDAs are always scarce on resources. Users will not tolerate applications that use up most of their file space or take long to start up.
- **Supporting different devices:** Support for a practical range of target devices is probably the most critical issue for any mobile phone software developer. While typical PIM (personal information manager) applications can easily adapt to various devices, application that require low level OS and hardware access quickly run into problems such as driver bugs or unexpected hardware platforms (such as screen and camera image layouts). In personal discussions

mobile game developers reported managing more than hundred build targets. One can find roughly ten different screen resolutions on today's mobile phones – writing an application that supports screen resolutions of 160x120 as well as 480x640 pixels is therefore a non-trivial task. CPU power can vary in the range of an order of magnitude too.
Accurate pose tracking requires calibrating for specific camera types. Unfortunately, device series and brands can often not be reliably detected in practice, which prevents automatic usage of calibration files for known devices.

- **Sticking to UI conventions:** While it is tantalizing to create unique user interfaces that are optimal for the specific applications, it is more important to stick to the user interface conventions of the target device. As learned the hard way during development and deployment of the Signpost 2007 application, users prefer well known UI concepts over more optimized, but not standard conform alternatives.
- **Behaving nicely:** Applications must behave nicely, which includes not taking over full control of the target device and reacting to device specific events. E.g. users will not tolerate missing phone calls, due to an application that did not pass through notification messages. Some devices have special features such as automatically rotating screen content when a keyboard is slides out. Naturally users expect applications to follow this behavior – even if it is of low usage for the actual application.
- **Driver issues:** Today, AR applications are still untypical for mobile phones, which includes that these applications use the hardware differently than most other applications do. A major problem often involves accessing the built-in camera. While these cameras often have compelling specs, the built-in applications are usually they only ones actually using the camera. Unfortunately this means that drivers are often buggy and device creators use proprietary methods to take full advantage of the camera. Therefore, in practice many devices do not deliver the full potential or behave erroneously when using the standard APIs.
- **Considering non-optimal environments:** Researchers usually know very well under which circumstances AR software runs fine and where it performs poor. In contrast, end users will run applications in unpredictable situations and still expect it to function as promised.
- **Developing on the PC, final testing on the device:** Debugging on embedded devices is cumbersome and on some platforms such as Symbian not possible without expensive tools. Doing as much work as possible on the PC should be

preferred since it results in faster development cycles and often even cleaner code due to the increased portability requirements.

8.2 Guidelines on Performance

Despite the many advantages mobile phones offer as a platform for mobile Augmented Reality, the poor processing capabilities demand special care during application development. From our experience of developing multiple applications for broad range of mobile devices we arrived at the following set of guidelines of how to achieve optimal performance:

- **Sequential vs. parallel:** Even though ARM CPUs usually do not have parallel execution units, many operations such as reading the camera or network communication can be successfully accelerated using multi-threading because they are I/O rather than CPU bound.
- **Camera resolution:** Some high-end phones can deliver video streams at resolutions up to 640x480 pixels. In practice these videos are limited in camera quality. Unlike high quality PC cameras, there is only a minimal improvement in tracking quality. The reasons for this are the low quality lenses and camera sensors with high noise levels.
- **Multi-marker tracking:** Using high quality cameras on PCs allows stable single marker tracking. Larger markers can sometimes compensate for the lower image quality of mobile phone cameras, but user interface designs often prevent this. Multi-marker tracking provides highly stable tracking - at the expense of higher computational costs though.
- **Id-based markers:** With a growing number of markers known to the tracking system, the process of template matching can seriously degrade overall performance. Id-based markers do not share this weakness and are always faster to detect than template markers.
- **Camera pixel formats:** While on the PC applications tend to read camera frame in convenient standard formats, such as RGB, this is not the case on mobile phones, which often expose lower quality and stronger compressed format. Some of these formats, such as YUV12 store the luminance values image data in a separate block, which is ideal for vision based tracking.
- **Compilers:** As shown in chapter 3.4 some compilers can increase tracking speed in certain situations. In older versions of ARToolKitPlus which contained more floating point code we noticed speedups up to 70%.

Unfortunately these compilers are often expensive and generated code, which only works on specific CPUs. Furthermore we also noticed slowdowns, such as when compiling Klimt with the Intel compiler.

8.3 Future Work

While the current Augmented Reality framework supports the creation of practical AR setups, the resulting applications are usually still far from perfect. A major topic for future research will be tracking. Although robust tracking on mobile phones is possible today, the necessity for fiducial markers hinders deployment to an even broader range of applications.

In many environments attaching markers is not an option. This includes large areas, such as for city-wide games and applications as well as environments where attaching markers is not allowed or impractical.

Therefore, a next major step will be moving from marker tracking to natural feature tracking. Other than marker tracking, natural feature tracking is still a hot topic even on high performance platforms. The low processing capabilities of mobile phones therefore provide hard preconditions for this task. While we do not expect mobile phone CPUs to improve a lot in the next years, other processing units are currently being integrated into these devices, such as signal processors (DSPs), graphics processing units (GPUs) and multi-media units.

Due to the strong request for graphical applications and games on phones, the usage of GPUs for general purpose processing seems to be a promising approach. The current generation of GPUs with its fixed function pipelines (OpenGL ES 1.x) is only suitable for rendering. The next generation though will introduce freely programmable pixel shaders and therefore will allow outsourcing simple general purpose calculations.

Another topic for future research is massive multi user AR applications. The foundation for such applications has been laid in this thesis, making it possible to practically target applications with thousands of users. However, a significant amount of research will have to go into designing new user interface concepts for such scenarios.

When targeting the aforementioned multi user applications, content creation will become an even more important issue. While many commercial game companies currently set their hopes on developments such as COLLADA, it is unclear, how well these developments suit the requirements of Augmented Reality applications.

Similarly, content distribution will become another hot topic. Years of research has gone into data streaming on desktop and VR platforms, but little work has targeted mobile phones.

The solutions developed in the course of this thesis have created interest from commercial entities. While in its infancy these solutions have been only used by project partners, Studierstube ES is currently in the process of transitioning from a research prototype into a commercially available and supported AR framework. The author of this thesis therefore believes that in the near future Augmented Reality will become a viable user interface for a wide, public audience.

Chapter 9

Appendix

9.1 Studierstube ES Example Applications

9.1.1 Minimal Example

The following code excerpts show how to write a minimal Studierstube ES application. This example renders a virtual cube on top of a marker. We do not require any C++ code to implement such an application with StbES. Hence, we only require a configuration file (see Figure 9.1) that sets up StbES and a scene file (see Figure 9.2) that contains the field connections to the tracking system and the cube itself.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<StbES>
  <!-- Define type and target for logging -->
  <Logging file="data/log.txt" draw-fps="false" level="INFO" />

  <!-- Specify to not log memory usage during startup
      (only useful for debuuging) -->
  <Memory logging="FALSE" />

  <!-- Setup an application window. we use EGL so that hardware acceleration
       can be used if available. -->
  <Window type="Standard" rotation="0" />

  <!-- Define the render target to use Onscreen memory -->
  <RenderTarget type="VidMem" />

  <!-- Use DirectShow for video input -->
  <Video type="DSVideoCE" />

  <!-- Specify the root scene file to load -->
  <Scene file="data/scene.xml" />

  <!-- Setup the widget manager (responsible for 2D GUI, etc.
       and define which bitmap font to load. -->
  <WidgetManager font="data/raster_8x12_256.png" char-width="8" char-height="16" />

  <!-- Setup the tracking module. For this demo application we want to
       tracking debug infos on the screen -->
  <Tracker displayInfos="true" >

    <!-- Setup the ARTK+ tracker. Specify the camera file and to cache the
         lookup table. Set marker border width to half of the default size -->
    <TrackerARToolKitPlus cam-file="data/CE.cal" cache-lut="true" border-width="0.125"

      <!-- Only use one target with an ID of 200 and a size of 80mm -->
      <TrackingTarget name="TestTarget" id="200" size="80"/>
    </TrackerARToolKitPlus>
  </Tracker>

</StbES>
```

Figure 9.1: Config file for Windows CE for a minimal application that does not require loading an "application file" (DLL).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Scene name="MyScene">

  <!-- Defines the video background to show the camera video stream -->
  <Background backgroundType="BACKGROUND_IMAGE" />

  <!-- Links the camera projection matrix of the tracking system
       into our scene setup -->
  <MatrixCamera projMatrix="REF Tracker.projMatrix" />

  <!-- Makes sure that the transformation changes are undone and disables
       rendering of the cube when the marker (taregt) is not visible -->
  <TransformSeparator active="REF TestTarget.visible">

    <!-- Retrieves the pose data of the tracked target
         and applies it to the scene -->
    <MatrixTransform matrix="REF TestTarget.matrix" />

    <!-- Makes sure that lighting changes are undone -->
    <LightSeparator>

      <!-- Adds a directional light to the scene -->
      <DirectionalLight direction="0 -0.5 0.3" />

      <!-- Moves the virtual cube 40 unit (half the size) upwards -->
      <Transform name="PlayerTransform" translation="0 0 40" />

      <!-- Renders the virtual cube with a size of 80 milimeters -->
      <Cube width="80" height="80" depth="80" />
    </LightSeparator>
  </TransformSeparator>

</Scene>
```

Figure 9.2: Commented scene file for a minimal application that renders a virtual cube on a marker

9.1.2 Model Viewer Application

This example is an extended version of the simple application. It loads an actual model file and displays it on the marker. Furthermore it allows the user to rotate the model and optionally start/stop animations. To achieve this, writing a short C++ application is required. A single class is implemented that derives from StbES::Application (as all StbES applications do) and from StbES:: IRawInputListener to register for key presses. The C++ header (see Figure 9.3) and source files (see Figure 9.4) are listed here. The configuration and scene files are almost identical to the minimal example above and therefore skipped.

```
#ifndef __STBES_PLAYER_HEADERFILE__
#define __STBES_PLAYER_HEADERFILE__

#include <StbES/Kernel/Application.h>
#include <StbES/Window/IRawInputListener.h>
#include <StbES/Math/Math.h>

namespace StbES {

class SgAnimator;
class SgTransform;

class Player : public Application, public IRawInputListener
{
public:
  Player();
  virtual ~Player();

  // Implement the Application interface
  void init();
  void render2D();

protected:
  // Implement the IRawInputListener interface
  void ril_MouseDown(int nX, int nY)  {}
  void ril_MouseUp(int nX, int nY)  {}
  void ril_MouseMove(int nX, int nY)  {}
  void ril_KeyDown(int nKey);
  void ril_KeyUp(int nKey);
  void ril_Char(int nKey)  {}

  Vector<SgAnimator*> anims;         // Stores optional animations
  SgTransform*        transform;     // Pointer to main transform

  float               rotZ;          // Rotation value around z-axis
  bool                lDown, rDown;  // Left and right key states
};

}  // namespace StbES

#endif //__STBES_PLAYER_HEADERFILE__
```

Figure 9.3: Header file of the model viewer application.

```
#include "Player.h"
#include <StbES/Kernel/Kernel.h>
#include <StbES/Kernel/FrameStats.h>
#include <StbES/Math/MathTool.h>
#include <StbES/Scenegraph/ScenegraphDB.h>
#include <StbES/Scenegraph/SgAnimator.h>
#include <StbES/Scenegraph/SgTransform.h>

// Implements entry point into module (DLL) to
// allow StbES to load this application
IMPLEMENT_APPLICATION_ENTRYPOINT(Player);

namespace StbES {

Player::Player() : transform(NULL), rotZ(0.0f)
{
  lDown = rDown = false;
}

Player::~Player()
{
  // Unregister from capturing key presses when the application exits
  kernel->getWindow()->unregisterRawInputListener(this);
}

// Called after the application was fully loaded
void
Player::init()
{
  if(SgNode* root = kernel->getSceneRoot())
  {
    SgNode* node = NULL, *parent;
    SgNode::TYPE classType = SgAnimator::getClassType();

    // Find all animator nodes in the scene-graph
    while(node = ScenegraphDB::findNodeByIsOfType(root, classType, parent, node))
      anims.push_back(SgNode::cast<SgAnimator>(node));

    // Find the main transform node in the scene-graph
    if(SgNode* trans = ScenegraphDB::findNodeByName(root, "PlayerTransform"))
      transform = SgNode::cast<SgTransform>(trans);
  }

  // Register to receive all key press and release events
  kernel->getWindow()->registerRawInputListener(this);
}

// Called each time before the scene-graph is traversed
void
Player::render2D()
{
  float dt = Kernel::getInstance()->getFrameStats().getFrameDuration().getFloat();

  if(lDown)
    rotZ -= dt;
  if(rDown)
    rotZ += dt;

  if(transform)
  {
    Vec4 quat = MathTool::quaterionFromAxisAngle(Vec4(0.0f, 0.0f, 1.0f, rotZ));
    transform->rotation.setValue(quat);
  }
}

// Called when the user pressed down a button
void
Player::ril_KeyDown(int nKey)
{
  switch(nKey)
  {
  case IRawInputListener::KEY_LEFT:
    lDown = true;
    break;

  case IRawInputListener::KEY_RIGHT:
    rDown = true;
    break;

  case IRawInputListener::KEY_PREVIOUS:
    // Play all animations
    for(size_t i=0; i<anims.size(); i++)
      anims[i]->running.setValue(!anims[i]->running.getValue());
    break;

  case IRawInputListener::KEY_NEXT:
    // Reset all animations
    for(size_t i=0; i<anims.size(); i++)
      anims[i]->reset();
    break;
  }
}

// Called when the user releases a button
void
Player::ril_KeyUp(int nKey)
{
  switch(nKey)
  {
  case IRawInputListener::KEY_LEFT:
    lDown = false;
    break;

  case IRawInputListener::KEY_RIGHT:
    rDown = false;
    break;
  }
}

}  // namespace StbES
```

Figure 9.4: Source file of the model viewer application.

9.2 Pose Refinement

```
SE3 optimizePose(const SE3& initialPose, const vector<Vector<3> >& points,
                 const vector<Vector<2> >& observations)
{
    // jacobian an error variables, we build these directly,
    // therefore less storage requirements!
    Matrix<6> JTJ;
    Vector<6> JTE;
    Zero(JTJ);
    Zero(JTE);

    // build jacobian and error vector
    for(int i = 0; i < points.size(); ++i){
        Matrix<2,6> Jacobian; // the jacobian of the parameters for this one point
        Matrix<2,3> Jacobian_Point; // the jacobian with respect to point parameters

        // computes all the jacobians and the point projection in one go :)
        // much more efficient than the above stuff
        Vector<2> projectedPoint = initialPose.transform_and_project(points[i],
                                                              Jacobian_Point,
                                                              Jacobian);
        // add local part to JTJ
        // JTJ += Jacobian.T() * Jacobian;
        // faster version avoiding temporary in += evaluation
        add_product(Jacobian.T(), Jacobian, JTJ);
        // add local part to JTE, this is JT * error
        // JTE += Jacobian.T() * (observations[i] - projectedPoint);
        add_product(Jacobian.T(), observations[i] - projectedPoint, JTE);
    }

    // compute inverse with Cholesky decomposition
    Cholesky<6> chol(JTJ);

    // compute inverse * error with backwards substitution
    Vector<6> delta = chol.inverse_times(JTE);

    // left multiply to the initialPose to create new SE3
    return SE3::exp(delta) * initialPose;
}
```

Source code for refining a given pose using Gauss-Newton iteration (optimizePose() implements one refinement step). Source code provided by Gerhard Reitmayr and uses the TooN numeric library.

9.3 Questionnaires

9.3.1 Virtuoso – Survey on Realism of the Virtual Character

The following three pages contain the questionnaires handed out for the user study performed at HitLAB New Zealand in March 2006.

Survey Questions

Date:

User ID:	**Gender:**	**Age:**
	Start Time:	**End Time:**

How familiar are you with handheld PDAs (circle one)?:

1	2	3	4	5
Not very Familiar				Very Familiar

How familiar are you with famous art works (circle one)?:

1	2	3	4	5
Not very Familiar				Very Familiar

You are going to play a game called “Virtuoso”. In this game you will see virtual items shown on a PDA screen overlaid on markers in the real world. These markers represent a timeline. Your task is to put the items on the timeline into the correct order and find out as much about the items as possible. Left on the timeline means older, while right on the timeline means newer. To do that you can pick items of the wall by clicking on the item on the PDA screen. The item is then put onto your PDA and you can read some basic information about that artwork. There is no hurry, the most important thing it to learn as much about the artwork as possible as you will be tested on what you have learned.

If you want to find out more about an item you can ask for help from a virtual character called Mr.Virtuoso. To do this put it onto Mr.Virtuoso’s desk and he will come and explain in more detail what he knows about that specific artwork. In some cases he will appear in person, at other times you will only hear his voice or see additional text. You should try and use Mr. Virtuoso to find out about every item.

If you put an item onto its correct spot on the timeline it will reveal its date of creation and you can no remove that item from the timeline. This also means that you cannot ask Mr.Virtuoso about items that are already correctly positioned!

You will play five rounds of the game. Each time there will be different items to sort. After each round you will have to answer some questions about the items and how you felt about the PDA application.

Remember: Learn as much as you can about each item, rather than trying to place them on the timeline as quickly as possible.

Survey Questions **Condition/Itemset: / 5** **Time for Game:**
Correct Timeline: **# asked Mr.Virtuoso:**

On a scale of 1 to 7 please circle the number according to how much you agree or disagree with the following statements:

1) I enjoyed playing the art history game

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

2) Mr.Virtuoso seemed real to me

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

3) Mr.Virtuoso was helpful for completing the task

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

4) The PDA interface was easy to use

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

5) The task was easy to solve

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

6) Mr.Virtuoso improved the overall experience.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

7) Mr.Virtuoso seemed to be part of the real world for me.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

8) I felt that I learned new facts about art items from the art history game.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

9) I found Mr.Virtuoso to be friendly.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

Notre Dame du Haut, Ronchamp was created in

o 1920 o 1930 o 1950

Il Gesu is famous for its

o Façade o Statues o Size

Mozart died at the age of

o 31 o 35 o 42

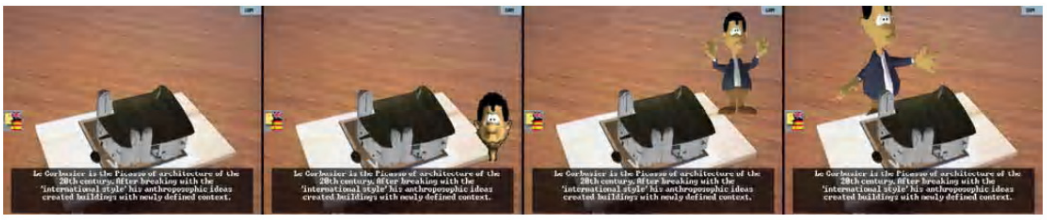
Passion of the Christ was produced in

o Aramaic o Greek o Hebrew

Ranking

You have just experienced an art history application where a virtual character (Mr.Virtuoso) helped you to complete the task. Mr.Virtuoso appeared in five different ways:

A: As a text-only window
B: As a text window with a spoken voice
C: As a text window with a spoken voice and a 2D picture of Mr.Virtuoso's head
D: As a text window with a spoken voice and a screen-aligned Mr.Virtuoso
E: As a text window with a spoken voice and an augmented reality Mr.Virtuoso


Le Corbusier is the Picasso of architecture of the 20th century. After breaking with the 'international style' his anthroposophic ideas created buildings with newly defined context.

Please rank the conditions in order for the following questions. Give each condition a unique number from 1 to 5 where 1 = highest, 5 = lowest.

1) Rank the conditions in order of how real Mr.Virtuoso seemed:

A:	B:	C:	D:	E:

2) Rank the conditions in order of how much fun they were:

A:	B:	C:	D:	E:

3) Rank the conditions in order of how much you learnt:

A:	B:	C:	D:	E:

4) Rank the conditions in order of how helpful Mr.Virtuoso was:

A:	B:	C:	D:	E:

Further comments:

9.3.2 Virtuoso – Survey of Collaboration with Handheld AR

The following eight pages contain the questionnaire used in the study performed in 2006 at the Graz University of Technology. The evaluation compared the handheld AR version of Virtuoso against a desktop version, implemented using Adobe Flash and a paper version of the game.

Virtuoso Fragebogen

Gruppe: _______	**Geschlecht:** _______
User ID: _______	**Alter:** _______

Wie gut kennst du dich mit PDAs aus?:

Gar nicht ☐ ☐ ☐ ☐ ☐ Sehr gut

Wie gut kennst du dich mit historischen Kunstwerken aus?:

Gar nicht ☐ ☐ ☐ ☐ ☐ Sehr gut

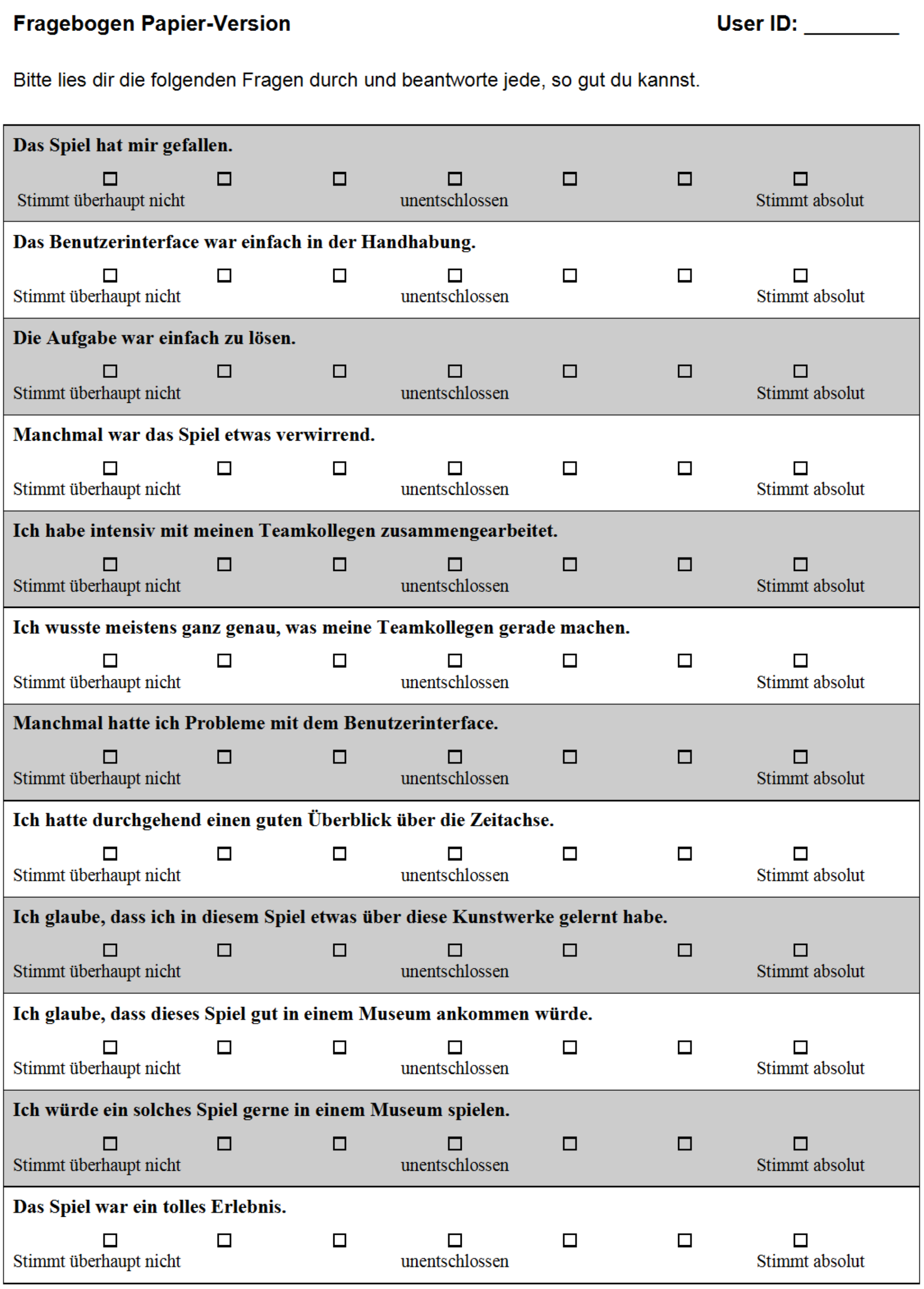
Einleitung

Du wirst nun ein Spiel namens “Virtuoso” spielen. In diesem Spiel geht es darum, historische Kunstwerke auf einer Zeitachse zu sortieren (links bedeutet älter, rechts bedeutet neuer). Du wirst drei verschiedene Versionen des Spiels spielen, aber das Ziel ist immer das gleiche: als Team gemeinsam die Kunstwerke in ihre korrekte Reihenfolge zu bringen.

Im Spiel kannst du Objekte aus der Zeitachse herausnehmen und an einen anderen freien Platz legen. Du kannst immer nur ein Objekt auf einmal halten. Wenn du ein Objekt nimmst, bekommst du grundlegende Informationen über jenes Kunstwerk. Wenn du mehr darüber wissen möchtest, kannst du Mr.Virtuoso, unseren virtuellen Kunsthistoriker, um Rat fragen.

Im Spiel geht es nicht um Zeit. Es ist nicht das Ziel, die Aufgabe so schnell wie möglich zu lösen. Stattdessen sollst du das Spiel und Kunstwerke kennenlernen und mit deinen Teamkollegen zusammenarbeiten. Nach jedem Spiel bekommst du einen Fragebogen, um zu sehen, wie dir die aktuelle Variante gefallen hat und um den Lerneffekt zu prüfen (jede Einzelperson beantwortet inhaltliche Fragen zum Spiel). Am Ende gewinnt jene Gruppe, bei der alle Spieler zusammen am meisten Fragen richtig beantwortet haben.

Noch mal: Lieber mehr über die Kunstwerke lernen und Information an die Teamkollegen weitergeben, als so schnell wie möglich die Zeitachse aufräumen!



Fragebogen Papier-Version **User ID: ________**

Bitte lies dir die folgenden Fragen durch und beantworte jede, so gut du kannst.

Das Spiel hat mir gefallen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Das Benutzerinterface war einfach in der Handhabung.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Die Aufgabe war einfach zu lösen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Manchmal war das Spiel etwas verwirrend.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich habe intensiv mit meinen Teamkollegen zusammengearbeitet.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich wusste meistens ganz genau, was meine Teamkollegen gerade machen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Manchmal hatte ich Probleme mit dem Benutzerinterface.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich hatte durchgehend einen guten Überblick über die Zeitachse.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich glaube, dass ich in diesem Spiel etwas über diese Kunstwerke gelernt habe.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich glaube, dass dieses Spiel gut in einem Museum ankommen würde.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich würde ein solches Spiel gerne in einem Museum spielen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Das Spiel war ein tolles Erlebnis.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Wissensfragen:

"Königin Theodora und ihr Gefolge" ist ein

o Gemälde o Mosaik o Skulptur

Die Skulptur von Rahotep und seiner Frau ist besonders bekannt für

o Ihre Größe o natürliche Hautfarbe o perfekte Proportionen

Il Gesu ist berühmt für

o die Façade o Statuen o Größe der Kirche

Mozart starb im Alten von

o 31 Jahren o 35 Jahren o 42 Jahren

„Passion of the Christ" wurde produziert in

o aramäisch o griechisch o hebräisch

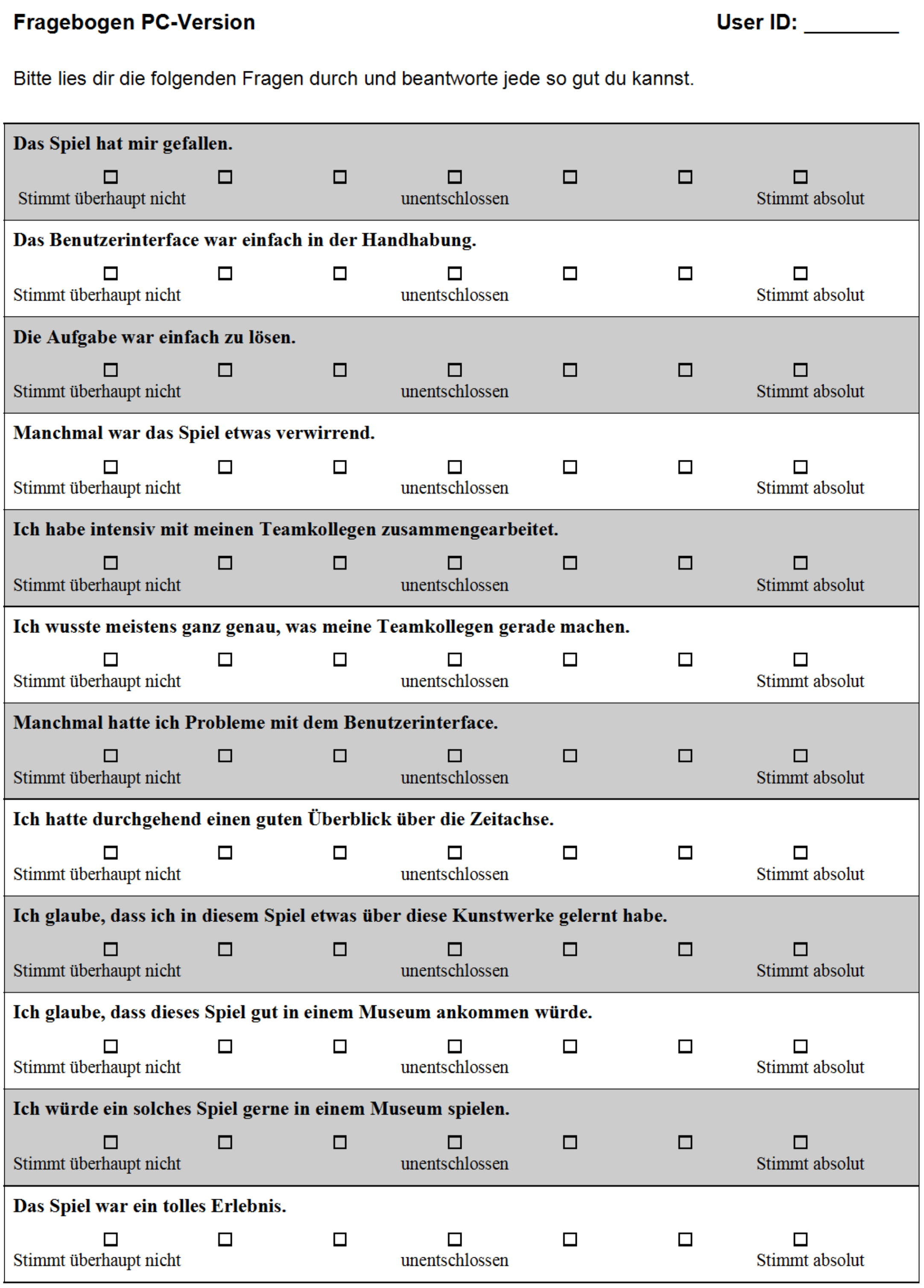
„Die gescheiterte Hoffnung" wurde gemalt von

o Caspar David Friedrich o Philipp Otto Runge o William Turner

Paul Gaugin was a der führende Maler im

o Post-Impressionismus o Expressionismus o Symbolismus

Anmerkungen:



Fragebogen PC-Version **User ID: ________**

Bitte lies dir die folgenden Fragen durch und beantworte jede so gut du kannst.

Das Spiel hat mir gefallen.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Das Benutzerinterface war einfach in der Handhabung.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Die Aufgabe war einfach zu lösen.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Manchmal war das Spiel etwas verwirrend.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Ich habe intensiv mit meinen Teamkollegen zusammengearbeitet.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Ich wusste meistens ganz genau, was meine Teamkollegen gerade machen.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Manchmal hatte ich Probleme mit dem Benutzerinterface.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Ich hatte durchgehend einen guten Überblick über die Zeitachse.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Ich glaube, dass ich in diesem Spiel etwas über diese Kunstwerke gelernt habe.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Ich glaube, dass dieses Spiel gut in einem Museum ankommen würde.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Ich würde ein solches Spiel gerne in einem Museum spielen.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Das Spiel war ein tolles Erlebnis.

☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Wissensfragen:

Die Trajansäule zeigt den Krieg gegen

o Gallier o Karthager o Daker

Giotto, der die "Thronende Madonna" malte, lebte im

o 13. Jhdt. o 15. Jhdt. o 17. Jhdt.

Der Altar von Isenheim wurde gemalt im

o Impressionismus o Expressionismus o Surrealismus

Caravaggios "Berufung des Apostels Matthäus" ist bekannt für

o genaue Pinselstrichführung o falsche Perspektive o heilige Szenen in alltäglicher Umgebung

Pablo Picasso begründete den

o Symbolismus o Kubismus o Dadaismus

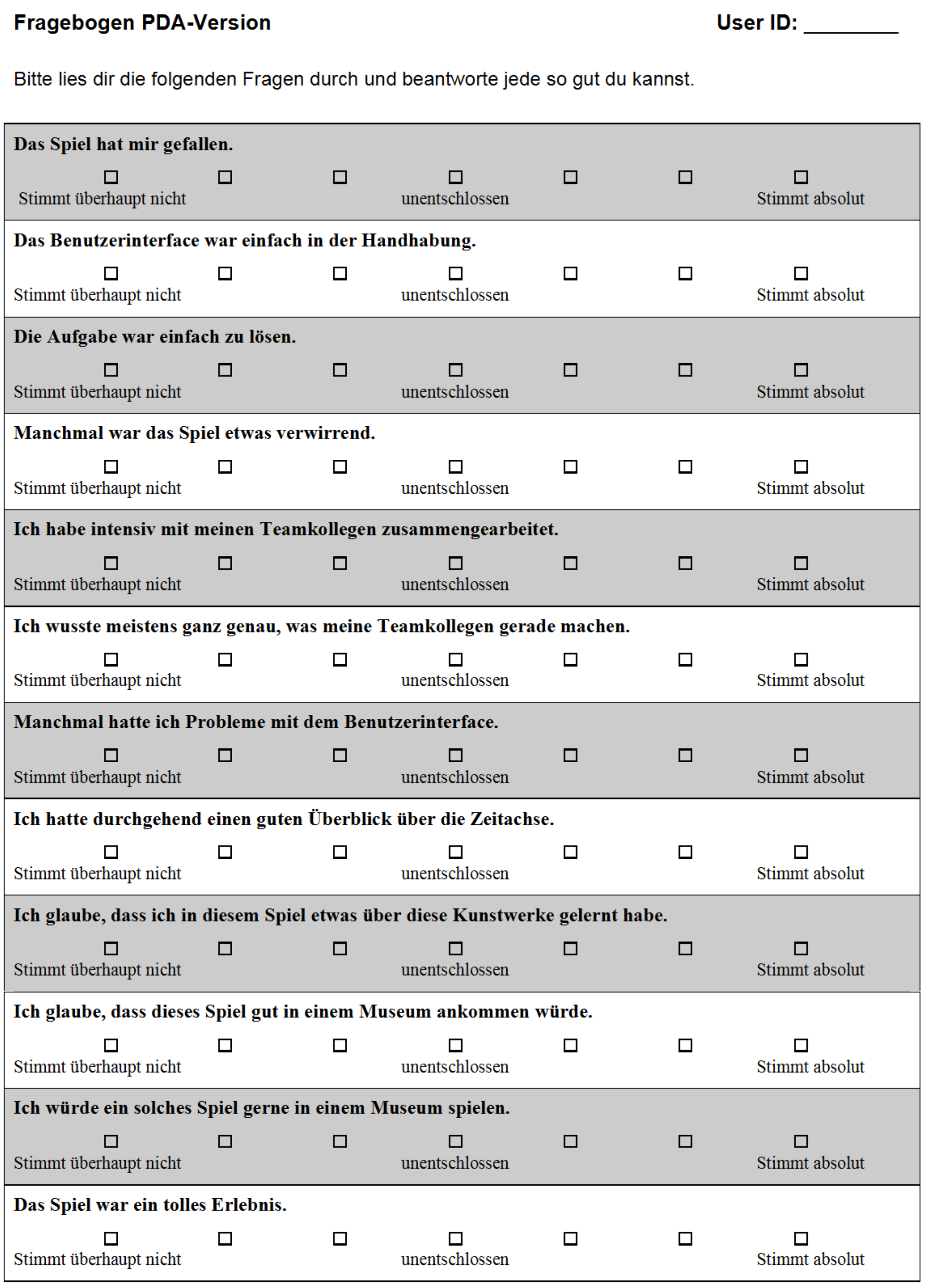
„Komposition mit Rot, Blau und Gelb" wurde gemalt von

o Russe Malewitch o Piet Mondrian o Theo van Doesburg

"Das Ende des 20. Jahrhunderts" ist von

o Antonio Saura o Norman Lindsay o Joseph Beuys

Anmerkungen:



Fragebogen PDA-Version **User ID: ________**

Bitte lies dir die folgenden Fragen durch und beantworte jede so gut du kannst.

Das Spiel hat mir gefallen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Das Benutzerinterface war einfach in der Handhabung.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Die Aufgabe war einfach zu lösen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Manchmal war das Spiel etwas verwirrend.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich habe intensiv mit meinen Teamkollegen zusammengearbeitet.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich wusste meistens ganz genau, was meine Teamkollegen gerade machen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Manchmal hatte ich Probleme mit dem Benutzerinterface.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich hatte durchgehend einen guten Überblick über die Zeitachse.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich glaube, dass ich in diesem Spiel etwas über diese Kunstwerke gelernt habe.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich glaube, dass dieses Spiel gut in einem Museum ankommen würde.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Ich würde ein solches Spiel gerne in einem Museum spielen.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut
Das Spiel war ein tolles Erlebnis.						
☐	☐	☐	☐	☐	☐	☐
Stimmt überhaupt nicht			unentschlossen			Stimmt absolut

Wissensfragen:

Die “Venus von Willendorf” wurde gefunden in

o Österreich o Deutschland o Schweiz

Das “Parthenon” wurde gebaut um folgenden Gott anzupreisen

o Hestia o Hera o Athena

Der “Stephansdom” wurde gebaut in

o Renaissance o Gotik o Romantik

Michelangelo schuf seine Werke in folgender Epoche

o Renaissance o Mannerismus o Barok

Das “Taj Mahal” wurde gebaut im

o 13. Jhdt. o 15. Jhdt. o 17. Jhdt.

Mozart starb im Jahr

o 1547 o 1683 o 1791

Die “Wallfahrtskirche bei Ronchamp“ wurde gebaut in

o 1920 o 1930 o 1950

Anmerkungen:

Ranking **User ID: ________**

Du hast gerade drei verschiedene Varianten eines Kunst-Lernspiels gespielt.

- Am PC-Bildschirm
- Ausdrucke auf Karton
- Mit PDAs

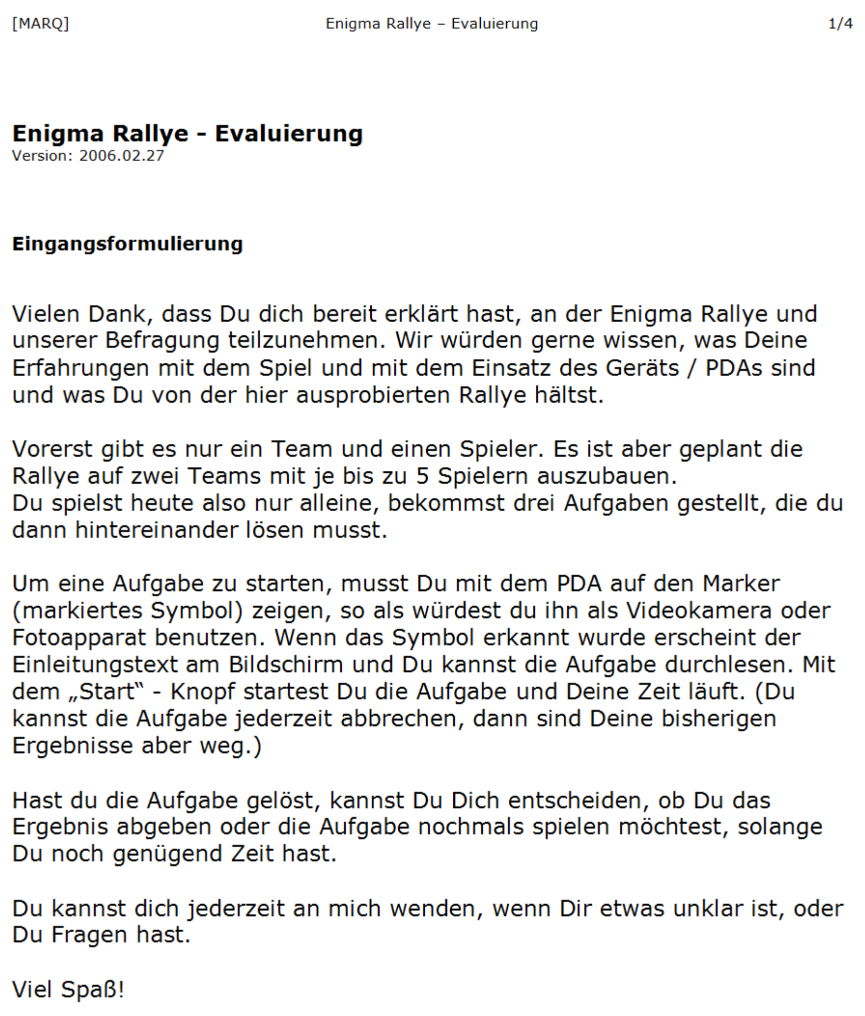
Bitte bewerte die folgenden Aussagen mit Zahlen von 1 bis 3, wobei **1 wenig** bedeutet und **3 viel**. Jede Zahl darf nur einmal pro Aussage verwendet werden (eindeutige Reihenfolge).

Bewerte, wie viel Spaß das Spiel gemacht hat.	Am PC-Bildschirm: Ausdrucke auf Karton: Mit PDAs:
Bewerte, wie viel du gelernt hast.	Am PC-Bildschirm: Ausdrucke auf Karton: Mit PDAs:
Bewerte, wie einfach das Spiel zu spielen war.	Am PC-Bildschirm: Ausdrucke auf Karton: Mit PDAs:
Bewerte, wie intensiv Du mit deinen Teamkollegen zusammen gearbeitet hast.	Am PC-Bildschirm: Ausdrucke auf Karton: Mit PDAs:
Bewerte, wie gut dein Überblick über die Zeitachse war.	Am PC-Bildschirm: Ausdrucke auf Karton: Mit PDAs:
Bewerte, wie sehr du glaubst, dass das jeweilige Spiel eine Museumsausstellung aufwerten würde.	Am PC-Bildschirm: Ausdrucke auf Karton: Mit PDAs:

Weitere Anmerkungen:

9.3.3 Questionnaire of the MARQ Evaluation at Technisches Museum Wien

The following 4 pages contain the questionnaire handed out to all visitors who played the Mobile Augmented Reality Quest game at Technisches Museum Wien in 2006.



[MARQ] Enigma Rallye – Evaluierung 1/4

Enigma Rallye - Evaluierung
Version: 2006.02.27

Eingangsformulierung

Vielen Dank, dass Du dich bereit erklärt hast, an der Enigma Rallye und unserer Befragung teilzunehmen. Wir würden gerne wissen, was Deine Erfahrungen mit dem Spiel und mit dem Einsatz des Geräts / PDAs sind und was Du von der hier ausprobierten Rallye hältst.

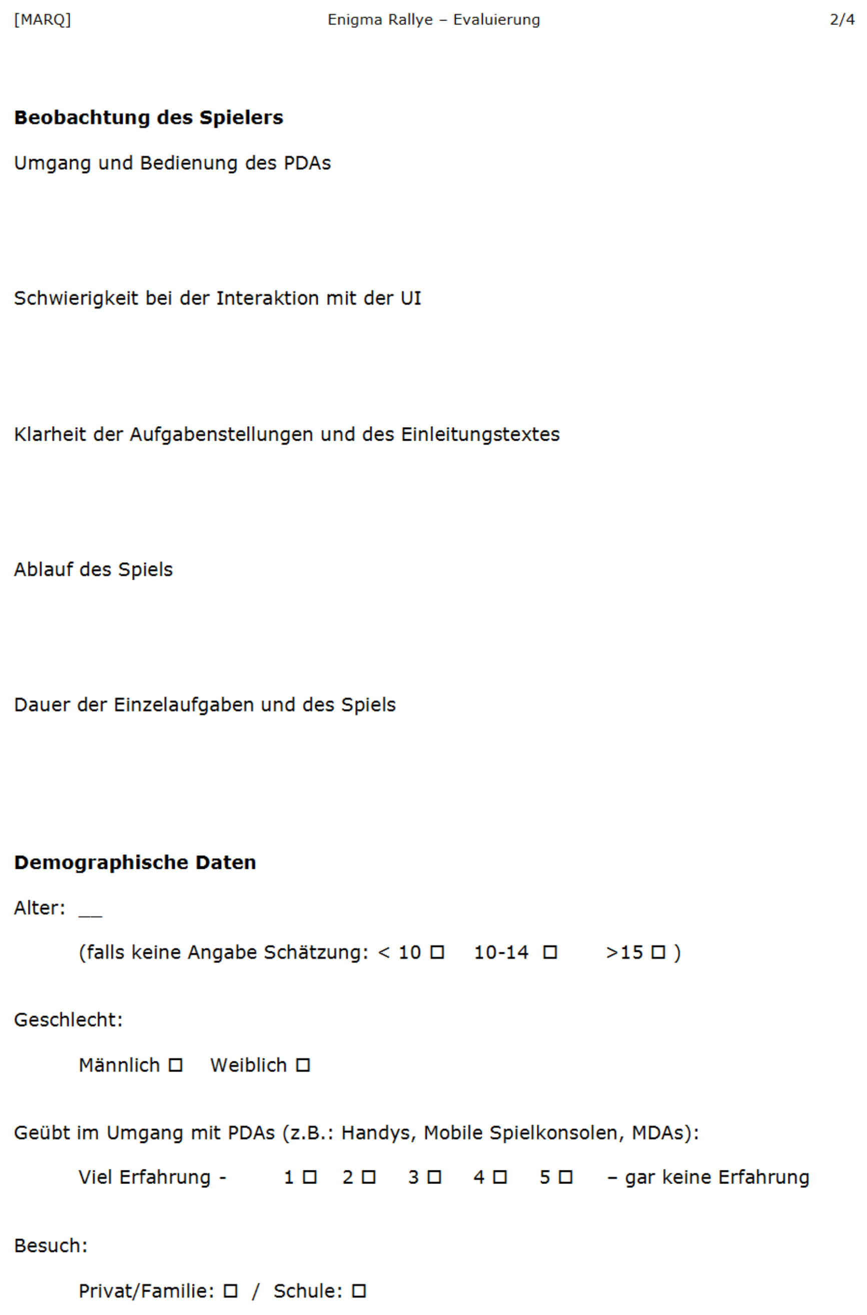
Vorerst gibt es nur ein Team und einen Spieler. Es ist aber geplant die Rallye auf zwei Teams mit je bis zu 5 Spielern auszubauen.
Du spielst heute also nur alleine, bekommst drei Aufgaben gestellt, die du dann hintereinander lösen musst.

Um eine Aufgabe zu starten, musst Du mit dem PDA auf den Marker (markiertes Symbol) zeigen, so als würdest du ihn als Videokamera oder Fotoapparat benutzen. Wenn das Symbol erkannt wurde erscheint der Einleitungstext am Bildschirm und Du kannst die Aufgabe durchlesen. Mit dem „Start“ - Knopf startest Du die Aufgabe und Deine Zeit läuft. (Du kannst die Aufgabe jederzeit abbrechen, dann sind Deine bisherigen Ergebnisse aber weg.)

Hast du die Aufgabe gelöst, kannst Du Dich entscheiden, ob Du das Ergebnis abgeben oder die Aufgabe nochmals spielen möchtest, solange Du noch genügend Zeit hast.

Du kannst dich jederzeit an mich wenden, wenn Dir etwas unklar ist, oder Du Fragen hast.

Viel Spaß!



[MARQ] Enigma Rallye – Evaluierung 2/4

Beobachtung des Spielers

Umgang und Bedienung des PDAs

Schwierigkeit bei der Interaktion mit der UI

Klarheit der Aufgabenstellungen und des Einleitungstextes

Ablauf des Spiels

Dauer der Einzelaufgaben und des Spiels

Demographische Daten

Alter: __

(falls keine Angabe Schätzung: < 10 ☐ 10-14 ☐ >15 ☐)

Geschlecht:

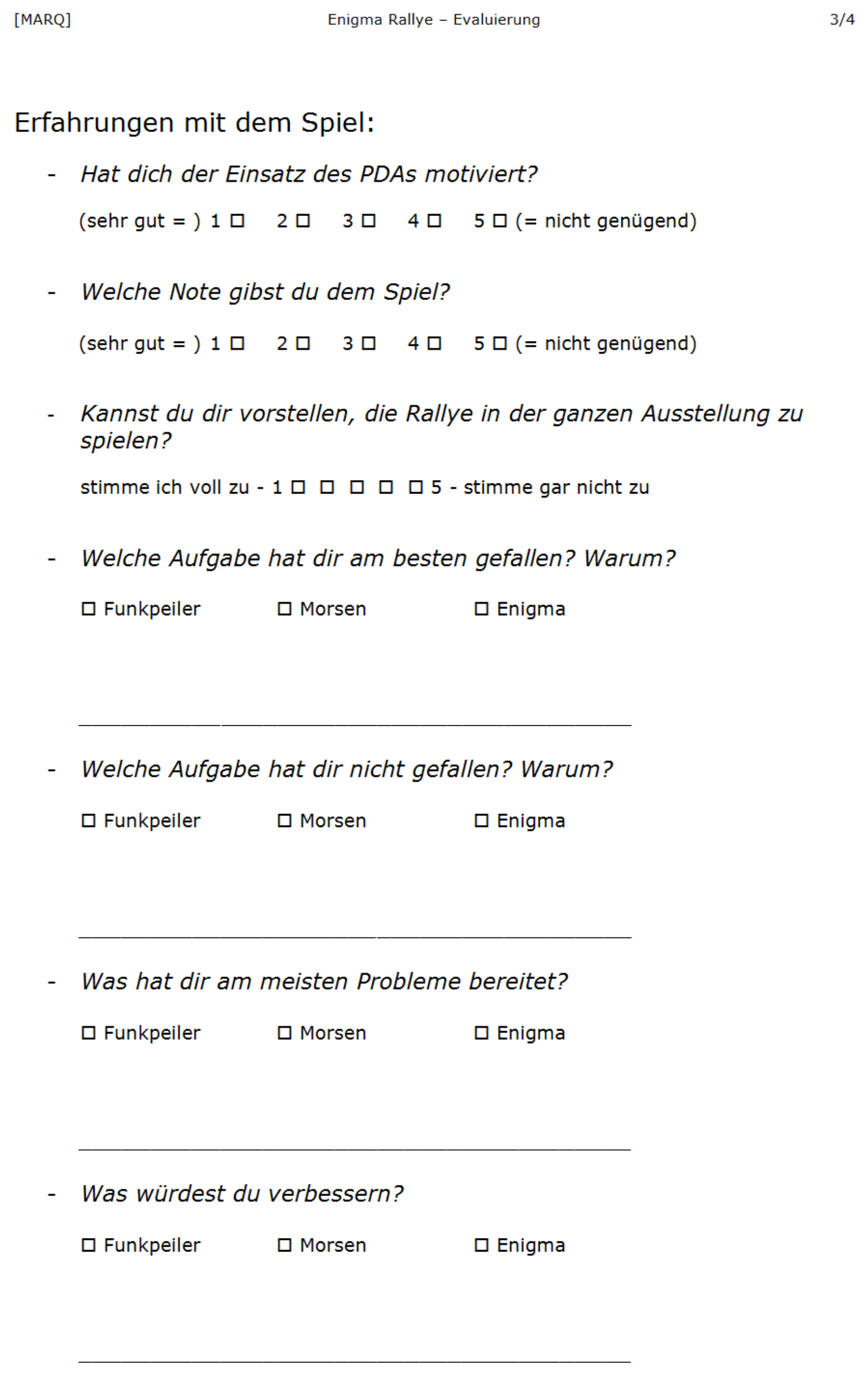
Männlich ☐ Weiblich ☐

Geübt im Umgang mit PDAs (z.B.: Handys, Mobile Spielkonsolen, MDAs):

Viel Erfahrung - 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ – gar keine Erfahrung

Besuch:

Privat/Familie: ☐ / Schule: ☐



[MARQ] Enigma Rallye – Evaluierung 3/4

Erfahrungen mit dem Spiel:

- *Hat dich der Einsatz des PDAs motiviert?*

  (sehr gut =) 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ (= nicht genügend)

- *Welche Note gibst du dem Spiel?*

  (sehr gut =) 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ (= nicht genügend)

- *Kannst du dir vorstellen, die Rallye in der ganzen Ausstellung zu spielen?*

  stimme ich voll zu - 1 ☐ ☐ ☐ ☐ ☐ 5 - stimme gar nicht zu

- *Welche Aufgabe hat dir am besten gefallen? Warum?*

  ☐ Funkpeiler ☐ Morsen ☐ Enigma

  ____

- *Welche Aufgabe hat dir nicht gefallen? Warum?*

  ☐ Funkpeiler ☐ Morsen ☐ Enigma

  ____

- *Was hat dir am meisten Probleme bereitet?*

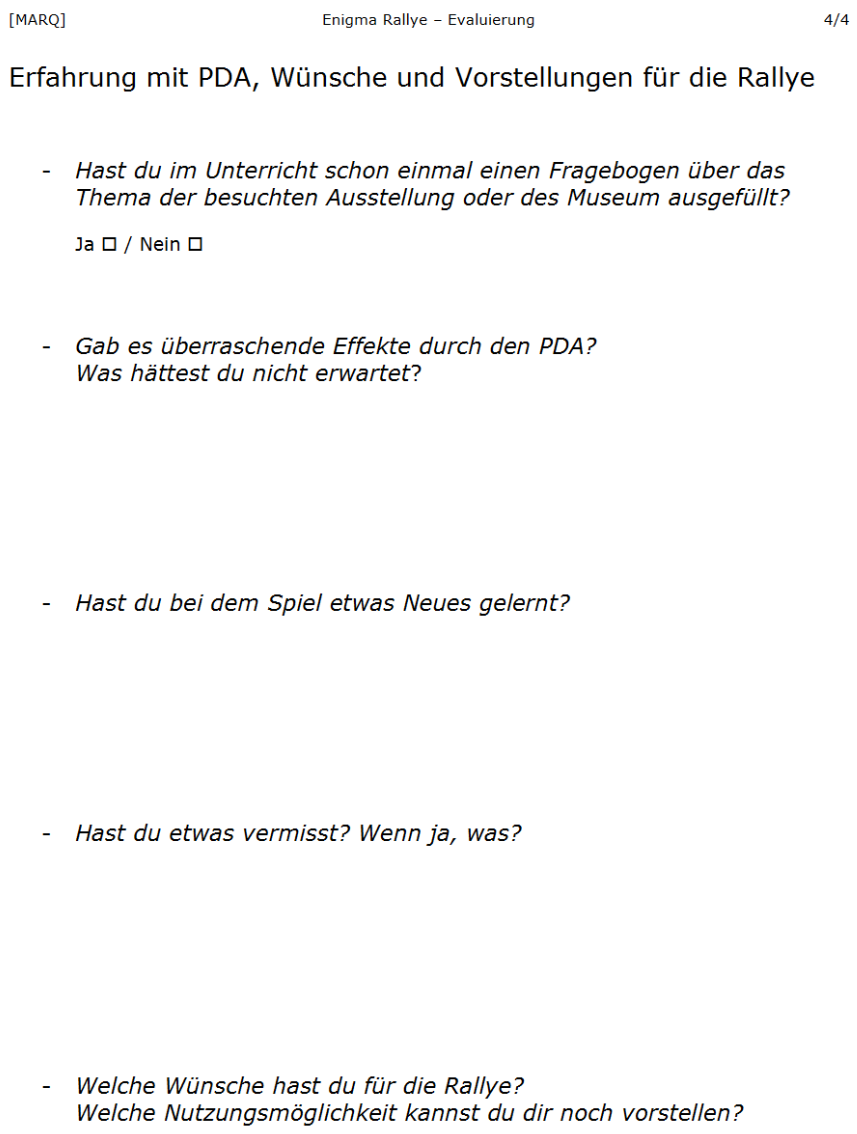
  ☐ Funkpeiler ☐ Morsen ☐ Enigma

  ____

- *Was würdest du verbessern?*

  ☐ Funkpeiler ☐ Morsen ☐ Enigma

  ____



[MARQ] Enigma Rallye – Evaluierung 4/4

Erfahrung mit PDA, Wünsche und Vorstellungen für die Rallye

- *Hast du im Unterricht schon einmal einen Fragebogen über das Thema der besuchten Ausstellung oder des Museum ausgefüllt?*

  Ja ☐ / Nein ☐

- *Gab es überraschende Effekte durch den PDA?*
  Was hättest du nicht erwartet?

- *Hast du bei dem Spiel etwas Neues gelernt?*

- *Hast du etwas vermisst? Wenn ja, was?*

- *Welche Wünsche hast du für die Rallye?*
  Welche Nutzungsmöglichkeit kannst du dir noch vorstellen?

9.3.4 Evaluation of Signpost 2007

The following page contains the questionnaire handed out to all users who fully played through the game built into the Signpost 2007 application. The evaluation was performed at the Microsoft embedded developers conference (MEDC) in Las Vegas, USA in 2007.

Signpost2007 Questionnaire

Please circle:

I am a novice / average / power user on PDAs or Smartphones.

On a scale of 1 to 7 please circle the number according to how much you agree or disagree with the following statements:

1) Signpost2007 was easy to use.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

2) Signpost2007 was more useful than a conventional map.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

3) Those black-and-white markers disturbed me.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

4) I'd like to see the other users' current positions on my device too.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

5) I think Signpost2007 can be used by novice PDA or Smartphone users.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

6) I was able to quickly access and understand the information (schedule and map) I searched for.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

7) I enjoyed using Signpost2007.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

8) Signpost2007 improved my location awareness.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

9) Signpost2007 should be used on other events too.

Strongly Disagree 1 2 3 4 5 6 7 Strongly Agree

Finally please give us some ideas for which other purposes this technology could be used:

Thanks for participating in this user study!

Chapter 10

Bibliography

[1] Amselem, D., A window on shared virtual environments. Presence, Vol. 4, No. 2, pp. 130-145, 1995

[2] Anabuki, M., Kakuta, H., Yamamoto, H., Tamura, H., Welbo: An Embodied Conversational Agent Living in Mixed Reality Space, CHI 2000, pp. 10-11, 2000, The Netherlands

[3] Azuma, R.: A Survey of Augmented Reality. In SIGGRAPH '95 Proceedings, Course Notes #9: Developing Advanced Virtual Reality Applications (1995), 1-38

[4] Bachmann, E., Duman I., Usta, U., McGhee R., Yun, X., Zyda, M., Orientation tracking for Humans and Robots Using Inertial Sensors. International Symposium on Computational Intelligence in Robotics & Automation (CIRA 99), pp. 187-194, 1999, USA

[5] Bajura, M., Fuchs, H., Ohbuchi, R., Merging Virtual Reality with the Real World: Seeing Ultrasound Imagery Within the Patient, In ACM SIGGRAPH Computer Graphics, Volume 26, Issue 2, pp. 203-210 1992, USA

[6] Balcisoy, S., Torre, R., Ponder, M., Fua, P., Thalmann, D., Augmented Reality for Real and Virtual Humans. IEEE Computer Graphics International, pp. 303-308, 2000

[7] Barakonyi, I., Fahmy, T., Schmalstieg, D., Kosina, K.: Collaborative work with volumetric data using augmented reality videoconferencing, In Proceedings of the 2003 International Symposium on Mixed and Augmented Reality (ISMAR 2003), pp. 333–334, 2003 Japan

[8] Barakonyi, I., Psik, T., Schmalstieg, D.: Agents That Talk And Hit Back: Animated Agents in Augmented Reality, IEEE and ACM International Symposium on Mixed and Augmented Reality 2004 (ISMAR'04), pp. 141-150, 2004, USA

[9] Barakonyi, I., Weilguny, M., Psik, T., Schmalstieg, D., MonkeyBridge: Autonomous Agents in Augmented Reality Games, ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE'05), pp. 172-175, 2005, Spain

[10] Barkhuus, L., Chalmers, M., Tennent, P., Hall, M., Bell, M., Sherwood, S., Brown B., Picking Pockets on the Lawn: The Development of Tactics and Strategies in a Mobile Game, UbiComp 2005, pp. 358-374, 2005, Japan

[11] Bauer, M., Bruegge, B., Klinker, G., MacWilliams, A., Reicher, T., Riß, S., Sandor, C., Wagner M., Design of a Component-Based Augmented Reality Framework, In Proceedings of The Second IEEE and ACM International Symposium on Augmented Reality (ISAR'2001), pp. 45-54, 2001, USA

[12] Benford, S., A distributed architecture for large collaborative virtual environments, IEEE Colloquium on Distributed Virtual Reality, pp. 9/1-9/7, 1993

[13] Benford, S., Fahlén, L., A Spatial Model of Interaction in Large Virtual Environments, 3rd European Conference on Computer Supported Cooperative Work (ECSCW '93), pp. 109-124, 1993, Italy

[14] Benford, S., Greenhalgh, C., Reynard, G., Brown, C., Koleva, B.: Understanding and Constructing Shared Spaces with Mixed-Reality Boundaries, In ACM Transactions on Computer-Human Interaction, Vol. 5, No. 3, pp. 185-223, 1998

[15] Bier, E.A., Stone, M.C., Pier, K., Buxton, W., DeRose, T.D.: Toolglass and magic lenses: The see-through interface, In: Proceedings of the 20st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 1993), pp. 73-80, 1993

[16] Billinghurst, M., Cheok, A.D., Prince, S., Kato, H.: Real world teleconferencing. In IEEE Computer Graphics and Applications Volume 22, Issue 6, pp. 11–13, 2002

[17] Billinghurst, M., Kato, H., Poupyrev, I.: The magicbook: a transitional ar interface. Computers & Graphics 25 (2001) 745–753

[18] Bimber, O., Fröhlich, B., Schmalstieg, D., and Encarnação, L.M. , The Virtual Showcase. IEEE Computer Graphics & Applications, vol. 21, no.6, pp. 48-55, 2001

[19] Bleser, G., Wuest, H., Stricker, D., Online camera pose estimation in partially known and dynamic scenes, In Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR'06), pp. 56-65, 2006, USA

[20] Brown, B., MacColl, I., Chalmers, M., Galani, A., Randell, C., Steed, A., Lessons from the lighthouse: collaboration in a shared mixed reality system, Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 577-584, 2003, USA

[21] Brown, D. G., Julier, S. J., Baillot, Y., Livingston, M. A., Rosenblum, L. J., Event-based data distribution for mobile augmented reality and virtual environments, Presence - Teleoperators and Virtual Environments, Vol. 13(2), pp. 211-221, 2004

[22] Bucolo, S., Billinghurst, M., Sickinger, D.: Mobile maze: a comparison of camera based mobile game human interfaces, Proceedings of the 7th international conference on Human computer interaction with mobile devices & services (MobileHCI'05) pp. 329-330, 2005, Austria

[23] DeVaul, R., Sung, M., Gips, J., Pentland, A., MIThril 2003: Applications and Architecture, Proceedings of the 7th IEEE International Symposium on Wearable Computers (ISWC), pp. 4-11, 2003

[24] Feiner, S., MacIntyre, B., Seligmann, D., Knowledge-based augmented reality, In Communications of the ACM, Volume 36, Issue 7, pp. 52-62, 1993

[25] Fiala, M., ARTag, a Fiducial Marker System Using Digital Techniques, In Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2, pp. 590-596, 2005

[26] Fitzmaurice, G. W. Situated Information Spaces and Spatially Aware Palmtop Computers, Communications of the ACM, Vol.36, Nr.7, pp 38-49, 1993

[27] Freeman, E., Hupfer, S., Arnold, K., JavaSpaces Principles, Patterns, and Practice, Pearson Education, ISBN 0201309556, 1999

[28] Gelernter, D., Generative communication in Linda, ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 9(1), pp. 80-110, 1985

[29] Greenhalgh C., Izadi S., Rodden T., Benford S.: The EQUIP Platform: Bringing Together Physical and Virtual Worlds. Technical Report, University of Nottingham, 2001. http://www.crg.cs.nott.ac.uk/~cmg/Equator/Downloads/docs/equip-platform.pdf

[30] Feiner, S., MacIntyre, B., and Höllerer, T., Wearing it out: First steps toward mobile augmented reality systems. In Proceedings of ISMR'99, pp. 363- 377, 1999, Japan

[31] Feiner, S., MacIntyre, B., Höllerer, T., Webster, A.: A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. Proceedings of the First International Symposium on Wearable Computers (ISWC), pp. 74-81, 1997, USA

[32] Föckler, P., Zeidler, T., Brombach, B., Bruns, E., Bimber, O., PhoneGuide: Museum Guidance Supported by On-Device Object Recognition on Mobile Phones, Proceedings of International Conference on Mobile and Ubiquitous Computing (MUM'05), pp. 3-10 2005, New Zealand

[33] Friedrich, W., ARVIKA - augmented reality for development, production and service. Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR), pp. 3-4, 2002, Germany

[34] Gabber, E., Wool, A. How to prove where you are: tracking the location of customer equipment, In Proceedings of the 5th Conference on Computer and Communications Security, pp. 142-149, 1998, USA

[35] Gausemeier, J., Fruend, J., Matysczok, C., Bruederlin, B., Beier, D., Development of a real time image based object recognition method for mobile AR-devices, Proceedings of the 2nd International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (Afrigraph 2003), pp. 133-1392003, Africa

[36] Geiger, C., Paelke, V., Reimann, C., Mobile Entertainment Computing, In Lecture Notes in Computer Science, Vol. 3105 / 2004, Springer Verlag, pp. 142-147, 2004

[37] Gelernter, D., Generative communication in Linda, ACM Transactions on Programming Languages and Systems (TOPLAS), Vol. 9(1), pp. 80-110, 1985

[38] Gelernter, D., Carriero, N., Coordination Languages and their Significance, In Communications of the ACM, Vol. 35, No. 2, 1992

[39] Grimm P., Haller M., Paelke V., Reinhold S., Reimann C., Zauner J., AMIRE - Authoring Mixed Reality, In Proceedings of The First IEEE International Augmented Reality Toolkit Workshop, 2002, Germany

[40] Güvem, S., Feiner, S., Authoring 3D hypermedia for wearable augmented and virtual reality. In Proceedings of the 7th International Symposium on Wearable Computers (ISWC), pp- 118-126, 2003, USA

[41] Güven, S., Feiner, S., Oda, O., Mobile Augmented Reality Interaction Techniques for Authoring Situated Media On-Site, In Proceedings of Mixed and Augmented Reality, pp. 235-236, 2006, USA

[42] Hakkarainen, M.; Woodward, C.: Symball-Camera Driven Table Tennis for Mobile Phones, Poster at ACM SIGCHI International Conference on Advances in Computer Entertainment Technology (ACE 2005), pp. 321-324, 2005, Spain

[43] Hartley, R., Zisserman, A., Multiple View Geometry in Computer Vision (2^{nd} Edition), Cambridge Pres,s 2003

[44] Henning M., Massively Multiplayer Middleware, ACM Queue, Vol. 1 (10), pp. 40-45, 2004

[45] Henrysson, A., Billinghurst, M., Ollila, M., Face to Face Collaborative AR on Mobile Phones. Proceedings International Symposium on Augmented and Mixed Reality (ISMAR'05), pp. 80-89, 2005, Austria

[46] Hesina, G., Schmalstieg, D., Fuhrmann, A., Purgathofer, W., Distributed open inventor: A practical approach to distributed 3D graphics." In Proceedings of the ACM symposium on Virtual reality software and technology (VRST'99), pp. 74–81. 1999, UK

[47] Heumer, G., Amor, H.B., Weber, M.; Jung, B., Grasp Recognition with Uncalibrated Data Gloves - A Comparison of Classification Methods, In Proceedings of Virtual Reality Conference (VR'07), pp. 19-26, 2007, USA

[48] Höllerer, T., Feiner, S., Pavlik, J., Situated Documentaries: Embedding Multimedia Presentations in the Real World, Proceedings of ISWC '99 (Third Int. Symp. on Wearable Computers), pp. 79-86, San Francisco, CA, 1999

[49] Höllerer, T., Feiner, S., Terauchi, T., Rashid, G., and Hallaway, D. Exploring MARS: Developing Indoor and Outdoor User Interfaces to a Mobile Augmented Reality System. Computers and Graphics, 23(6), Elsevier Publishers, Dec. 1999, pp. 779-785, 1999

[50] Holweg D., Jasnoch U., Kretschmer U.: GEIST - Outdoor Augmented Reality in an Urban Environment, Computer Graphics Topics, pp. 5-6, June 2002

[51] Hsiao, T.-Y., Yuan S.-M.. Practical middleware for massively multiplayer online games, IEEE Internet Computing, Vol. 9(5): pp. 47–54, 2005

[52] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., Klosowski, J. T., Chromium: a stream-processing framework for interactive rendering on clusters, ACM Transactions on Graphics, Volume 21, Issue 3, pp. 693-702, 2002

[53] IBM TSpaces, http://www.almaden.ibm.com/cs/TSpaces, 2000

[54] Ingram, D., Newman, J., Augmented Reality in a Wide Area Sentient Environment, Proceedings of the 2nd IEEE and ACM International Symposium on Augmented Reality (ISAR 2001), p. 77-86, 2001, USA

[55] Johanson, B., Fox, A., Hanrahan, P., Winograd, T., The Event Heap: A Coordination Infrastructure for Interactive Workspaces, IEEE Workshop on Mobile Computing Systems and Applications (WMCSA), pp. 83-93, 2002

[56] Kalkusch, M., Lidy, T., Knapp, M., Reitmayr, G., Kaufmann, H., Schmalstieg, D. Structured Visual Markers for Indoor Pathfinding, Proceedings of the First IEEE International Workshop on ARToolKit (ART02), 2002

[57] Kato, H., Billinghurst, M., Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System, In Proceedings of the 2nd International Workshop on Augmented Reality (IWAR 99), pp. 85-94, 1999, USA

[58] Klopfer, E., Perry, J., Squire, K., Jan, M., Steinkuehler, C., Mystery at the Museum - A Collaborative Game for Museum Education, CSCL (Computer Supported Cooperative Learning) 2005, pp. 316-320, 2005, Japan

[59] Lamberti, F., Sanna, A., A Streaming-Based Solution for Remote Visualization of 3D Graphics on Mobile Devices, In IEEE Transactions on Visualization and Computer Graphics, Vol. 13, No. 2, Marc/April 2007

[60] Ledermann, F., Schmalstieg, D., APRIL: A High-level Framework for Creating Augmented Reality Presentations, In Proceedings of Virtual Reality, pp. 187-194, 2005, Germany

[61] Lee, G.A., Nelles, C., Billinghurst, M., Kim, G.J., Immersive Authoring of Tangible Augmented Reality Applications, In Proceedings of the Third IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'04), pp. 172-181, 2004, Austria

[62] Lindt, I., Ohlenburg, J., Pankoke-Babatz, U., Ghellal, S., A report on the crossmedia game epidemic menace, Computers in Entertainment (CIE), Volume 5, Issue 1, Section on Pervasive gaming, ACM Press, 2007

[63] Liu, Y.; Wan, G., Techniques for Selecting and Manipulating Object in Virtual Environment Based on 3-DOF Trackers and Data Glove, In Proceedings of Conference on Artificial Reality and Telexistence (ICAT '06), pp. 662-665, 2006, China

[64] Long, S., Aust, D., Abowd, G. D., Atkeson, C., Cyberguide: Prototyping Context-Aware Mobile Applications. Proceedings of the CHI '96, pp. 293-294, 1996, USA

[65] MacIntyre B., Feiner S., A Distributed 3D Graphics Library. In Proceedings of International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH'98), Annual Conference Series, pp. 361-370, 1998

[66] MacIntyre, B. Gandy, M., Prototyping applications with DART, the designer's augmented reality toolkit. In Proceedings of STARS 2003, pp 19-22, 2003, Japan

[67] MacIntyre, B., Bolter, J.D., Moreno, E., Hannigan, B., Augmented Reality as a New Media Experience, International Symposium on Augmented Reality (ISAR 2001), p. 197, 2001, USA

[68] Makri, A., Arsenijevic, D., Weidenhausen, J., Eschler, P., Stricker, D., Machui, O., Fernandes, C., Maria, S., Voss, G., Ioannidis N., ULTRA: An Augmented Reality System for Handheld Platforms, Targeting Industrial Maintenance Applications, Proceedings of 11th International Conference on Virtual Systems and Multimedia (VSMM'05), 2005, Belgium

[69] Milgram P., F. Kishino: A taxonomy of mixed reality visual displays. EICE Transactions on Information Systems, Vol E77-D, No.12 December 1994

[70] Milgram, P., Zhai, S., Drascic, D., Grodski, J., Applications of augmented reality for human-robot communication, In Proceedings on Intelligent Robots and Systems (IROS '93), volume 3, pp. 1467-1472, 1993, Japan

[71] Mogilev, D., Kiyokawa, K., Billinghurst, M., Pair, J., AR Pad: an interface for face-to-face AR collaboration, Conference on Human Factors in Computing Systems (CHI'02) Extended abstracts on Human factors in computer systems, pp. 654-655, 2002, USA

[72] Möhring, M., Lessig, C., Bimber, O., Video See-Through AR on Consumer Cell Phones. Proceedings of International Symposium on Augmented and Mixed Reality (ISMAR'04), pp. 252-253, 2004, USA

[73] Murphy, A.L, Picco, G.P., Roman, G, LIME: A Middleware for Physical and Logical Mobility, International Conference on Distributed Computing Systems (ICDCS), pp. 524-541, 2001

[74] Naef, M., Lamboray, E., Staadt, O., Gross, M., The blue-c distributed scene graph, In IEEE Proceedings on Virtual Reality (VR'03), pp. 275-276, 2003, Switzerland

[75] Newman, J., Schall, G., Barakonyi, I., Schürzinger, A., Schmalstieg, D., Wide-Area Tracking Tools for Augmented Reality, In Proceedings of the 4th International Conference on Pervasive Computing, 2006, UK

[76] Piekarski, W., Thomas, B., Tinmith-Metro: New Outdoor Techniques for Creating City Models with an Augmented Reality Wearable Computer, In Proceedings of 5th International Symposium Wearable Computers (ISWC'01), pp. 31-38, 2001, Switzerland

[77] Piekarski, W., Thomas, B., Tinmith evo5 - An Architecture for Supporting Mobile Augmented Reality Environments. 2nd International Symposium on Augmented Reality (ISAR), pp. 177-178, 2001, USA

[78] Pilet, J., Lepetit, V., Fua, P., Augmenting Deformable Objects in Real-Time, In Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR 2005), pp. 134-137, 2005 Austria

[79] Pirchheim, C., Visual Programming of User Interfaces for Distributed Graphics Applications, Master Thesis at Graz University of Technology, http://studierstube.icg.tu-graz.ac.at/stb-thesis.php

[80] Regenbrecht, H.T., Specht, R., A Mobile Passive Augmented Reality Device - mPARD. Proceedings of ISAR, pp. 81-84, 2000, Germany

[81] Reitinger, B., Zach, C., Karner, K., Schmalstieg, D., Automated Model Acquisition using 3D Reconstruction for Urban Planning, Demo at the ISMAR 2006 symposium, 2006, USA

[82] Rekimoto, J., Matrix: A Realtime Object Identification and Registration Method for Augmented Reality. Proceedings of Asia Pacific Computer-Human Interaction (APCHI) 1998, pages 63-68, 1998, Japan

[83] Rekimoto, J., TransVision: A Hand-held Augmented Reality System for Collaborative Design, Proceedings of Virtual Systems and Multi-Media (VSMM '96), pp. 18-20, Gifu, Japan, 1996

[84] Rekimoto, J., Ayatsuka, Y., CyberCode: Designing Augmented Reality Environments with Visual Tags, Proceedings of DARE 2000, pp. 1-10, 2000, Denmark

[85] Rekimoto, J., Nagao, K. The World through the Computer: Computer Augmented Interaction with Real World Environments, User Interface Software and Technology (UIST '95), pp. 29-38, 1995

[86] Reitmayr, G., Drummond, T., Going out: Robust, Model-based Tracking for Outdoor Augmented Reality, In Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR'06), pp. 109-118, 2006, USA

[87] Reitmayr, G., Schmalstieg, D., Collaborative Augmented Reality for Outdoor Navigation and Information Browsing, Geowissenschaftliche Mitteilungen (Proc. 2nd Symposium on Location Based Services and TeleCartography), pp. 53-62, Vienna University of Technology, 2003

[88] Ribo, M., Lang, P., Ganster, H., Brandner, M., Stock, C., Pinz, A., Hybrid tracking for outdoor augmented reality applications, In Computer Graphics and Applications (CG&A) 2002 Vol.22, No.6, pp. 54-63, Nov. 2002

[89] Rohs, M., Gfeller, B., Using Camera-Equipped Mobile Phones for Interacting with Real-World Objects. Advances in Pervasive Computing, Austrian Computer Society (OCG), pp. 265-271, 2004, Austria

[90] Rohs, M.: Marker-Based Embodied Interaction for Handheld Augmented Reality Games, In Proceedings of the 3rd International Workshop on Pervasive Gaming Applications (PerGames) at PERVASIVE 2006, 2006, Ireland

[91] Schall, G., Newman, J., Schmalstieg, D., Rapid and Accurate Deployment of Fiducial Markers for Augmented Reality, In Proceedings of the 10th Computer Vistion Winter Workshop (CVWW'05), 2005, Austria

[92] Schmalstieg, D., Wagner, D., Experiences with Handheld Augmented Reality, The Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07), 2007, Japan

[93] Schmidt, D. C., Huston, S. D., C++ Network Programming: Systematic Reuse with ACE and Frameworks, Addison-Wesley Longman, 2003

[94] Schweighofer, G., Pinz, A., Robust Pose Estimation from a Planar Target, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 12, pp. 2024-2030, 2006

[95] Shibata, F., Mobile Computing Laboratory, Department of Computer Science, Ritsumeikan University, Japan, http://www.mclab.ics.ritsumei.ac.jp/research.html

[96] Siltanen, S., Hyväkkä, J., Implementing a natural user interface for camera phones using visual tags, In Proceedings of the 7th Australasian User interface conference - Volume 50, pp. 113-116, 2006, Australia

[97] Snowdon, D. N., West, A. J., AVIARY: Design Issues for Future Large, Scale Virtual Environments, In Presence, Teleoperators and Virtual Environments, 3(4), pp. 288-308, 1994

[98] Stapleton, C.B., Hughes, C.E., Moshell, J.M., MIXED FANTASY: Exhibition of Entertainment Research for Mixed Reality, CM International Symposium on Mixed and Augmented Reality (ISMAR 2003), pp. 354-355, 2003, Japan

[99] Strauss, P. S., Carley, R., An Object-Oriented 3D Graphics Toolkit, In Proceedings of the 19th annual conference on Computer graphics and interactive techniques (SIGGRAPH`92), pp 341-349, 1992, USA

[100] Sunblad, O., Sundblad, Y.,. OLGA - a Multimodal Interactive Information Assistant. Proceedings of the conference on Human Factors in Computing Systems (CHI 98), pp. 183-184, 1998, USA

[101] Sutherland, I., The Ultimate Display, In Proceedings of International Federation of Information Processing, Spartan Books, pp. 506-508, 1965

[102] Sutherland, I., A Head-Mounted Three Dimensional Display, In Proceedings of Fall Joint Computer Conference, pp. 757-764, 1968, USA

[103] Thomas, B. H., Piekarski, W., Glove Based User Interaction Techniques for Augmented Reality in an Outdoor Environment. Virtual Reality: Research, Development, and Applications, Vol. 6, No. 3, 2002

[104] Tramberend, H., Avocado: A distributed virtual reality framework. In Proceedings of IEEE Virtual Reality (VR'99), pp. 14–21, 1999, USA

[105] Vacchetti, L., Lepetit, V., Fua, P., Combining Edge and Texture Information for Real-Time Accurate 3D Camera Tracking. In Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR'04), pp. 48-57, 2004, USA

[106] Viega, J., Conway, M. J., Williams, G., Pausch, R.: 3D magic Lenses, In Proceedings of the 9th annual ACM symposium on User interface software and technology, pp. 51-58, 1996

[107] Vlahakis, V., Ioannidis, N., Karigiannis, J., Tsotros, M., Gounaris, M., Stricker, D., Gleue, T., Daehne, P., Almeida, L., Archeoguide: An Augmented Reality Guide for Archaeological Sites, IEEE Computer Graphics and Applications, V.22 N.5, pp. 52-60, 2002

[108] Wang, J. Zhai, S., Canny, J., Camera Phone Based Motion Sensing: Interaction Techniques, Applications and Performance Study, In ACM UIST 2006, 2006, pp. 101-110, Switzerland

[109] Wagner, D., Barakonyi I.: Augmented Reality Kanji Learning, In Proceedings of the 2nd IEEE/ACM Symposium on Mixed and Augmented Reality (ISMAR 2003), pp. 335-336, 2003, Japan

[110] Wagner, D., Billinghurst, M., Schmalstieg, D., How Real Should Virtual Characters Be?, Conference on Advances in Computer Entertainment Technology (ACE 2006), 2006, USA

[111] Wagner, D., Pintaric, T., Ledermann, F., Schmalstieg, D., Towards Massively Multi-User Augmented Reality on Handheld Devices, Proceedings of the 3rd International Conference on Pervasive Computing (PERVASIVE 2005), pp. 208-219, 2005, Germany

[112] Wagner, D., Schmalstieg, D., ARToolKit on the PocketPC Platform, The Second IEEE International Augmented Reality Toolkit Workshop, 2003, Japan

[113] Wagner, D., Schmalstieg, ARToolKitPlus for Pose Tracking on Mobile Devices, Proceedings of 12th Computer Vision Winter Workshop (CVWW'07), 2007, Austria

[114] Wagner, D., Schmalstieg, D., Muddleware for Prototyping Mixed Reality Multiuser Games, Proceedings of IEEE Virtual Reality 2007 (VR2007), 2007, USA

[115] Wagner, D., Schmalstieg, D. First Steps Towards Handheld Augmented Reality. Proceedings of the 7th International Conference on Wearable Computers (ISWC 2003), pp. 127-135, 2003, USA

[116] Wagner, D., Schmalstieg, D., Billinghurst, M., Handheld AR for Collaborative Edutainment, Proceedings of 16th International Conference on Artificial Reality and Telexistence (ICAT), 2006, China

[117] Zhang, X., Fronz, S., Navab, N., Visual Marker Detection and Decoding in AR Systems: A Comparative Study, Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR'02), pp. 97-108, 2002, Germany

[118] Zyda, M., Gossweiler, R., Morrison, J., Singhal, S., Macedonia, M., Panel: Networked Virtual Environments, In Proceedings of the Virtual Reality Annual International Symposium, VRAIS '95, pp. 230-231, 1995

[119] Zyda, M., Macedonia, M., Special Issue on Networked Virtual Environments. Presence: Teleoperators and Virtual Environments, 3(4)