AUGMENTED REALITY DEVICE FOR FIRST RESPONSE SCENARIOS

BY

ROBERT ANDRZEJ BOGUCKI
MS, Gdansk University of Technology, Poland, 2002

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science
in
Electrical Engineering

December, 2006

This thesis has been examined and approved.

Thesis Director, Dr. Richard A. Messner,
Associate Professor of Electrical and Computer Engineering

Dr. John R. LaCourse,
Chairman, Professor of Electrical and Computer Engineering

Dr. Michael J. Carter,
Associate Professor of Electrical and Computer Engineering

Date

DEDICATION

Dedicated to my loving family: parents Krysia and Andrzej and sister Kasia.

ACKNOWLEDGEMENTS

I wish to thank, first and foremost, my advisor Dr. Richard Messner for the mentoring, sound advice and encouragement he provided during my work on this project, and for welcoming me into the Synthetic Vision and Pattern Analysis research group. It was a great pleasure to be a part of this stimulating and focused research environment.

I am very grateful to Dr. Michael Carter and Dr. John R LaCourse for serving on my defense commitee, taking the time to review my thesis, and for their invaluable suggestions, advice and support.

I would like to thank Dr. Andrzej Rucinski and Dr. Barbara Rucinska for giving me the opportunity to study in the US and for helping me acclimate in a new environment.

I would also like to extend my thanks to the whole wonderful faculty of the Electrical and Computer Engineering Department, including Dr. Kent Chamberlin, Dr. Andrew Kun, Dr. Neda Pekaric-Ned and Mr. Frank Hludik, and others. Many thanks to always helpful and understanding Department's Administrative Assistant Kathy Reynolds, and Electronics Technician Adam Perkins.

Thanks to all my UNH colleagues, especially Jakub, Pavlo, Dragan, Alex and Zeljiko for their suggestions, readiness to help and many interesting conversations.

I am always deeply grateful to my family and close friends for always being there for me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

API – Application Programming Interface

AR – Augmented Reality

CCD – Charge Coupled Device

COTS – Commercial Off-The-Shelf

DRM – Dead Reckoning Module

EM – Electromagnetic

GIS – Geographical Information System

GPU – Graphical Processing Unit

GUI – Graphical User Interface

HCI – Human Computer Interaction

HMD – Head Mounted Display

HWD – Head Worn Display

HUD – Heads Up Display

NIC – Network Interface Card

PDA – Personal Digital Assistant

RFID – Radio Frequency Identification

SDK – Software Development Kit

UTM - Universal Transverse Mercator

WIMP - Windows, Icons, Menu, Pointing Device

ABSTRACT

AUGMENTED REALITY DEVICE
FOR FIRST RESPONSE SCENARIOS

by

Robert Andrzej Bogucki

University of New Hampshire, December, 2006

A prototype of a wearable computer system is proposed and implemented using commercial off-shelf components. The system is designed to allow the user to access location-specific information about an environment, and to provide capability for user tracking. Areas of applicability include primarily first response scenarios, with possible applications in maintenance or construction of buildings and other structures. Necessary preparation of the target environment prior to system's deployment is limited to noninvasive labeling using optical fiducial markers. The system relies on computational vision methods for registration of labels and user position. With the system the user has access to on-demand information relevant to a particular real-world location. Team collaboration is assisted by user tracking and real-time visualizations of team member positions within the environment. The user interface and display methods are inspired by Augmented Reality[1] (AR) techniques, incorporating a video-see-through Head Mounted Display (HMD) and fingerbending sensor glove.

---

[1] Augmented reality (AR) is a field of computer research which deals with the combination of real world and computer generated data. At present, most AR research is concerned with the use of live video imagery which is digitally processed and "augmented" by the addition of computer generated graphics. Advanced research includes the use of motion tracking data, fiducial marker recognition using machine vision, and the construction of controlled environments containing any number of sensors and actuators. (Source: Wikipedia)

CHAPTER I.

INTRODUCTION

1.1 Problem Statement

Making life-critical decisions based on limited information about a situation is undeniably a fact of life for First Response professionals, including, but not limited to, firefighters, HAZMAT teams, antiterrorist teams, law enforcement officers or rescue team members. The unpredictable, chaotic and complex nature of many emergencies, scarcity or overabundance of information coupled with the psychological effects of urgency and imminent danger pose difficult cognitive challenges to first responders. Responders are required to exhibit strong situation awareness, continuously process incoming information and possess an extensive knowledgebase. First response organizations attempt to address some of those needs by providing its members with training and drills within real, augmented (Harmless Hazards® system[2] by Harmless Hazards Training, LLC,) or virtual settings (Carnegie Mellon's Hazmat Hotzone[3], [Marinelli05]). Procedures and regulations for various scenarios and possible cases act as protective measures in situations with unpredictable outcomes. A clearly defined chain-of-command is usually employed to ensure responsible and strategically coordinated actions by groups of people in rapidly changing conditions. As necessary and efficient these measures may be, they are not without limitations. Simulation and training, although growing more sophisticated and realistic each year, still cannot mimic the complexity and randomness

[2] [http://www.harmlesshazards.com/]

[3] [http://www.etc.cmu.edu/projects/hazmat/videogallery.php]

of real-world situations. Procedures, while unarguably necessary, may be limited and inflexible, or simply ineffectual for a given situation. Several recent events illustrate the need for further improvements in the handling of emergencies. In the wake of the September 11th tragedy, an evaluative report of the New York City Fire Department's response [McKinsey02] was commissioned. The findings and recommendations contained therein point out areas for future improvement particularly relating to coordination and communications. Some of the problems experienced by the NYFD on Sept. 11th were inaccuracies in the tracking of the deployed units due to inadequate staging procedures (reporting to a predefined resource management area in vicinity of the incident). This resulted in over-deployment of units, insufficient orientation and misinformation of firefighters entering the towers, as it turned out many responders could not differentiate between WTC 1 and WTC 2 buildings.

1.1.1 Environment Annotation – Location Awareness

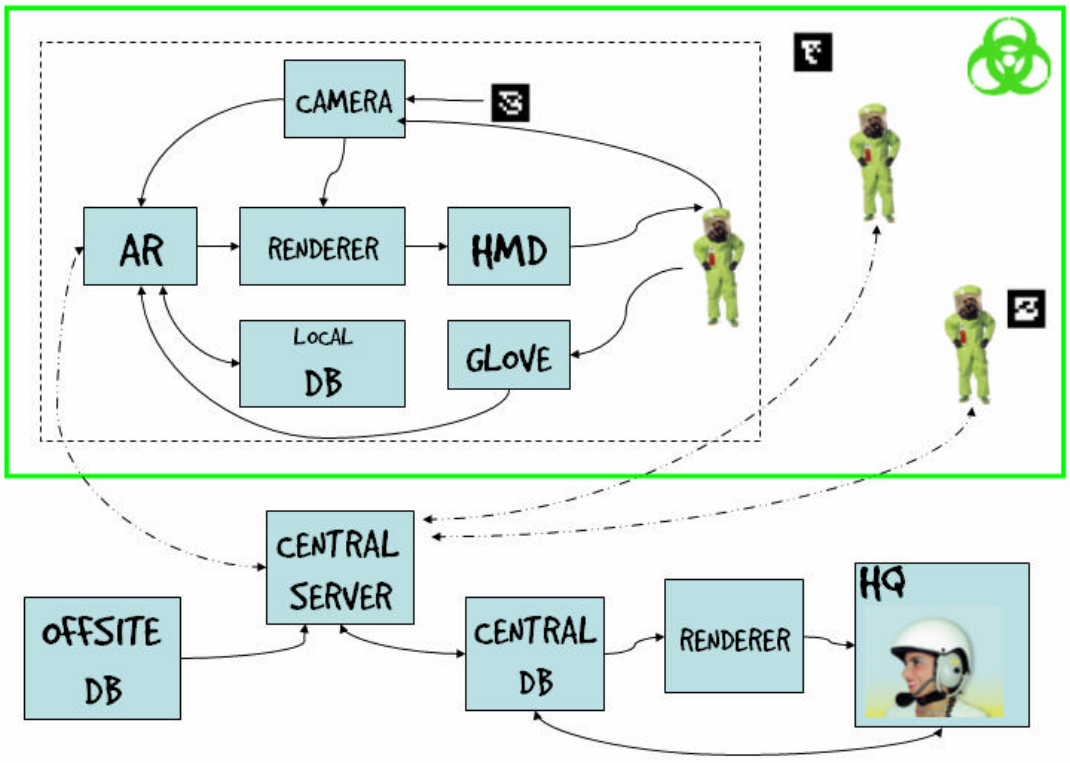
If a wartime analogy were to be used for a first response situation, the tactics and strategy employed in a given scenario are strongly determined by the environment. For example, a fire at a chemical processing facility needs to be handled in a different manner than one at a subway station full of people. Early availability of detailed knowledge regarding environmental specifics may facilitate the decision-making process and improve the outcome for a given response. In a majority of emergency situations, an external first response team is dispatched to the site of the emergency event with very short notice, leaving little or no time for close familiarization with the affected area in question. Information about the location available on-site may be limited or non-existent. Local personnel may be in distress or already evacuated and available reference materials such as blueprints and floor plans may be inaccurate, obsolete or incomplete. Recognizing the significance of the issue, fire departments attempt to provide their personnel with up to date location familiarization on a regular basis, however staying up-to-date in areas with ongoing development is a serious challenge.

To help in these situations it is proposed to construct 'self-annotated' environments which provide information about themselves on demand in an intuitive, filtered, location-specific manner. Such a system would consist of two parts, the informational content embedded within the environment, and the equipment necessary to access that content. Some of the key factors contributing to potential success or failure of introducing a novel system of this sort are the familiarity factor (social acceptance) and ease and cost of implementation (availability). The solution proposed herein is passive and noninvasive, requiring no substantial modifications on-site. It involves using optical markers (see Chapter 4, Section 4.4) , which can be printed and easily integrated into existing and required by law safety signage, acting as a natural extension of currently existing procedures. It can be argued that such 'low-profile' characteristics are more likely to facilitate widespread introduction and acceptance of the system, providing first response agencies with a rationale for equipping personnel with the necessary hardware equipment.

Having too much information available at hand at any given moment can be just as undesired as having too little information, since the cognitive costs of finding and accessing an item of interest grow with the number of available items. Therefore, in the proposed solution the user's current location will be utilized as a filter allowing for the reduction of the cognitive load by presenting only information relevant to that user based on their location. In order for the reader to better understand, an example system interaction scenario that illustrates the system's usage is now discussed.

In the proposed system, when a responder equipped with a wearable computing device looks at a cluster of fiducial labels placed nearby, she is provided with the descriptions of content available via each of the labels. The responder may then select one of the labels and inspect a piece of information. For example, an up-to-date chemical manifest of the storeroom located behind the door where the label was spotted could be accessed. Such content may also include floorplans, 3D models of the building, explanations for a device operation, valve labeling, or layout of the locations electrical or water supply systems. In Section 2.1 of Chapter 2, the Reader

will be presented with further example scenarios alongside descriptions of implemented functionality. The responsibility of prior preparation of the content for such an annotation system would lie locally with the building's/structure's administrators, allowing for frequent local updates and review for accuracy. A copy of the building's database would need to be stored off-site, available to the first responders in case of an emergency (see Fig. 1.1. below). Such a scheme would also require providing a standardized authoring environment alongside appropriate guidelines for content creation. It is worth noting that there is growing recognition for the need of better accessibility of such electronic resources, illustrated for example by a resolution passed by the Chicago City Council requiring buildings more than 80 feet tall to submit electronic floor plans to the Office of Emergency Management to help rescue workers navigate inside. Some suggestions and remarks toward a large scale deployment of the proposed scheme can be found in Section 2.2 of Chapter 2 in an attempt to outline future work directions.



DB – database server, **HMD** – Head Mounted Display, **HQ** - Headquarters

Figure 1.1. Overview of proposed system's deployment.

An off-site database server holds a copy of the location specific information, which is retrieved when an emergency situation occurs. Dotted lines represent wireless communications. Green boundary symbolizes the area affected by the emergency (for example a contaminated or burning building).

The functionality described above would be provided by means of *markers*, or *tags* previously placed throughout the environment, providing the necessary 'hooks' between the realm of the physical world and the informational space of the reference database. 'Tagging' of the environment by various means, including RFID (Radio Frequency Identification) or machine vision technology, is a concept closely related to HCI (Human-Computer Interaction) research related to exploration of alternative computing paradigms, such as augmented reality, or ubiquitous[4] (for an example see [Lampe2004]) and tangible[5] computing. In general, such approaches can be often characterized by a blending of the boundaries between the computer interface and the users' physical environment.

In the system proposed here, optical planar fiducial markers[6] are utilized. Easily printed on paper, they can be quickly produced and placed by the building's administration at low cost, which fits well with the vision of a system which has to be relatively inexpensive and easy to deploy.

---

[4] Ubiquitous computing (ubicomp, or sometimes ubiqcomp) integrates computation into the environment, rather than having computers which are distinct objects. Another term for ubiquitous computing is pervasive computing. Some simple examples of this type of behavior include GPS-equipped automobiles that give interactive driving directions and RFID store checkout systems. (Wikipedia)

[5] Tangible computing, refers to computing systems that use physical artifacts as the representation and the control of digital information. In other words, tangible computing is about having devices that represent some information by ways of their color, behaviour, sound or other properties. (Wikipedia)

[6] In applications of augmented reality or virtual reality, fiducials are often manually applied to objects in the scenery to recognize these objects in images of the scenery. For example, to track some object, a light emitting diode can be applied to it. With the knowledge of the color of the emitted light, the object can easily be identified in the picture. (Wikipedia)

Tagging of the environment with fiducial markers is relatively nonintrusive and inexpensive. The simplicity of this approach also translates to a certain degree of robustness, although some restrictions apply. For example, one of the drawbacks of using visual fiducial markers is the requirement of line of sight viewing in order to recognize a marker. This might be difficult in environments where smoke or other particulates interfere with a vision based system ([CLS86],[Zinn77]).

1.1.2 Responder Position Tracking

The second main benefit of the system proposed here is responder position registration. When the environment tagging technique described above is coupled with wireless networking technology, information about tags viewed by a responder can be reported back to the base of operations as well as to other responders in a team.

Usefulness of such technology becomes more obvious in the light of a user needs study in [Wilson05] which pointed out that that;

> *(...) firefighters must often make a best guess of where the fire started and where it is traveling in a building by determining which alarm activated first (if possible), looking for smoke issuing from windows, and by word of mouth from building occupants. From this information, they then guess where the safest and most effective place to enter the building is. Further adding to the guesswork, they do not have maps with them when they are in a building. This is because paper floor plans are impossible to carry and read while trying to fight a fire and rescue victims. Instead, firefighters may navigate by unreeling a rope as they go, tying knots to mark important locations, or marking doors with large crayons. When the smoke becomes thick, the "left-hand rule" is used: they drag their left had along the wall so as not to become disoriented. Thermal cameras, when available, are also used to navigate smoke. These methods do*

not always work, as there have been incidents in which firefighters have become lost and suffocated to death in thick smoke.

Firefighters are often equipped with a special motion-sensing Personal Alert Safety System (PASS) device worn at the belt or as part of their SCBA (Self Contained Breathing Apparatus). The purpose of such a device is to activate an audible alarm if the subject remains motionless for a predetermined time interval, since this may indicate that they need rescuing. Coupled with the system proposed here, such an alert broadcast over a wireless communications channel can be accompanied by information about the firefighter's 'last seen' position based on location of the most recently viewed label. Team member tracking is also possible throughout the duration of the whole operation. Firefighters' positions may be incorporated into the content viewed within the proposed system. For example a 3D building map may show the approximate position of the user and other team members within the building. This information may be also relayed to the center of operations, allowing the Dispatch, the Chief Officers and the Field Communication Unit to track individual units in a robust and scalable manner. Bookkeeping for larger operations is also facilitated, since locations of units and personnel can be recorded and re-examined, making it easier to identify missing or deceased personnel.

1.1.2.1 Registration Techniques for Location Awareness and Position Tracking.

In the prototype implementation of the system proposed here, a specialized software library for 2D planar fiducial marker detection was utilized (ARTag, [Fiala04]). A more in-depth discussion of the library can be found in Section 4.4 of Chapter 4. In the prototype, an integrated head-mounted camera relays video to an image recognition algorithm, which allows the system to detect fiducials placed within user's field of view. Each unique fiducial has an ID associated with it which may be linked to information about the fiducials real world location, and/or to other types of information.

This method of providing location awareness and responder movement tracking has some limitations. The resolution and reliability of tracking is largely dependent on the density of marker placement throughout the environment, and the responders' consistency in allowing the system to register markers passed along on the way, as line-of-sight with a fiducial marker is required for registration. Although fiducial labels can be made damage-resistant to a certain degree, obviously in the case of a raging fire or structural collapse such a system would not be of much use. Since the system is computational vision-based, visibility of labels is critical for the system to function. Low or no light conditions can be at least partially circumvented by utilizing photo-luminescent paint in the production of the labels. New generations of photo-luminescent pigments can emit a strong glow for eight hours or more after 'blackout', and are already widely used for safety signage applications. However, the system proposed here would still not be adequate for scenarios with heavy smoke coverage. Strategic placement of markers to keep them below the smoke level, as well as predictability and density of marker placement may be considered as a partial remedy.

There are several alternatives when it comes to providing location awareness functionality, based on other tracking and sensing methods researched elsewhere or commercially available; some of them are listed below. A comprehensive overview of existing indoor navigation methods for first responders can be found in [Miller06], a feasibility study conducted by NIST (the computational vision approach taken here is not among the methods listed by the authors).

- Detection of different types of tags in the vicinity, for example RFID tags. RFID technology has a definite advantage over vision based methods in that it doesn’t require tag visibility. Some potential drawbacks include privacy concerns, as contents of an RFID tagged building could be scanned from a distance by terrorists. Also, in an Augmented Reality application (see discussion later in this chapter) the usefulness of RFID tags is limited, as precise tag position registration is very challenging. Supplementing the proposed system with RFID tags

alongside the visible labels seems to be an interesting possibility, potentially allowing for increased robustness in limited visibility conditions, as detection of an RFID tag in close proximity would still be possible even in presence of smoke, dust, or flames [Miller06].

- Tracking via pre-installed in-building sensing networks or beacons (infrared, RF, ultrasound etc). Some examples of such approaches are SureStep (Integrated Sensors, Inc) and MoteTrack (described in [Lorincz05]). While robust under a wide range of operating conditions, such methods are significantly more costly and invasive, which may be seen as prohibitive to very wide scale deployment.
- Position tracking independent of the environment infrastructure (no markers needed), including:
    - GPS (satellite) based solutions. These are limited to outdoor use, largely restricting their usefulness for first response applications.
    - RF methods (triangulation, external transmitter). As noted in [Miller06], indoor environments typical of first response situations may not be 'friendly' to RF propagation.
    - other methods of tracking (accelerometers, magnetometers, pedometers). Dead-reckoning by means of accelerometers suffers from problems related to significant sensor readout drift over time. Magnetometer- based solutions are sensitive to magnetic field disturbances (metallic objects, EM radiation from electronic equipment) and require taking into consideration natural variation in the direction of Earth's magnetic field [Miller06], while pedometer based solutions require calibration for users average stride length, and are not very reliable over longer periods of time [Randell03].
    - Computational vision based non-fiducial (environment feature extraction, visual flow field analysis).

- Hybrid approaches attempting to alleviate the weaknesses of individual techniques by fusing data provided by different types of sensors, such as in The Human Odometer described in [Wong05] or the system described in [Miller06] (DRM, magnetometer and RFID waypoint corrections).

- Active electronic devices with displays, providing access to information and the building's systems, *ala* Mark Weiser's "ubicomp" [Weiser91].

1.1.2.2 Human Factors Design Considerations

As computer technology becomes increasingly pervasive in most aspects of human activity, new human-computer interface paradigms emerge, encompassing novel display techniques and devices, new interaction methods and I/O devices. Technologies such as Personal Digital Assistant devices (PDA's), mobile and wearable computers, augmented reality, tangible interfaces, ubiquitous computing challenge the way we think about computers and how we use them. Existing successful examples such as Heads Up Displays (HUD) in fighter jets, automobile onboard satellite navigation systems illustrate the potential of novel technological approaches.

A key issue in the design of a successful system is to give thorough consideration for human factors (HCI) issues related to the design of the user interface and interaction. Design of a highly specialized application such as the proposed first response system, intended for use in life-threatening situations requiring heightened situational awareness, poses a completely different set of questions than the design of an office productivity tool application. Beside software and hardware development challenges, the scope of the problem includes ergonomics as well as psychological issues of perception and cognition. The following paragraphs introduce some of the related HCI issues and discuss available design choices in that regard.

The question of what would be the most fitting computing metaphor for a device assisting first responders, and what are the main requirements and constraints for such an application had to be considered from the very beginning of the design process. The previous discussions within

this chapter were concerned mostly with the methods of environment 'annotation', the informational content part of the system and its ties with the physical space. In this section the focus will shift to the 'reader' device portion of the system, the apparatus used by the first responder to extract and inspect the information from the environment. The design requirements and assumptions listed below pertain to this mobile computing device part of the system.

- The device must not restrict user's mobility;
    - Provisions for untethered operations. The device should be battery powered and communications should be based on a wireless transmission channel.
    - The interface should allow for operation while performing typical first-response actions. It should be possible to use while carrying large or heavy objects, for example an unconscious person being rescued, or heavy tools such as axes or compressed air bottles.
- The device should be lightweight;

  Burdening the first responder with too much weight (besides the already substantial weight of existing equipment such as the breathing apparatus) would negatively affect their endurance and agility.
- Visibility of the display in a wide range of lighting conditions;

  The device's display should be readable in the widest possible range of lighting conditions possible. A rescue action may take place on a sun-drenched snow-covered mountain slope as well as inside a burnt out oil rig with the only illumination being that carried by the responders.
- Convenient placement of the display;

  In some cases the responders' field and quality of vision may already be restricted by a breathing mask, SCUBA device or HAZMAT suit. Ways to seamlessly integrate the display within such equipment should be investigated if possible.

- Intuitive interface;

  The required information should be accessible in an intuitive and natural manner, through simple actions on behalf of the user. Information presented shouldn't require time-consuming interpretation (the cognitive gap should be bridged), and must not significantly interfere with the processing of the information from the outside environment.

Considering the above requirements, the interaction scheme and display technique seem to be strongly determined by the envisioned pattern of information access described previously. Since the information accessed with the system is tightly related with the external environment, it can be argued that the Augmented Reality metaphor provides the best match. The creation of ties between real environments and 'virtual' visualizations of information lies at the heart of Augmented and Mixed Reality[7] paradigms.

Augmented Reality (AR) is an important subclass of Mixed Reality, one in which "the display of an otherwise real environment is augmented by means of virtual (computer graphic) objects" [Azuma2001]. As defined in the [Azuma97] survey paper, an AR system is a system that combines real and virtual objects in a real environment, runs interactively in real-time, and registers real and virtual objects with each other. Although beginnings of AR trace back to 1960s, most of the work in this field was done within the last 15 years (see updated survey [Azuma2001]), when several technological barriers were being gradually overcome. As of today, it is becoming more and more plausible to construct inexpensive and functional augmented reality

[7] The term Mixed Reality encompasses a broad range of user interfaces based on an interplay between the real environment and a virtual one. In Milgram and Kishino [Milgram94] a reality-virtuality continuum is proposed to describe the amount of contribution from each of the real and the 'virtual' worlds within a Mixed Reality. Within this continuum, Mixed Reality devices may be located anywhere between the extremes of purely computer-generated VR simulations and the external, physical reality.

devices using commercial off-the-shelf components, then coupling them with freely available software components. Some strong technological enablers for AR are the game entertainment industry-driven explosive growth of the graphics processor unit industry, particularly the recent introduction of powerful graphical accelerator cards in mobile computers. Head Mounted Displays using new types of LCD displays providing resolutions, color depth and contrast almost comparable to desktop LCD monitors are now available off the shelf and relatively affordably priced (under a thousand dollars). Wireless networking technology is fast becoming ubiquitous, and may be combined with powerful mobile computers capable of performing tasks such as computational vision analysis of live video, high-performance 3D modeling, and handling data from many input devices concurrently.

1.1.2.4 Augmented Reality in Maintenance Tasks.

The AR family of technologies brings a promise of assisting humans with a variety of tasks in the near future. In their work concerned with AR in manufacturing and maintenance authors Neumann and Majoros argue that Augmented Reality presents us with "a media form (...) complementary to human cognitive processes" [Neumann98]. The tasks considered in that study are related to aircraft technical maintenance; however there are many underlying similarities between scenarios discussed there and those located within a first response context.

While providing navigation and location-awareness within a building can be seen as a major use for a first response device, emergencies often require interaction with an environment's technical infrastructure. This calls for inclusion of maintenance-oriented information within the proposed system. Shutting down malfunctioning equipment damaged in the emergency, locating the layout of pipes, identifying the purpose of valves and switches or operating a complicated device would be greatly facilitated if this infrastructure is 'tagged' with relevant information. When considering the usefulness of such annotations, infrastructure-heavy environments such as a chemical or power plants, oil-drilling platforms, marine vessels are the ones which come to

mind first and foremost. However, it can be argued that the complexity of most modern office buildings and high-rises also warrants uses for such a system.

An interesting fact about maintenance tasks is brought up in [Neumann98] regarding aircraft maintenance workload characteristics. It turns out that approximately *"45 percent of a technicians shift is spent on finding and reading procedural and related information"*. In light of this it seems obvious that AR has potentially much to offer with regard to such tasks. Not only can the search for information (traditionally associated with hefty paper manuals) be reduced by triggering the display of relevant information simply by the action of looking at a workpiece (low information access cost), the cognitive overhead is further reduced by having the information reside in the same perceived location as the workpiece, decreasing the need for attention switching. Information can be disclosed gradually using interactive methods, as illustrated by a sequence of automated instructions for an airplane subsystem in [Neumann98].

1.1.2.5 Device Format

Augmented Reality is a broad design category encompassing devices of many forms and shapes. What type of a system would be most suitable for the envisioned first response application? The untethered mobility requirement suggests either a portable PDA-based device integrating the computing device and the display, or a wearable computer coupled to some sort of head-worn display device.

An attempt to integrate a miniature display device into a breathing apparatus mask, making it possible to view thumbnail-sized floorplans, is described in [Wilson05]. The paper also reports the results of user needs studies performed with the cooperation of 50 firefighters and fire chiefs from the Chicago and Berkeley Fire Departments.

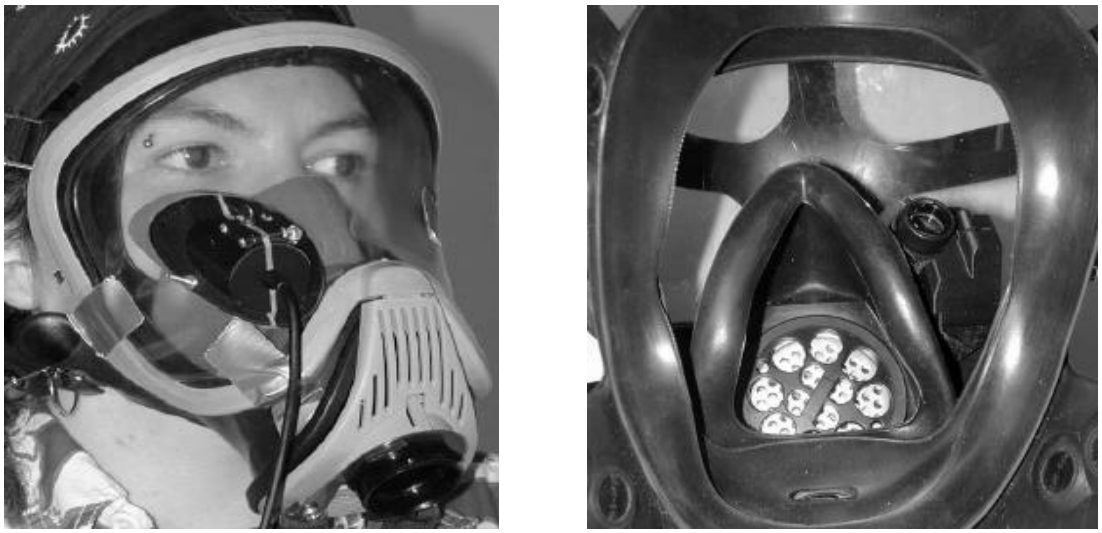


Figure 1.2 Prototype Head Mounted Display embedded within a Drager firefighter's mask. Left: External view, right: inside of mask. The coin-sized visor allows viewing downscaled floorplans. Source: [Wilson05]

The authors of the paper conclude that: "A large area of future work is the graphical user interface (GUI) design. The HMD will have a GUI that is simple and effective. It will show firefighters only what they need to know, in a clear and concise manner. There will also be a control system allowing some manual operations by the user. This control system will ideally be simple and hands-free.".

The prototype device described in this work attempts to address the above issues. An I-glasses head mounted display (see Figure 1.3) is used as the display device in the prototype. When coupled with a Head Mounted Camera, this type of display device opens up rich possibilities for user input which do not involve user's hands, for example performing selection tasks utilizing only head movements when a fiducial marker is in view, as described in Section 2.1 of Chapter 2. Helmet mounting systems are available for commercially available HMD displays (including the i-glasses HMD). This makes it possible to experiment with using such displays for applications involving first responders wearing safety helmets. Examples of existing systems currently in use by the military include the Nomad display based situation-awareness system or traditional nightvision displays. While the ergonomics of low-cost HMDs from the

public sector are still an issue for serious first response applications (the weight of the I-glasses device is 7 ounces), the prototype setup allows for experimenting with the proposed interaction scheme and verifying its usefulness. A more in-depth discussion of HMD characteristics and current state of the technology can be found in Section 3.2 of Chapter 3.

Since not all of the envisioned user interface functionality could be easily implemented using head movements alone, an additional input device is required. An approach based on fingerbending sensors was chosen which is described in section 3.4 of Chapter 3. The implementation is based on a modified P5 glove controller device. While not "hands-free" in the strict meaning of the word, the glove-based approach offers the responder more manual freedom than the majority of typical input devices, allowing them to carry heavy objects or perform other manual tasks while concurrently using the proposed interface.

Figure 1.3. An I-glasses HMD with a helmet mounting system.
(Source: manufacturer's website).

Figure 1.4. P5 glove controller device. Infrared sensing tower shown top right.
(Source: manufacturer's website).

CHAPTER II

SYSTEM FUNCTIONALITY

2.1 Interaction and Interface Design

2.1.1 Overview

In creating an effective Graphical User Interface for the first-response device, effectiveness, intuitiveness and cost of knowledge should be considered. The implemented interaction technique described in this section is very direct, being based on a visual target acquisition model which is natural to humans.

The functionality of the implemented prototype will be discussed using a 'usage scenario' framework. Several scenarios are presented in the following section, Such a format allows the presentation of the specifics of the sequence of interactions alongside the rationale for the design decisions behind them, as well as the intended methods of system deployment. Screenshots of implemented functionality are presented where appropriate, however for more in-depth implementation details the Reader will be referred to Chapters 3 and 4.

2.1.2 Scenarios, Storyboards Illustrating Interaction

2.1.2.1 SCENARIO 1 (Labels, Active Area, Selection, Browsing Content, 3D Model)

The following interaction scenario illustrates the following functionalities and interaction methods:

- accessing resources relevant to a physical location based on 2D fiducial markers
- 'details on demand' interaction based on an active area in the center of the AR display

- More specifically, the user is:
- selecting from a number of available markers (each giving access to different content).
- accessing a 3D mesh of a building enhanced with a “You Are Here” indicator
- using head movement (position of an active marker within users' view) for changing the virtual viewpoint when viewing a 3D model

Interaction methods for browsing other types of content are described in Section 2.1.2.1.2.

The user, equipped with the AR device, enters a location (shown on Figure 2.1, plan view). Noticing several markers placed throughout the location, the user flips down the head mounted display to access location based information. The fiducial markers viewed through the AR video-see-through display are augmented with brief descriptions of content that is associated with them, as shown in Figure 2. Additionally, an ‘active’ area is signified by the circular area in the middle of display. This area is effectively a ‘fovea’ or a ‘looking glass’ which allows the user to ‘zoom in’ on items of interest. Of course, calling the active area a ‘fovea’ is only a metaphor, since a user may in fact foveate on any point of the AR display at any given moment. Since the movement of the display and camera is coupled with the movement of the wearer’s head, it is through the movement of the whole head (or body) that the user brings an augmented marker into the center of the screen. The screen distance of a marker from this active area has a subtle scaling effect on the size of font used for the text overlay describing the marker, providing a cue to the user that the center of screen is the ‘closer examination’ area. In this case, two markers were found in the user’s field of view, with an abbreviated description of the type of content provided.

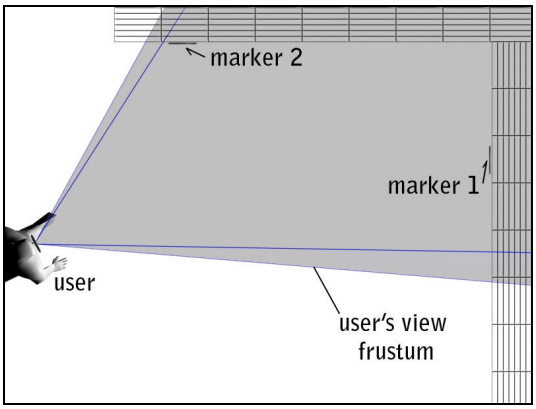


Figure 2.1. Topside view of the location, showing the user's field of view and the placement of fiducials on the walls.

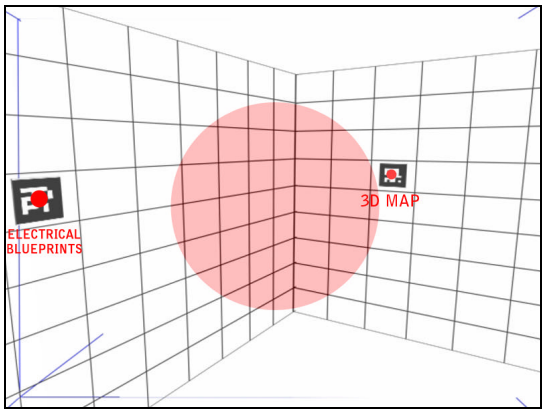


Figure 2.2. User's Augmented Reality view of the scene, showing two markers and the active area.

NOTE: In the figures accompanying this scenario, for clear distinction between the camera video and the augmentations, the real-world imagery is shown in shades of grey, while shades of red are used to illustrate synthetic AR overlays.

Let us assume that the user is more interested in perusing the 3D map than the electrical blueprints. In this case they would approach the relevant marker (see Figures 2.3 and 2.4). The scale of the viewed marker (corresponding to real-world distance and/or physical size of the marker) affects the marker label size and visibility as well. In this manner the user's physical movement through space is also having an effect on the relative visibility of augmentations, providing a simple means of filtering, since a large number of equally prominent labels could crowd the display and considerably increase cognitive overload.

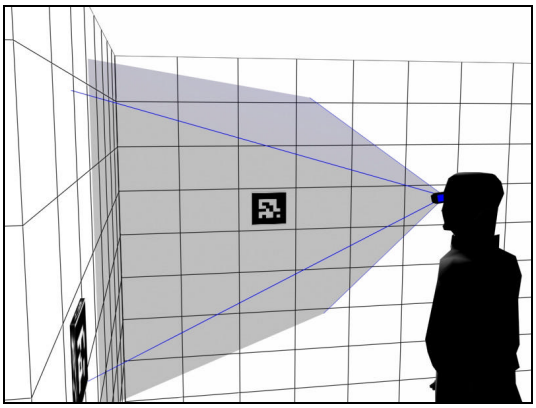


Figure 2.3. The AR view, as observed by the user (External view).

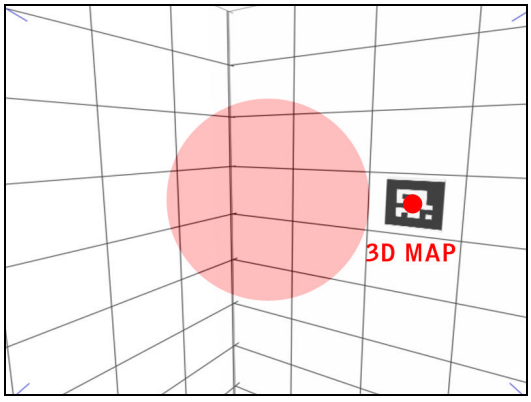


Figure 2.4. The AR view, as observed by the user (User's view).

Since the user is interested in the content associated with this marker, they center their field of view on it, bringing it into the active area (see Figures 2.5 and 2.6).

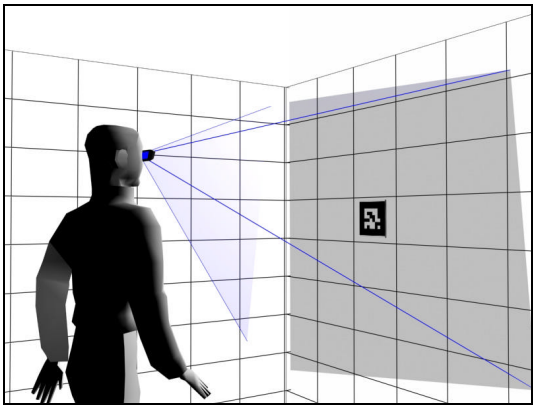


Figure 2.5. User 'foveates' on the marker of interest (external view)

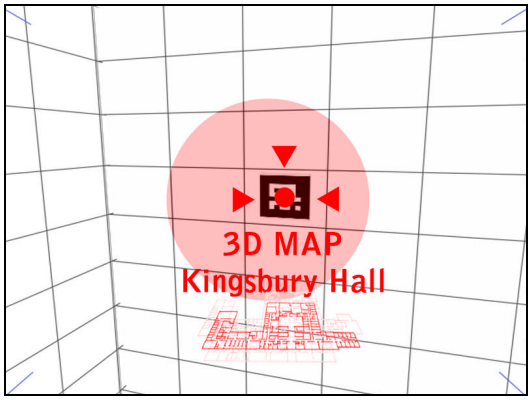


Figure 2.6. User 'foveates' on the marker of interest (User's view).

For a marker positioned within the central active area, the label is enlarged considerably, and since the marker is now 'active' (selectable), it is enhanced with additional information, such as specific details pertaining to the content available. In the example shown in Figure 2.6, the label now also indicates the name of the building, and contains an iconified representation of the content.

Within the active area, the scaling effect of the marker's movement toward or away from the center of the display is further emphasized. The maximum label size corresponds to the exact center of the active area with size sharply dropping off with the increase in distance between the markers center and the display center point. If there are more markers within the central area, it is still possible to avoid overcrowding and overlapping of labels. Additionally, if the label legibility is low for some reason, precise positioning of the marker over the dead center allows the user to expand the label to a clearly readable size. The marker that is closest to the center of the active

area is the ‘active’ marker, which is signified by the expanded label and red color used to render the label’s text.

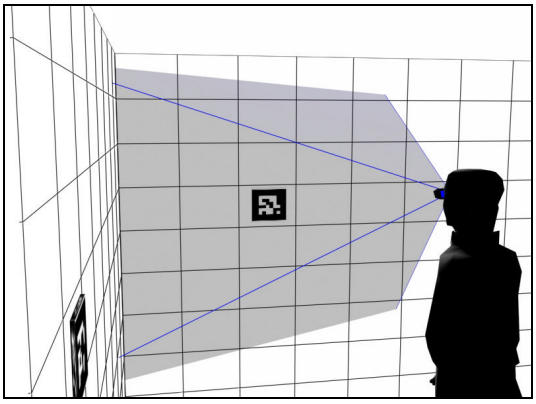


Figure 2.7. Activating a selected marker (External view).

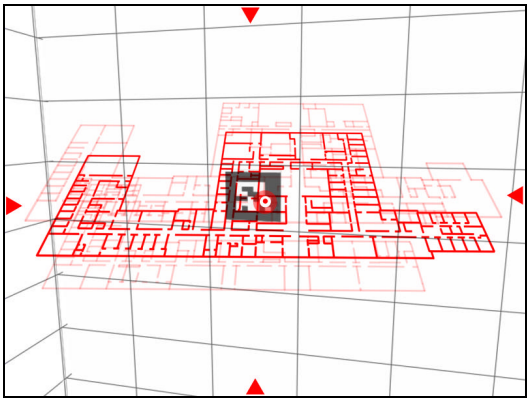


Figure 2.8. Viewing a wireframe mesh model/map of the building. Default position – center of display corresponds to user’s physical location within the building.

In the next step (Figure 2.7), the user activates the content associated with the marker using a palm of hand gesture (like the rapid flicking of the index finger, detected by the fingerbending sensors embedded into a glove).

The content is displayed with a default viewpoint (Fig. 2.8). For a 3D map of a location, this would mean that the "You Are Here" position is in the center of view immediately after the marker has been activated. The orientation of the mesh matches the users head position when the marker is in front of them.

The original marker, still visible in the center of the screen in the real world video imagery in the background, becomes a ‘controller widget’. Its position is now indicated using the edge of the screen ‘crosshairs’ augmentations instead of a direct ‘in place’ augmentation, in order to reduce visual interference with the visual content now visible in center. The active area is hidden as well, for the whole duration of content browsing.

The size and placement of a fiducial marker within the physical space is known to the system, so from the perceived marker sizes the users position may be interpolated. In the example, the fact that the map was brought up by activating a particular marker is enough information to allow the system to provide the user with a "You Are Here" indicator, as well as orienting the map properly with regard to the users' position while viewing the marker in front of them. In a multi-user setting, other team member positions based on markers recently acquired by their head mounted cameras (or by other means of positioning as well) may be displayed within the map as well.

At this stage, the user may inspect the mesh from a predefined range of possible viewing angles, by moving their head and relocating the original marker within their field of view. The x and y Cartesian position of the control marker's center is mapped to the virtual camera position used for rendering the mesh, or to the mesh object yaw, pitch, roll rotation coordinates. For example, if the marker is relocated to the left when the user looks to the right (while still keeping the marker within the field of view) the mesh yaw is modified, so that the left wall of the building is now closer to the user (see Figures 2.9 and 2.10).

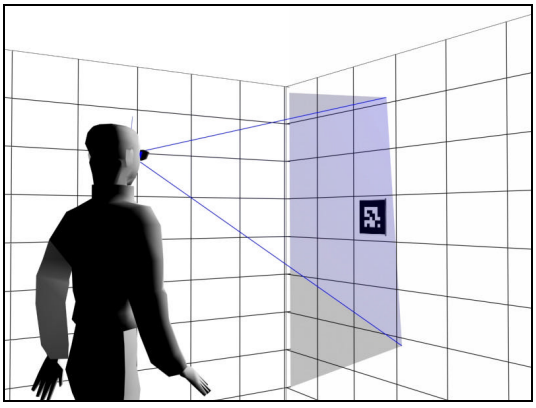


Figure 2.9. User moves their head to the left. Active (control) marker is now in the right-hand side of their field of view (External view).

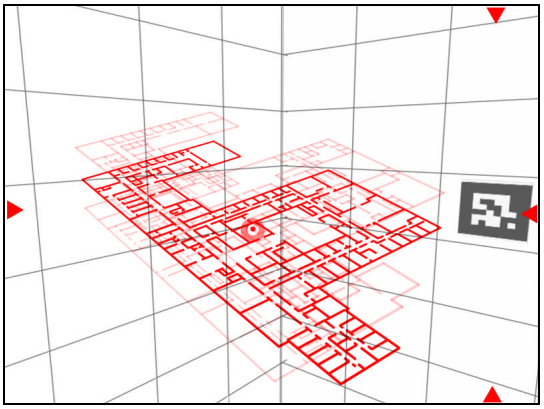


Figure 2.10. ..the 3D augmentation reacts by rotating accordingly (User's view).

Such a mapping is intuitive as the user is now looking at the right side of the object as if the object was placed to the left of user, therefore the user should be looking leftward.

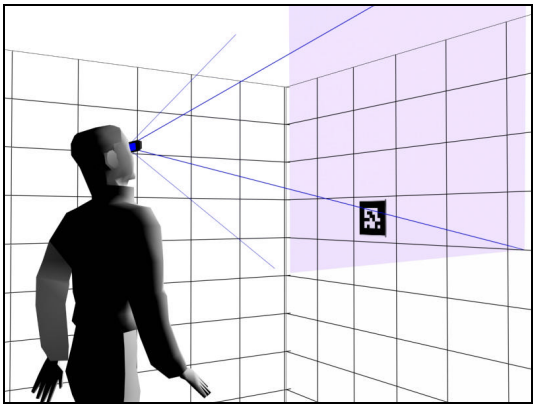


Figure 2.11. User looking upward (External view).

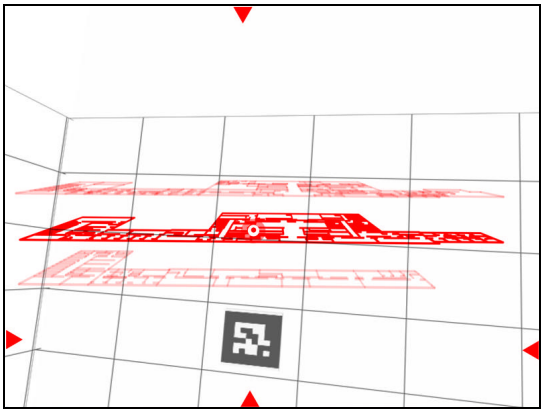


Figure 2.12. Mesh pitch modified accordingly (User's view).

When the user looks upward (see Figures 2.11 and 2.12), the marker moves down in their field of view, and the mesh pitch is modified so that the building is now viewed from a lower vantage point, allowing for instance to determine which teammates are on which floor.

A situation where the user loses the control marker from sight temporarily is illustrated in Figures 2.13 and 2.14 below. While inspecting the mesh from the left side the user accidentally moves their head too far to the right. The control marker is now outside the user's field of view.

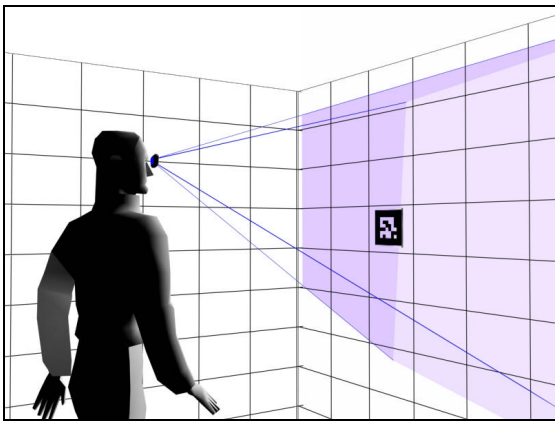


Figure 2.13. User looks too far to the right, losing the control marker from view (External view).

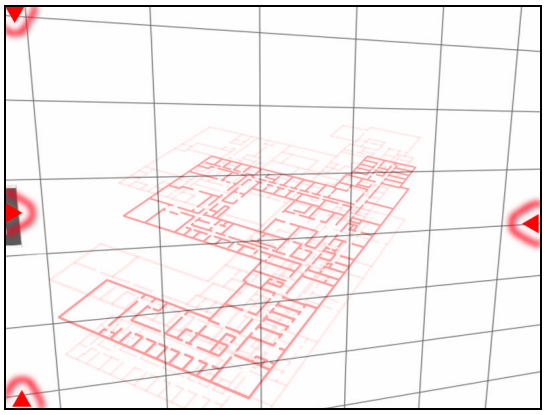


Figure 2.14. De-activation of time-out, gradual reduction of opacity. Control marker position indicators flash, prompting user to reacquire marker (User's view).

Content associated with the marker (3D mesh) is immediately reduced in opacity, while the indicators of control marker positions flash, indicating the 'last seen' position and prompting the user to reacquire the marker. A time-out is activated, unless user performs a 'hold' gesture with the glove (for instance clenching of whole fist) which freezes the timeout, allowing for a momentary distraction, like talking to someone briefly before resuming content browsing.

Timeout countdown is represented by a gradual reduction of opacity of the building mesh. If the marker is not reacquired within a predetermined amount of time, the building mesh disappears entirely, the content browsing mode is terminated and the active area reappears (see Figure 2.4). This is a perfectly acceptable method of exiting the browsing mode. Another method can be performed by the user directly. In this method the user performs an 'exit' gesture (rapid flicking of thumb, for example).

In the example the user reacquires the marker quickly enough, and resumes browsing, as illustrated in Figures 2.15 and 2.16

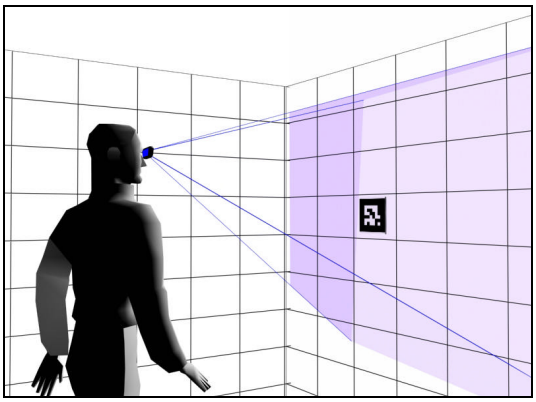


Figure 2.15. User reacquires marker (External view).

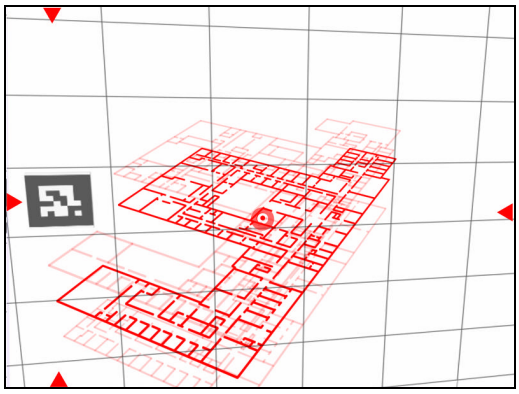


Figure 2.16. Content browsing is resumed (User's view).

2.1.2.1.1 Implementation

The majority of the features described in the above scenario have been implemented successfully. For example, the rendering of an interactive 3d building mesh in the implementation is illustrated on Figure 2.17.

Several non-priority features described in the above scenario were omitted in the implementation as they were not crucial to the overall functionality. They include the time-out on content browsing, 'hold' gesture and using the physical marker's size as another parameter for the visualization. These features were retained in the above scenario as guidelines for future development, and their implementation should be straightforward.

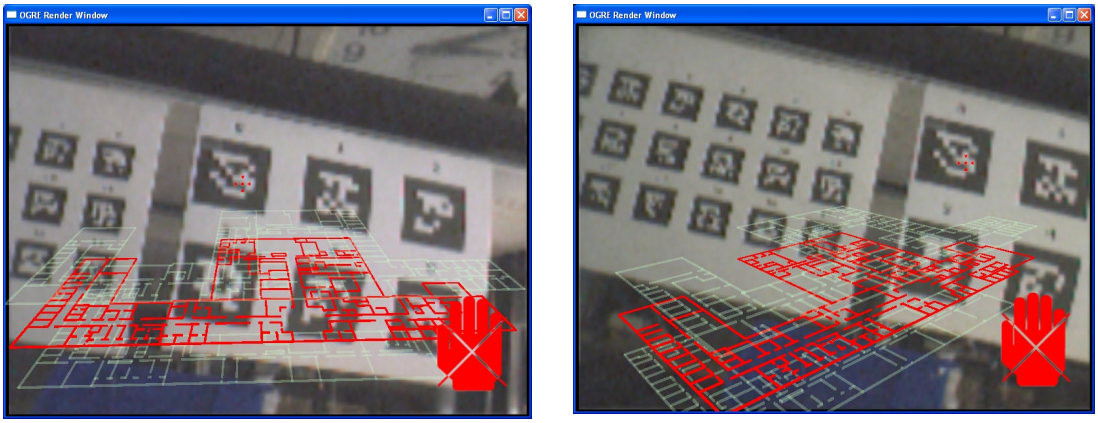


Figure 2.17. Screen captures showing the implementation of the interactively rendered 3D building mesh.
Right: viewed model reacts to head movement by rotating.

2.1.2.1.2 Scenario Variants – Content Types.

The mappings between the control marker's position and the camera viewpoint used for a content object viewing can differ depending on the object types involved. For instance, when viewing a 2D schematic, left and right head movement might be mapped to horizontal panning, or clockwise and anticlockwise rotation, while up and down may be mapped to vertical panning or scale. Two additional types of content object types beside the already described 3D model are presented here, alongside screenshots of their implementation.

2.1.2.1.2.1 VARIANT: Content Type: 2D MAP/SCHEMATIC)

2.1.2.1.2.1.1 Motivation

It can be expected that in majority of cases the annotation information transferred into the system will be originally provided in printed form on paper, and then scanned into digital representation. The system therefore must include a flexible planar bitmap image viewing functionality, to be used with floor plans, blueprints, diagrams and other 2D imagery. Provisions for panning, scaling, rotating may need to be provided to facilitate exploration of the data, using a

combination of the control fiducial marker and glove-based input, as in the case of the 3D content described in the previous section.

Figure 2.18. Panning a 2D bitmap floor plan in the implementation.

2.1.2.1.2.1.2 Implementation

2D bitmap viewing functionality was implemented as shown on Figure 2.18. Imagery provided to the system as bitmaps can be overviewed and examined in detail. The position of control fiducial in the field of view is used for panning the image, which provides the illusion of a glass pane overlaid with the translucent image floating between the user's eyes and the observed environment, allowing for head movements to naturally translate to the change of viewed image region. Glove gestures are used in order to zoom in and out of the image.

2.1.2.1.2.1.3 Further Work

- Vector Based Maps

Since 2D floor plan data may be also available in vector-based file format (CAD), the viewer described in this section should also recognize and handle planar vector data. This type of

content can be considered a special case of 3D model type content, still the interaction methods involved will be identical to those used for handling a regular bitmap image.

- Overlay Blending Techniques to Support Divided Attention Tasks

It would be desireable to implement customized methods of blending the overlay imagery with the underlying video background, in a manner which preserves the maximum of visual information from both layers. Such techniques are researched typically in the context of traditional desktop computing interfaces and displays, with the goal of supporting divided attention tasks. It can be argued that some of them may be applied in the context of Augmented Reality. For example, "multiblending" proposed in [Baudisch04] involves smart image filtering (blurring, high-pass 'sharpen' filtering, color channel separation) based on the properties and purpose of visual information contained in the foreground and background layers. A simple example of the benefits of such filtering is demonstrated in the prototype's implementation of the 2D map content object. The floor plan bitmap (shown on Fig 2.18) used for the prototype demonstration has been pre-processed to use non-uniform opacity combined with high-pass filtering. Edge information critical for a floor plan was preserved at full opacity while more of the video background was allowed to show through the areas with no high spatial frequency features in the foreground.

A near real-time implementation of high-pass filtering has been experimented with (utilizing real-time image processing algorithms[8] provided by OpenCV library), however it was not incorporated into the prototype because of the computational requirements.

[8] Canny edge detection algorithm, and distance transform were used in creating the opacity (alpha) map for the 32-bit bitmap.

2.1.2.1.2.2 VARIANT: Content Type: 360^0 VR PANORAMA

While providing a necessary navigational aid within the floors of a building or other structure, 2D or 3D maps are unlikely to contain many detailed specifics about the contents and the layout of a given room or area. A 'VR panorama' photography based addition to the types of content offered by the system is proposed and implemented.

2.1.2.1.2.2.1 Background

VR Panoramas are based on a computationally low-budget Virtual Reality technique which allows one to present the user with scenes in which they can freely "look around", although without the ability to change the position of the 'virtual camera'. A fixed observation point allows one to utilize 'environmental mapping' rather than real-time world-model computations in order to generate the displayed imagery. Depending on the type of mapping, a geometrically simple object such as a cube, a cylinder or a sphere is covered (projectively textured) with a precalculated synthetic or photographic environment bitmap texture. A virtual camera placed in a fixed point in the center of the cube provides for realistic undistorted views of the original scene. An example of a cubemap (environmentally mapped texture based on the 3D shape of a cube) is pictured on Figure 2.19. Similar techniques are widely used in computer games, for example to provide the images of sky or distant vistas beyond the drawing distance used by the 3D scene renderer.

VR panoramas were popularized by the inclusion of the QuickTime VR format into Apple Computer's QuickTime multimedia family of formats (described in [Chen95]), and soon an international community of QTVR enthusiasts was created, which ensured the wide availability of authoring software, some of it available as freeware. Photorealistic panoramas can be created from series of preferrably wide-angle photographs which are "stitched" together using appropriate software, such as:

- Helmut Dersch's free Panorama Tools[9]
- Pictosphere™ GoSpherical[10] (commercial)

  This package is notable for accepting 180^0 fisheye lens photographs as input – allowing a complete 360^0 panorama to be stitched from as few as two original extreme-wideangle photographs, largely facilitating the process.

An extension of the static VR panorama concept is a video-texture based panorama (PVT), described in literature by [Foote00], [Sun01] , [Agarwala05]. PVT differs from a static VR panorama in that the scene viewed by the user is animated. It is interesting to note the commercial availability of multi-camera systems such as the Ladybug®2 Spherical Vision system from Point Grey Research [http://ptgrey.com] which fuse the video streams in real-time, allowing to easily record footage of video-texture based panoramas.

2.1.2.1.2.2.2 Motivation

The rationale for including Panoramic VR content functionality into the prototype system is that it allows the responder to get a detailed "surround" overview of a location before they physically enter it, aiding navigation in limited visibility conditions. The knowledge of spatial arrangement of non-permanent fixtures such as furniture and contents of storage (in a chemical lab), equipment (servers in a server room), machinery (in a factory or a building site), cargo in a ship hold or wares in warehouses, can be beneficial in certain situations, such as figuring out the shortest path through a dangerous area, or the location of possible sources of hazardous material contamination.

In the case of a structurally damaged or altered location, it would be possible to view its original state and make decisions based on the enhanced set of information – for example when

[9] [http://www.all-in-one.ee/%7edersch/]

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

rescuing people from a collapsed building. As opposed to a blueprint or map, a photographic image conveys a greater richness of information and is easier to assimilate cognitively.

Possibilities for quick annotation exist, as the six bitmaps which are used for the cube mapping can be easily edited as any other bitmaps, allowing for addition of textual/pictorial annotations.

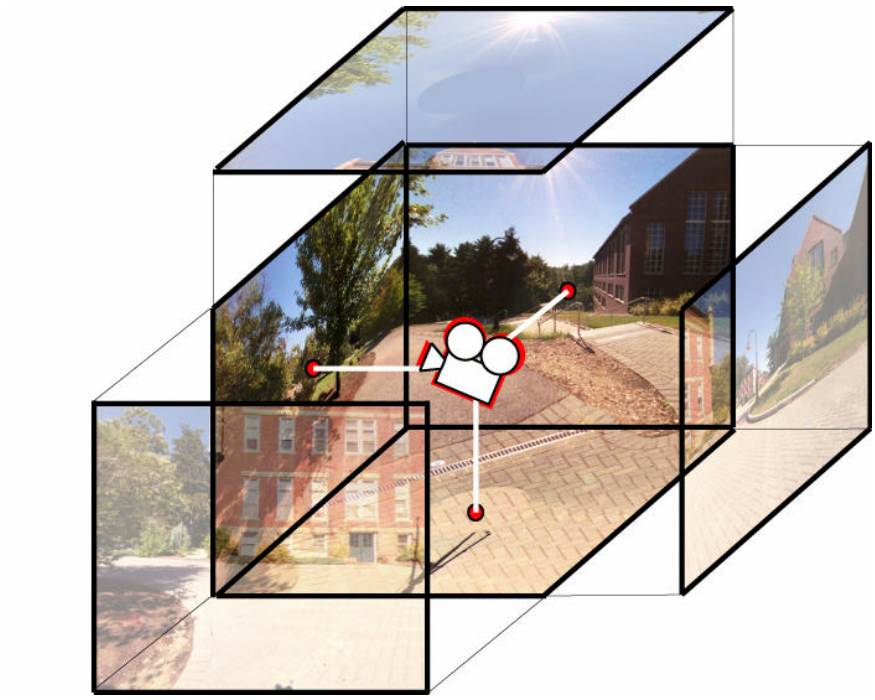
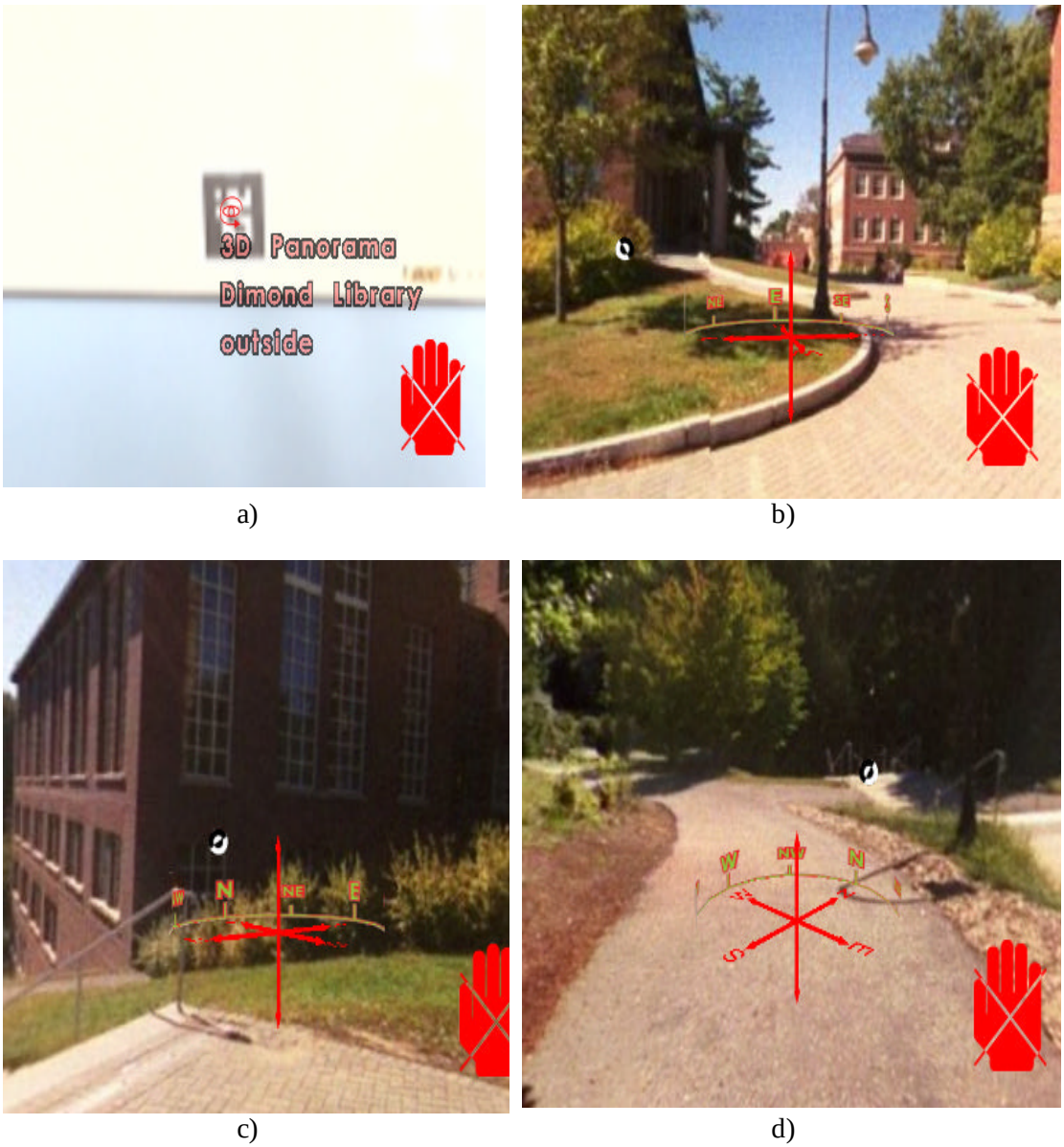


Figure 2.19. A cubemap of an outdoor area on the UNH campus.
The virtual camera used for rendering the scene is placed in the center of the cube, its rotation corresponds to a change in user's viewpoint. Photography and processing by author.

2.1.2.1.2.2.3 Implementation

The implementation of user interaction with a VR panorama content object implemented in described system is illustrated on Figure 2.20. Upon activating a fiducial marker label associated with a Panorama object (Fig. 2.20a), the user is presented with the view of the location (Fig. 2.20b). The user can look around in a natural fashion by turning their head around (Fig. 2.20b,c,d). The original fiducial's position in the field of view (indicated by a small rotating high-contrast pointer) is translated to the yaw and pitch of the virtual camera. The movement is

exaggerated, allowing the user to view the scene in it's entirety by making limited head movements, as the original fiducial needs to remain in the camera's view frustum the whole time. The virtual camera's starting viewpoint, corresponding to the middle of the screen position of the original fiducial, is encoded in the database, allowing to properly orient the panorama with regard to the fiducial's real world placement on a wall. A 3D compass is also displayed to show directions in the virtual reality of the panorama.



a) b)

c) d)

Figure 2.20. Screen captures of user's interaction with a Panorama content object. High-contrast rotating pointer provides reference to the fiducial position in field of view.

2.1.2.1.2.2.4 Further Work

Some intended features which were not implemented but would be a natural extension of the implemented functionality are:

- Control of the opacity of the viewed panoramic VR scene using glove gestures. Allows to specify the video background and overlay relative contribution

  Implementation suggestion:

  - two finger control of opacity
  - selected finger's absolute position (bend) mapped to the speed of change of the opacity value
  - The speed of change would correspond to the difference between current fingerbending sensor read-out and the threshold value
  - one finger controls increase, other decrease; both fingers can be used in conjunction, to allow fine-tuning by using difference
- Zooming in/out (by changing the Field-Of-View parameter of the virtual camera.

2.1.2.1.3 SCENARIO 1: Further Work

2.1.2.1.3.1 Multiple Control Markers

Using one marker as a control limits the degree of freedom (since the marker must be visible in the camera view to be used as a dynamic control). Therefore, an extension of this idea would be to provide groups of markers working in conjunction with each other. For instance a large, very detailed 2D map may be associated with a grid of markers placed on a wall. Any of the markers may be used to bring up the map (using an activation gesture while the marker is in the center of view, within the 'active area'), however the starting viewing position of the map would depend on the particular marker used. Changing the marker position within the field of view would allow the user to pan a large map. When more than one marker is visible, their

positions may be interpolated to provide smooth panning transitions when one marker disappears from the field of view, and hands the control function to another one which has come into view.

2.1.2.1.3.2 Other Control Variables

Additional input variables available for mapping are the observed scale and rotation of the control marker. The marker's apparent scale is a function of the fiducial's original size and the distance between it and the user. The difficulty with using scale as an input variable for browsing content is that it requires the user to move around, especially if the marker is a large marker placed at a distance. However, if the physical size of the markers is fixed or otherwise predefined and known to the system, the observed size of marker can be used to estimate the user's distance from the marker in an attempt to increase tracking precision.

2.1.2.2 SCENARIO 2 (Label Behaviors)

Marker labels are augmentations displayed by the system over the detected fiducials viewed through the head mounted display, their purpose is to attract users attention to the availability of annotation information. The labels consist of a graphical symbol (icon) indicating the type of content associated with a marker, and short textual description further identifying the content. The labels implemented in the user interface are very similar to *icons* used in typical WIMP (Windows, Icons, Menu, Pointing Device) interfaces. The labels in the system are used as visual mnemonics as well, however unlike icons their position is not fixed on the computer display. Instead, they appear to have a fixed position in the real world (as viewed through the AR device). Figure 2.21 shows several fiducial markers overlaid with labels. The graphical symbols for available types of content were designed to be visually distinct, and partially transparent in order to reduce visual conflicts with the video background. For future iterations of the design it is

strongly suggested to conduct a small user study to determine effectiveness of the icon set design, and the chosen rendering of the text.

Figure 2.21. Several fiducial markers augmented with marker labels of different type.

As mentioned before, the appearance of a given label depends on the on-screen position of the center of the detected fiducial. The graphical icon's appearance is static in the implementation; only the text description is rendered larger and more opaque as the label moves closer to the center of the screen. The required mapping is provided to the system in the form of a grayscale bitmap, like the one pictured in Fig. 2.22. The brightness of a particular pixel of the bitmap corresponds to the visibility weight associated with corresponding pixel coordinates of the video display. Additionally, an 'active' area is defined within the bitmap, as the area holding pixels with brightness above 50%. In order to interact with the content tied to a particular marker, the user must position their head so that the marker's label on screen location corresponds to that 'active' area on the bitmap. Since other labels may be present within the area, and only one marker can be interacted with at given time, the weightmap is consulted to determine the label with the highest weight value. That label is 'selected', and changes appearance in order to provide more detailed information, and indicate that activating the content is now possible. A label in a

'selected' state is expanded, providing additional lines of text description. Figure 2.23 depicts a number of labels, one of which is in 'selected' state (label text rendered in red).

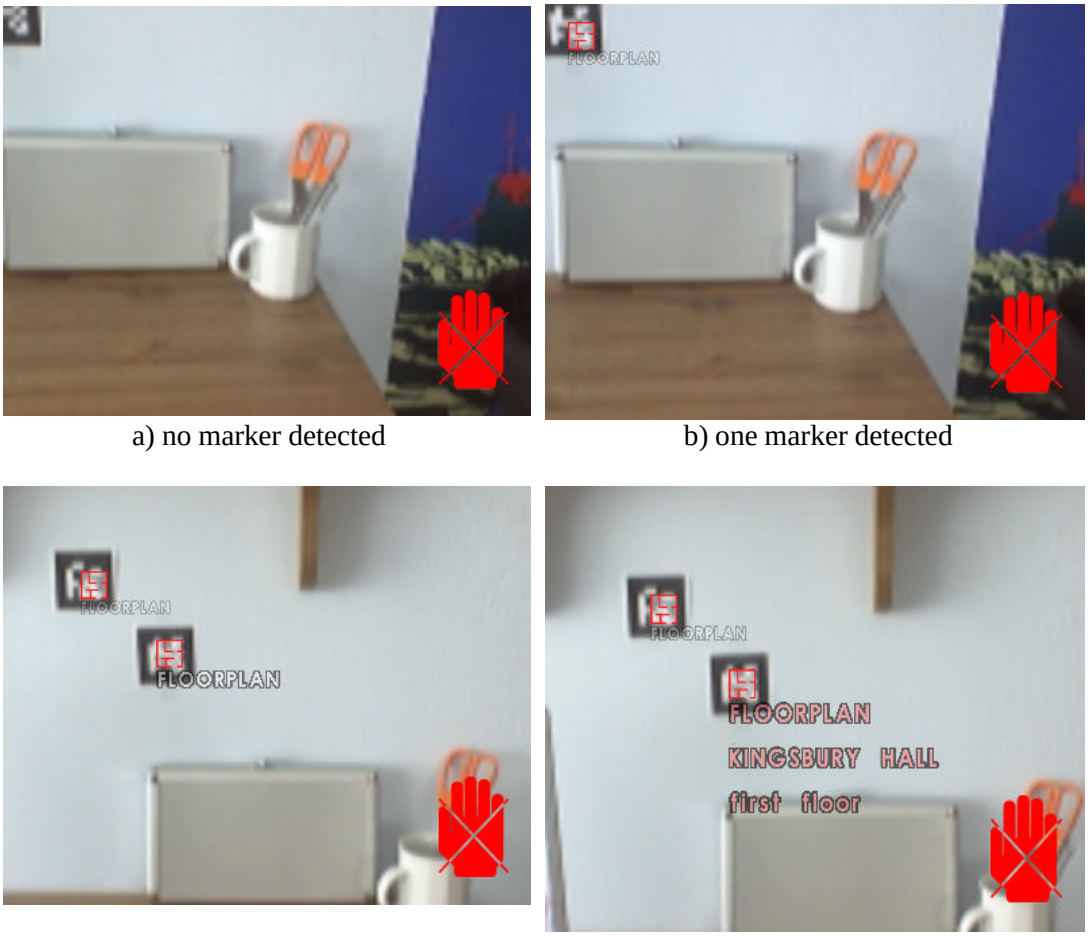
Figure 2.22 Grayscale bitmap encoding the weight mapping and specifying 'active' areas of the display for the purposes of marker selection.

Figure 2.23. Appearance of a label in 'selected' state.

A simple two label scenario illustrating the effects of label weightmap and active area is presented in Figure 2.24. The scenario starts with a 'no fiducials detected' situation shown on Fig

2.24a. ,where the fiducial in the upper left corner is not yet entirely within the camera's view. In 2.24b, as the user moves their head up diagonally in the leftward direction, a fiducial is detected and is immediately augmented with a label. The weight corresponding to the label's position is a low value, therefore the text description is not prominent visually. In the next step, as the user's gaze continues moving up, two markers are detected. The label of the bottom marker has greater weight, thus the appearance of the text is stronger. In stage shown on Fig. 2.24d the label of that marker has entered the active area, and more information is presented to the user.

.



a) no marker detected

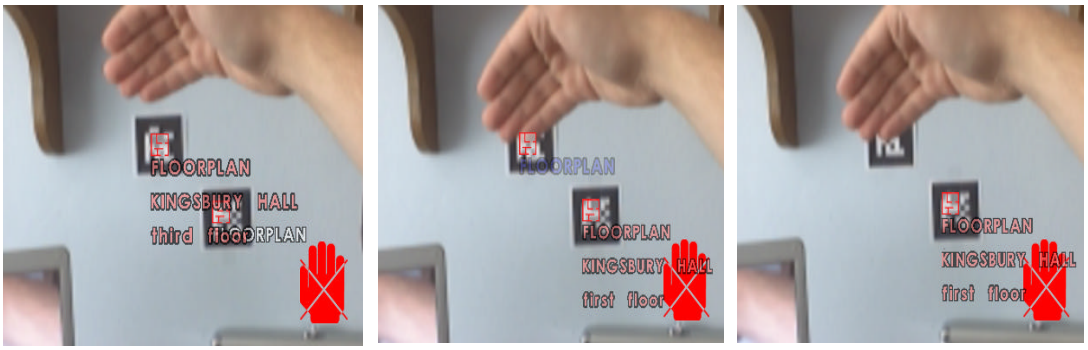
b) one marker detected

c) two markers: different weights.

d) lower marker moves into the active area and becomes selected.

Figure 2.24. Effects of label weightmap and active area as the user is panning her point of view across two fiducial markers.

In certain cases when a fiducial's visibility is lost, it is desirable to retain the associated label on screen for a short period of time instead of removing it immediately. It can be argued that such label behavior results in a more 'smooth' interaction in cases when a fiducial is temporarily obscured (for example by a passing person, or the user's hand), fails to be detected because of motion blur in a given frame of the camera video stream, or goes in or out of the user's field of view. In this manner, label flicker is minimized. The implementation of the behavior is shown on Fig. 2.25. In the first step (Fig 2.25a) two markers are detected, and accordingly two labels are shown. Both fiducials are within the active area, however the top left fiducial's position has a greater weight associated with it (being closer to the center of the screen), therefore the corresponding label is selected and shown in expanded form (three lines of text description). In Fig 2.25b, the selected marker is obscured. The system reacts by activating a timed opacity fade-out of the associated label (shown in blue). Since the fiducial which was previously selected has been internally removed from the currently visible fiducial list, the second fiducial is now selected. After a predefined timeout, the obscured fiducial's label is removed from view altogether (Fig 2.25c), unless the fiducial becomes visible (back to state shown on Fig 2.25a). Since no marker position updates are available for an obscured fiducial, in the implementation the 'fading' label's position is fixed, even if the camera viewpoint has changed since. A suggestion for future work would be to use motion prediction based on the label's previous displacement vector, or relative position to other visible fiducials.



a) b) c)

Figure 2.25. Timed fade-out of obscured marker labels.

2.1.2.2.1 Further Work

The visual behaviors contained within the GUI implementation described above are meant to provide a functional implementation of the general interaction concept in which the user's gaze is used for highlighting and selecting information. Several GUI design choices have been made in an attempt to avoid overcrowding of the display and to diminish visual interference with the responder's view of the surroundings. The design particulars of this implementation are not definitive. However, they should provide a good starting point to a more elaborate design based on an iterative process involving user feedback. Modifications to the visual appearance and behavior of the interface can be applied in a straightforward manner. For example, the weightmap image provided to the system can be easily modified to provide several active areas instead of one, or a completely different pattern of the label on-screen visibility. It may be, for example, required that the center of the screen should be kept relatively clear of augmentations, and the active area should be located in the lower section of the display. An example weightmap specifying such system behaviour is shown on Fig. 2.26.

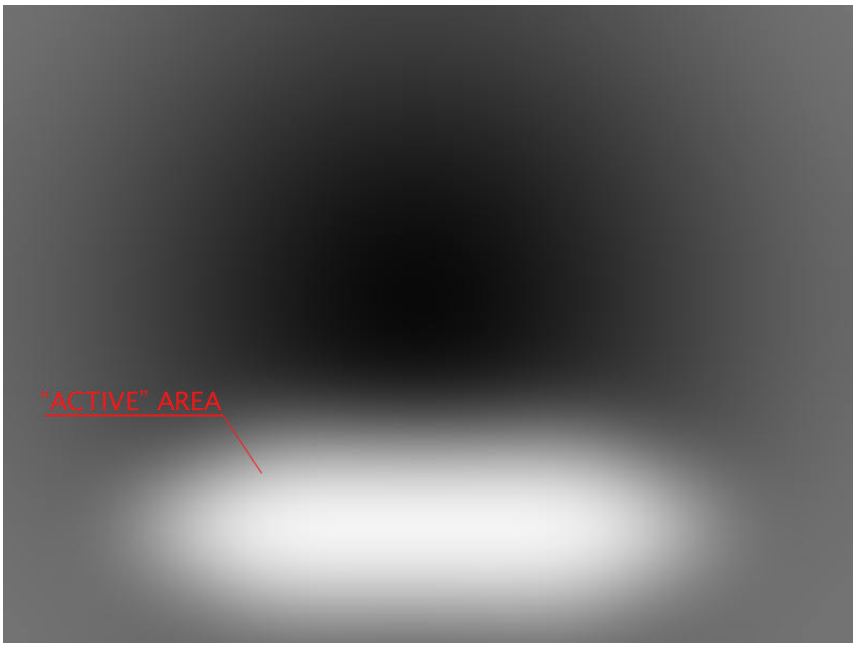


Figure 2.26. Alternative weightmap image.

2.1.2.3 SCENARIO 3 (Team Member Tracking)

The following multi-user scenario demonstrates:

- teammember tracking based on fiducials viewed by them.
- "You Are Here" self-localization
- teammember position visualization
- Incident Command Center visualization of building
- using the same content object (3d building model) with multiple fiducials

This scenario, illustrated on Figure 2.27, involves a group of responders scattered throughout the building where an emergency occurred. A mobile control center has been set up close to the scene, allowing the Incident Commander to coordinate team manoeuvres. The responders are equipped with the AR device, and fiducial markers are present in the environment, allowing for position tracking. The appearance of the responder's device displays as well as the Command Center visualization is shown on Fig. 2.27 using screenshots from the implemented prototype setup used in the demonstration.

2.1.2.3.1 Self-localization

While walking down a corridoor, responder A (shown in green on Fig. 2.27) briefly views a fiducial placed on the wall. The marker is recognized by the AR device, and thanks to a database of markers locally available on the mobile computer, responder's approximate position in the building becomes known to the system. Responder A activates the available content associated with the marker. The appearance of the device display at this stage is illustrated on Fig. 2.27 as "RESPONDER A's DISPLAY". The viewed content object is a 3D model of the building. Since the responder's position and the fiducial's facing direction are known, the building model is immediately shown oriented properly with regard to them. The building model is always positioned on screen so that the current location of responder (indicated with an avatar) is always in the center of the display. The location is also used as origin for 3D model rotations or scaling, providing an intuitive ego-centric point of view.

2.1.2.3.2 Team Member Position Visualization

At approximately the same time, Responder B notices a fiducial placed near a staircase. His position is registered by the system. Through the wireless link, the centralized responder tracking database located within the mobile control center is updated with the position of B. Upon activating of the fiducial content, the location tracking database is queried by B's device, and the responder is presented with a display shown on Fig 2.27. ("RESPONDER B's display"). B's position within the building is indicated with an avatar (shown in blue), alongside other team members avatars (shown in orange). Responder A's location is indicated by the green avatar. If responder A changes position (views a marker in another location) while Responder B is still viewing the building model, the position of A's avatar on B's display will be updated as soon as the updated position information propagates from A's device through the centralized database to B's device. The same applies to all other team member's positions.

It is worth noting that although responders A and B access the building model via different markers, the underlying content object is the same. The differences in the visualization result from the location information associated with the different marker ID's in the marker database. Such approach allows to provide access to a content object through different physical locations which makes sense in the case of the building overview schematic described in this example.

2.1.2.3.3 Control Center Building Visualization

The centralized position database can be accessed from a workstation in the mobile command center, allowing to present the Incident Commander with a building visualization overlaid with indicators of team member positions. A simple application accessing the responder tracking database and visualizing the team member placement has been created for the purposes of the prototype demonstration. A screen capture from the application is presented on Fig. 2.27, captioned "Command Center Display".

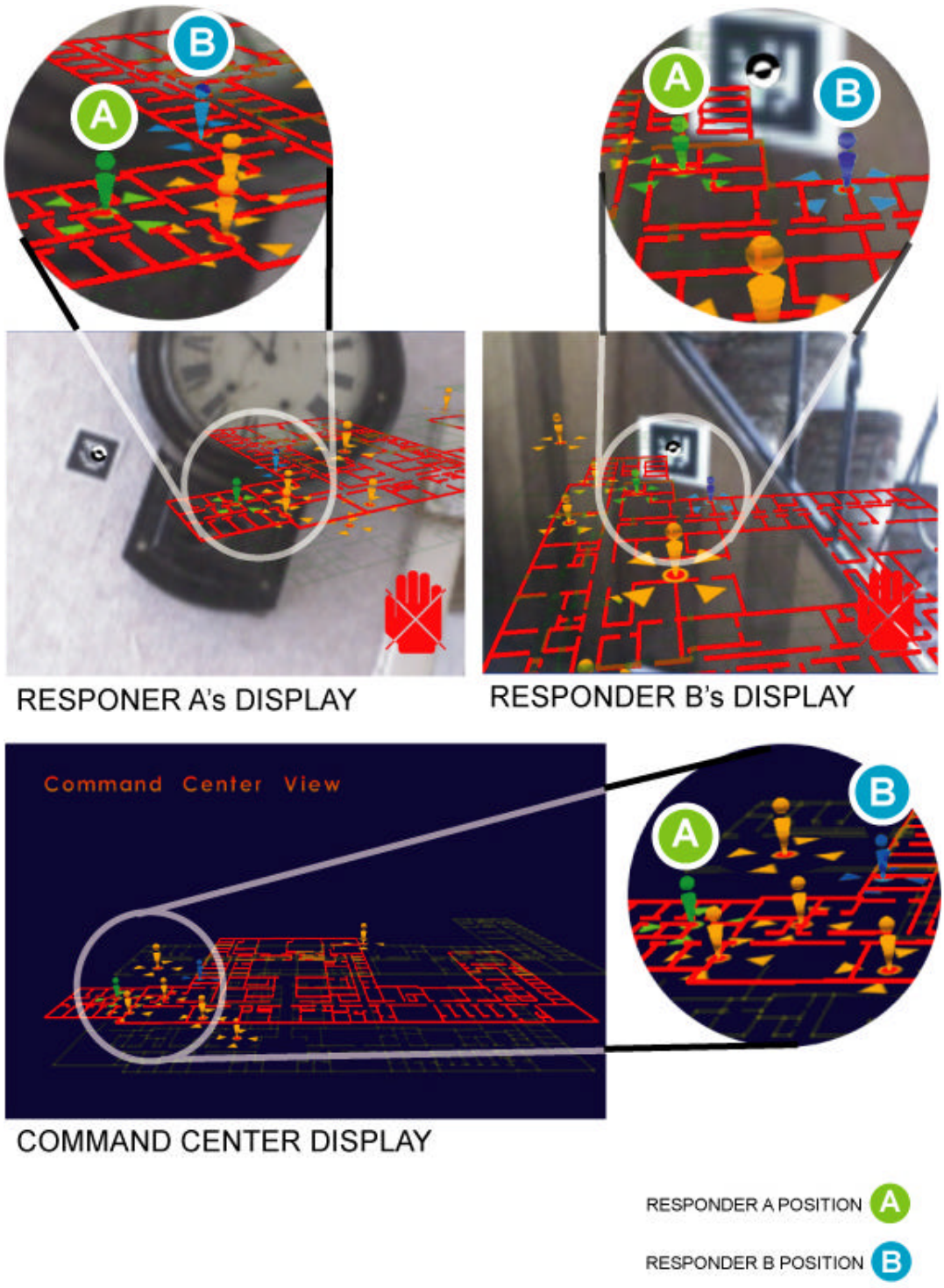


Figure 2.27. Team member position tracking

2.2 General Overview of the System's Deployment

The proof-of-concept prototype described here implements a subset of the first response system functionality in the end-target environment, with emphasis placed on end-user's interaction with the system. The target environment is a building affected by an emergency. In the usage scenarios presented in the preceding section it was assumed that the annotation data pertaining to that building is already available and present on the responders' mobile units and on the Incident Command Center's workstations.

This section presents a wider perspective on the deployment of the system as a whole. The described system's usefulness and effectiveness hinges on the completeness and availability of the annotation data for the largest possible set of locations. Therefore, for the final product to be deployed on a massive scale, a considerable effort will be required in order to create a suitable support infrastructure allowing for this information to be created, stored and distributed. Furthermore, as the validity, integrity and accuracy of the data are essential for a life-critical application, creation of new standards and guidelines may be necessary to ensure the data conforms to the system's and the responders' requirements. The next logical step is to initiate regulatory efforts extending the existing safety signage requirements into the digital domain.

The following remarks serve as a generalized blueprint for the evolution of the proposed system. The discussion is organized around the main stages of the flow of location annotation information within the system, beginning with the creation of content and ending with it being delivered to first responders during an emergency and utilized by them.

2.2.1 Stage One: Creation of Annotation Database for a Location and Fiducial Placement

The potential users of the system at this stage (see Figure 2.28) are the local administrators, users and inhabitants of the building, developers, city planners, and other involved

authorities. In this content creation stage it has to be determined what types of digital content relevant to the location should be made available to first responders through the system. The information then has to be created or transferred from other media and entered into the system in an appropriate format. In the next step, placement of fiducial markers within the physical location has to be determined and effected. Several issues pertaining to these two important steps will be discussed here.

One very crucial issue which needs to be resolved before the described system takes final shape is the question of the open-endedness of the annotation database. On one hand, a strictly regulated set of minimal requirements imposed upon the system users responsible for populating the system databases would facilitate systematic data entry and predictability of end-user (responder's) experience. It could be for example made mandatory to simply include digital versions of existing paper floor plans containing fire escape route information and information about presence of hazardous or toxic substances, mirroring the existing general safety requirements.

On the other hand, since the digital medium offers unparalleled possibilities for easy annotation of locations in a more spontaneous, unrestricted manner, it allows for inclusion of data which may be unofficial, incomplete or otherwise not conforming to regulatory demands. Such content may potentially be of great use to responders, and may include annotations to the purpose and operation of machinery, switches, valves or any other elements of infrastructure, lists of hazardous materials stored in a given location, pictures of the interior annotated with verbal explanations, scans of hand-drawn sketches and other material generated by hands-on users of the buildings infrastructure.

Separating the content into "mandatory" and "unofficial" subsets and reflecting that distinction in the way it is displayed to the end-user appears to be a reasonable middle-ground solution. Exactly what types of information would be most helpful to responders is a question that would be best answered by them, therefore it should be part of a user needs study conducted

with firefighters, fire chiefs and other first response professionals as one of the vital next steps in the system's development.

From a technical standpoint of the data preparation and entry, there are several issues which need to be resolved. First, the complete range of possible content objects (some types of which were suggested in the preceding section) needs to be defined, and support for them incorporated into the system.

2.2.1.1 Authoring Tools

One of the cornerstones of the envisioned system is that the data integration effort is decentralized and undertaken at local level, thus reducing the overall system deployment and running costs as well as the delay in introducing new updates. Such approach calls for creation of suitable authoring tools, simplifying the procedure for the data-entry responsible party by means of 'wizards' and extensive step-by-step instructions along the whole process.

This set of tools should be considered an integral part of the system, as the successful deployment of the system and its usefulness largely depends on cooperation of the users at this stage. Several of the areas of functionality which need to be provided are listed below.

2.2.1.1.1 Editors, Importers and File Conversion Utilities

Depending on the final set of content object types which are to be supported by the system, provisions for creating given type of content from scratch or importing if from files created using external tools should be included in the content authoring tool. Table 2.1 presented below tentatively lists selected types of content alongside corresponding popular file types.

Figure 2.28. Building annotation database is produced locally and electronically forwarded to a centralized repository.

In some cases, the file types are proprietary, thus it has to be decided whether the licensing costs outweigh the benefits of systems interoperation with an existing data format. There is a number of exporter utilities for popular digital content creation tools available for Ogre3D graphical engine used in the presented implementation, some of them were used in preparing the sample content used for the prototype demonstration.

Another issue to be considered when developing the authoring tools is the verification of properties of files being imported, such as file size, image resolution, size and color depth, sufficient image contrast, 3d model complexity and density, audio file sampling rate or compression settings, use of unsupported features of third-party file formats. Restrictions on some of these properties will most likely need to be imposed to match the systems capability and ensure robustness. Reduction of image size, simplification of a building mesh while retaining required level of detail, downsampling of a voice annotation sample file etc. performed during the importing operation could go a long way in facilitating system usability. Additionally, the

embedding into the authoring tools of simplified editing functionality for rapid annotation of certain files, for example for adding text to a photograph, scaling or rotating a 3D mesh could reduce the number of workflow steps needed by the user.

Written annotations	**.TXT, .RTF, .DOC, .PDF:** plain ACII text files, formatted text, text with embedded graphics **.HTML:** browseable hypertext with graphics
Bitmap Image	**.TIFF, .GIF, .JPG, .PNG:** all above supported by Ogre3D Used for: floor plans, site plans, building elevation, schematics hand-drawn sketches photographs
Vector Graphics	**.DXF:** Drawing eXchange Format, Autodesk's layered vector drawing format recognized by most CAD software packages **.DWG:** Native Autodesk's AutoCAD file format **.SHAPE:** Popular ESRI file format used in GIS systems (geographic information systems) **.DGN**: Intergraph graphics file format often used with digital topographic maps Used for: floor plans, site plans, building elevation, schematics maps

Table 2.1. Several content types and file types typically associated with them.

Table 2.1, continued:

3D Model	**.3DS**, .**MAX:** format used in the popular Kinetix' 3D Studio modeling software. Exporter for 3D Studio for Ogre3D available **.KML**: Google Earth's Keyhole Markup Language – an emerging standard in geo-annotation **.MESH**: Ogre3D internal mesh object .**WRL:** VRML (Virtual Reality Modeling Language) three-dimensional vector graphics format for the Web, the geometry can be animated or interactive. Format useful for educational purposes, but superseded by: **.X3D**: previously called VRML200x Used for: building models visualization of building's fixed infrastructure (piping, electrical systems, communications and IT systems). models of machinery and equipment annotation for controls of heavy machinery illustrating the freedom of movement (e.g. cranes)
VR Panorama	**.JPG, .PNG, .TIFF:** environmental cube maps (as described in section 2.1.2.1.2.2) provided as sets of six bitmap images – ***supported natively by Ogre3d*** **.MOV:** Quicktime VR panorama files Used for: realistic photographic depictions of the environment immersive diagrams of environments
Audio file	**.WAV**, **.MP3:** audio files, w/ or w/o compression Used for: voice annotations or guided instructions
Video file	**.AVI**, **.MPG**, **.MOV** – multimedia video files (with or w/o sound) Used for: animations illustrating operation of equipment fly-through visualizations of environment layout hands-on video tutorials

2.2.1.1.2 Georeferencing of Content

Some types of the authored content, such as floor plans or building models, are intended to provide reference frame for the team member tracking functionality, and/or need to be displayed properly oriented with regard to responder's current position and facing direction (as illustrated in the examples in Section 2.1.2). Therefore there is a need for embedding geolocation information within the content. For example, for a drawn-to-scale floor plan provided as a planar vector graphics file, the corresponding real-world geographical coordinates of at least one point represented on the plan must be known alongside the plan's scale and orientation relative to north. The geodetic coordinates may be provided as latitude and longitude, in UTM (Universal Transverse Mercator[11]) coordinates, or in an arbitrary local coordinate system as long as it is internally consistent (although the latter option is not recommended because of lack of interoperability). At this stage, assistance from local planning authorities may be required.

Leveraging on existing Geographical Information Systems (GIS) is an interesting possibility worth investigating. As the recent examples of Google Earth, Google Maps, Google SketchUp[12] demonstrate, the user base of GIS systems is rapidly expanding to include general public. Simple controls and intuitive user interface metaphor of Google SketchUp makes it possible for anyone without extensive 3D modelling training to create relatively sophisticated building models which can be then imported into the Google Earth application. Other leading providers of GIS systems appear to recognize the strength of Google's approach – and follow suit, as in the case of ESRI's (www.esri.com) soon-to-be released ArcGIS Explorer which targets

[11] the Universal Transverse Mercator (UTM) grid – mapping grid adopted by the National Imagery and Mapping Agency (NIMA) for military use throughout the world

[12] Google's [www.google.com] forays into GIS technology include Google Maps – a geographical search engine leveraging on availability of satellite imagery of the planet, Google Earth - an application for exploring a combination of satellite photographic data and georeferenced 3D objects (which can be edited),and Google SketchUp which is a powerful and intuitive 3D modeling application with an easy learning curve.

a very similar market as Google Earth, and ESRI ArcWeb Services which are an equivalent of Google Maps. The availability and ease of use of Google Maps API led to emergence of a novel Internet phenomenon such as Google Maps "mashups" – combining contents of the GIS with other sources of information. The "end-user empowerment" philosophy behind the above technologies leverages on massive user participation and interaction resulting in a new quality. It can be argued that a similar approach could work well for the system described here.

2.2.1.1.3 Fiducial Assignment and Placement

Since the annotation content needs to be linked to physical locations, the authoring tools should provide the following:

- assignment of a fiducial marker ID from the available fiducial pool for a given building. The number of unique fiducials which can be resolved by the system is determined by the properties of the planar marker system used (discussed in Section 4.4.6.1 of Chapter 4). Since this fiducial set is a limited resource, methods of reusing it wherever the annotation density is large or when there is a number of buildings to be annotated may need to be devised, such as using combinations of multiple fiducials alongside each other.
- The mapping of content to fiducials can be as one-to-one or one-to-many. A specific content item can be assigned to a number of different fiducial IDs placed throughout the environment. This way, the content can be made accessible from a variety of locations, while the different fiducial IDs allow for user tracking.
- producing the marker (for example by printing on self-adhesive material). One of the advantages of the proposed optical fiducial system is the ease of marker creation and mounting by using a low-end inkjet or laser printer and paper stickers. A slightly more advanced scheme could involve printouts utilizing photoluminescent paint to reduce the

impact of changing lighting conditions on systems' robustness. Other more durable materials could be used in marker production as well.

- allowing the user to assign a real world location to the fiducial marker and storing this information into the annotation database alongside the content. A user-friendly method for specifying a location for a marker could be based on precise floor plans of the building provided by the user to the system at the prior stage of database preparation. The user pinpoints the desired placement of the marker on the geo-referenced floor plan. The resulting fiducial printout can include mini-maps accompanying each of the markers to facilitate ease of placement.

2.2.1.1.4 Electronic Submission of Annotation Information

In the envisioned deployment scheme, after the annotation data is integrated at the local stage, it is electronically submitted over the Internet to a centralized repository (see Fig 2.28), from where it can be distributed to first response agencies as required. Alternative means for electronic submission should be provided for locations without internet access (or when the submitted database file size exceeds the upload bandwidth available), by providing CD/DVD media export of the data which can be mailed in this form. A local copy of the database is stored at the premises, allowing for ongoing verification and frequent updates and/or corrections, which also need to be mirrored in the repository.

2.2.1.1.5 Security Considerations

Potentially sensitive data about the layout and contents of real-world locations could be used with malicious intent (terrorism, theft), therefore it is of paramount importance to ensure the annotation information is secure and not disclosed to unauthorised third parties. Some preventive measures are listed below:

- hierarchical authorization system integrated into the authoring tools allowing for setting of database access privileges. Data entry personnel does not require full access to the annotation database, modifications to the data should be also restricted and logged to prevent tampering.
- encryption of stored data
- private/public key asymmetric encryption/identity verification scheme incorporated into the digital submission mechanisms to prevent modifications and eavesdropping when the data is in transit.
- Existence of a well-defined information security policy in place, both at the local stage (building) premises as well as at the centralized repository and any other storage locations (mirror repositories).

2.2.2 Stage Two: Storage, Distribution and Updating of Content.

2.2.2.1 Centralized Storage of Building Annotation Information

The annotation databases are submitted electronically to a central information warehouse (see Figure 2.29), where the data is stored securely and released only to authorized requesters, such as first response agencies. All updates and modifications performed at the local stage for an individual location should be uploaded to the repository immediately so that the data there always reflects the most current state. There are several advantages to the centralized approach, listed below:

- the repository provides an external storage unaffected by emergencies which could cause a locally stored database to become inaccessible
- the users at the local stage do not need to handle distribution of data on their own, updating of the data to one central location is simpler
- in case of bigger emergencies, for example affecting wide areas, out-of-state fire departments and other agencies may need to be called upon

- if the data exists, it can be found in the repository
- centralized storage allows for stringent security, whereas security in a more distributed approach would be much more difficult to monitor.

A central hub managing the annotation database set in its entirety allows easy mediation of information between parties. Updates committed to the repository by end-users from a particular location can be automatically relayed to appropriate branches of first-response agencies, if they have been registered/authorized for a given administrative/geographical subset of the repository's data. Centralized management of such links and associations offloads some of the administrative burden associated with the system's deployment from the first response agency and facilitates quick access to information when time is of value.

Mirror repositories in several geographical locations should be considered, as they would ensure redundancy in case of a *force majeure* situation, and load balancing in terms of Internet bandwidth availability.

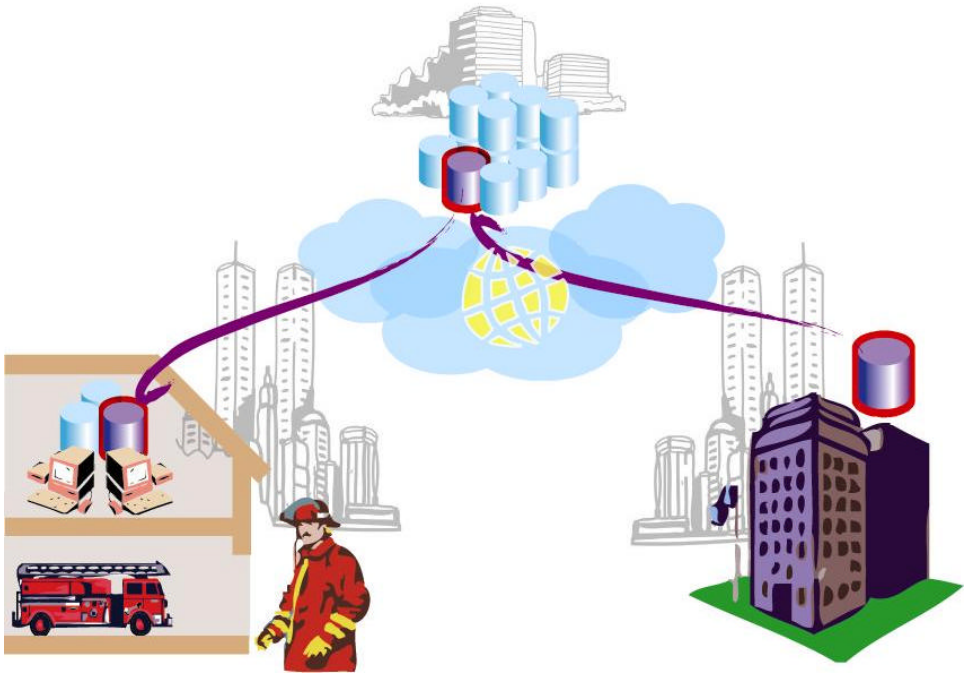


Figure 2.29. Building annotation database is electronically transferred to designated first response agency and updated when necessary.

2.2.3 Stage Three: Deployment of Annotation Content During an Emergency.

The data volume of multimedia content objects for a densely annotated building can be expected to be relatively large. Responder's mobile devices should be able to operate autonomously, therefore during the emergency operation all required annotative data should be stored locally on them. Since the area affected by the emergency is not known in advance of the emergency, a mechanism for quickly transferring the sets of multimedia and fiducial descriptive content for a potentially affected group of buildings to the responder's mobile computers is required. This initialization stage should be preferably performed over a wired networking link or using some form of removable media while the first response team prepares for being dispatched to the site of emergency. The required databases may be downloaded to the mobile devices from the agencies local network if an up-to-date copy is available (after checking with the repository). If the database is not in the possession of the agency yet, or the database is not recent enough, it needs to be downloaded from the repository to the agencies local computer systems first, and then distributed to the mobile devices. The flow of building annotation database just before dispatch is shown on Figure 2.30.

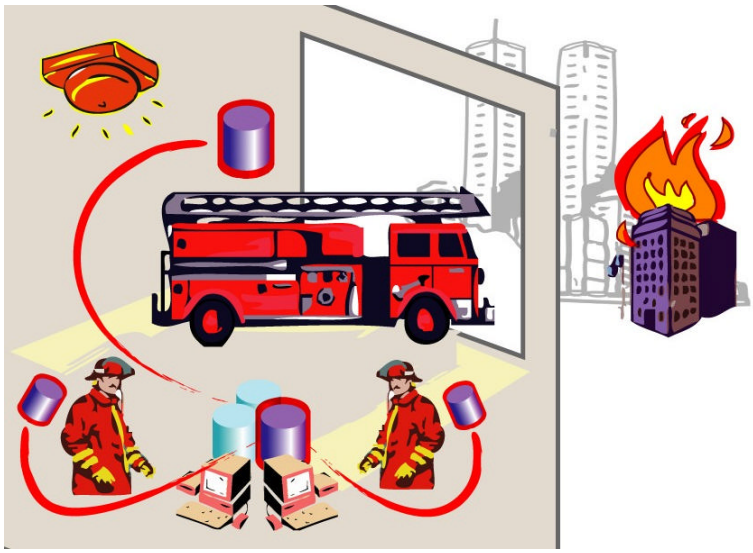


Figure 2.30. When an emergency occurs, the building annotation database is uploaded to emergency services personnel mobile AR devices.

CHAPTER III

HARDWARE COMPONENTS

The project's goal of building a functional proof-of-concept device implementing the envisioned functionality using commercial off-the-shelf (COTS) hardware components has been achieved. The resulting hardware configuration is described in this chapter. The prototype's limitations with regard to wearability requirements and further work directions are also discussed.

The prototype implementation of the system is based on a notebook computer, Fujitsu-Siemens Amilo-D 1840. It is a desktop-replacement class notebook based on a 2.6 GHz Intel Pentium IV processor with Hyper-Threading. More detailed technical specifications can be found in Table 3.1 below. The software was developed and tested using the same computer.

The other hardware components are:

- IO-Display **i-Glasses** HMD display (see Section 3.2)
- Unibrain **Fire-I** IEEE 1394 (FireWire) camera (see Section 3.3)
- Essential Reality **P5 Glove** Controller (see Section 3.4)
- D-Link IEEE 802.11g wireless networking adapter card (see Section 3.5 for networking architecture recommendations)

The hardware components and their interfacing are depicted on Fig 3.1. Fully un-tethered mobile operation of the entire system in its current form would require the addition of battery packs for the camera and the HMD display. The battery packs were not available for testing, thus main power was provided by power supplies plugged into 120V AC and a stationary desktop configuration was used.

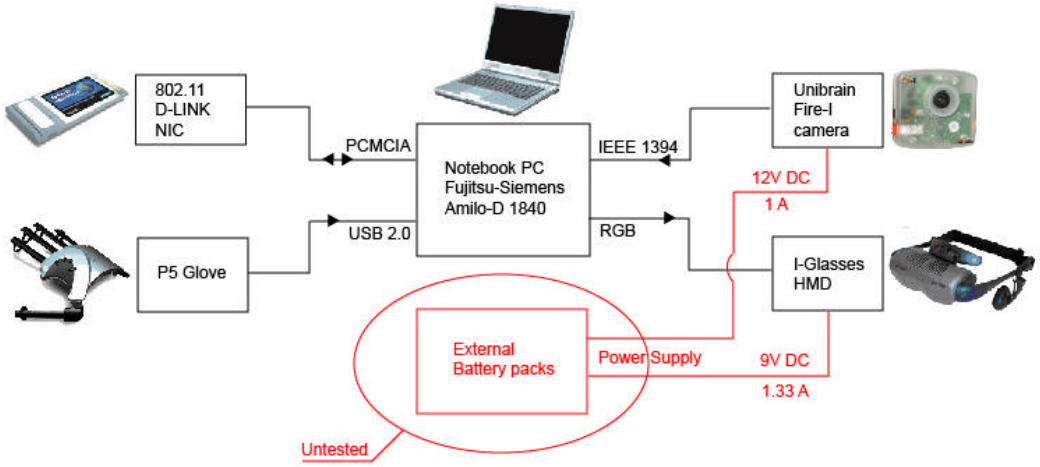


Figure 3.1. Prototype device hardware components.

The following Sections 3.1 thru 3.5 focus on each of the main hardware areas of the prototype. Aside from the current implementation details, suggestions for improvement and future development are provided where appropriate.

3.1 Mobile Computing Platform.

3.1.1 Amilo-D 1840 Notebook: Overview and Technical Specifications

The Fujitsu-Siemens notebook computer, used as the prototype hardware platform as well as the software and content development environment, provided sufficient processing power for the proof-of-concept needs, while still retaining a degree of portability. This allowed reduction of development time and achieving of acceptable realtime performance. The drawbacks of using

such a ‘desktop-replacement’ class portable computer are its high power consumption and relatively large weight. The former contributes to an extremely short battery life, approximately 1 hour only in author’s experience (normal use), and even shorter when the GPU is heavily utilized. A simple test was conducted in which a graphics-heavy application was run in battery-powered mode, starting with the battery fully charged – the battery was exhausted in 40 minutes time. Another related issue is the heat emission, particularly problematic in a wearable computing configuration, for example when the notebook is placed in a backpack worn by the user. Closing the notebook screen blocks one of the ventilation pathways in Amilo-D. Unless additional measures are taken to ensure proper ventilation, this could contribute to overheating and device failure or frequent shut-downs. This problem is also aggravated by the system’s requirement of constant graphics rendering and relatively intense computational processing – further increasing the heat production and need for efficient cooling. Switching off of the notebook’s LCD display does reduce power consumption. Additionally, the whole systems weight should be kept as low as possible for obvious reasons, and the Amilo-D weight is considerable, even after removing the optical drive.

Processor	2.6 GHz Intel Pentium IV HT
Memory	768MB RAM
Graphics Processing Unit	ATI Mobility Radeon 9600, 128MB memory.
Connectors	3 * USB 2.0 ports; 4-pin FireWire (IEEE 1394) port; VGA-out video connector; PCMCIA slot; serial,parallel port connectors;
Battery life	1 hour
Dimensions	35.8 cm x 26.9 cm x 3.6 cm
Weight	3.7 kg / 130.5 oz.

Table 3.1. Specifications of the Fujitsu-Siemens Amilo-D 1840 notebook used for the proof-of-concept

3.1.2 Further Work Recommendations.

The prototype in its present form, while adequate as a proof-of-concept implementation, does not provide a sufficiently reliable and rugged solution necessary to allow for extensive testing in field conditions. Weight and size of the whole system would negatively affect a first responder's safety and performance in a real-life emergency situation. Extending the processing units battery life would be another top priority. Further iterations of the prototype will therefore need to take advantage of the ever improving (processing power/energy consumption) ratio brought about by the ongoing miniaturization in the semiconductor industries. It is recommended for subsequent iterations of the prototype to utilize a mobile processor (Intel's Pentium 4-M or Pentium M CPU's or AMD's XP-M and Turion CPU's) based notebook computer offering considerably longer battery life, less need for cooling and smaller weight, the tradeoff in this case would be potential reduction in processing power.

It is quite likely that in a few years' time miniature embedded devices will offer processing capabilities equivalent to those of today's desktop workstations, allowing for much closer integration of system modules with existing first response equipment and clothing while still providing sophisticated real-time processing and visualization functionality. Already commercially available embedded computers such as, the Thermite® TVC (Tactical Visual Computer) offered by Quantum3D (see Fig. 3.2) are man-wearable equivalents of a PC-based workstation, engineered specifically to provide reliability in hostile environments, and offering a well-matched form and function factors for applications such as the one described here. Should a ready-to-use system such as the Thermite be used as the platform for the described application, the software porting effort could be predicted to be minimal or null, since full PC-compatibility is advertised, and Windows XP is supported.

Figure 3.2. Quantum3D THERMITE® TVC.
Main processing unit shown on left, example deployment for a tactical training application shown on right. Source: Quantum3D website [www.quantum3d.com]

Code optimization or modifications may be however needed to ensure optimal performance on less computationally powerful mobile processors offered with the Thermite (or similar solution), however this seems a highly acceptable trade-off given the extension of battery life and considerable ergonomics advantages.

Some of the features of the THERMITE® TVC computer matching the requirements of a first response wearable computer deployed in a hostile environment are listed below (based on information from the company website, updated 2/17/06):

- choice of Transmeta® Crusoe® or Intel® Pentium® M CPU from 1.0 to 1.4 GHz
- up to 1 GB system memory
- light weight, super-rugged sealed alloy enclosure with Mil-Spec connectors and conduction cooling
- choice of NVIDIA® discrete or integrated Mobile Graphics / Video Module with up to 128 MB DDR2 shared or dedicated frame buffer memory

- Supports accelerated video capture capability for sensor and/or camera input with a wide range of video input formats
- I/O capabilities include:
    - USB 2.0;
    - IEEE 802.11X,
    - IEEE 1394 Firewire®

    and optional factory options:
    - GPS,
    - Secure Radio,
    - Mil-Std-1553B
- extended battery life for reliable, continuous use during deployed operations
- support for Microsoft® WindowsXP®, XP Embedded and BlueCat and Redhat Linux
- hot-swappable battery support,
- compatible with BA5590 and BA2590 Batteries,
- rechargeable Li-Ion, Li-Polymer, and Li-Cobalt rechargeable smart batteries option
- options for solid state and shock-resistant, extended temperature rotating media drives

single or dual RGB QXGA and NTSC, PAL, RS-170/A, and S-Video output configurations.

3.2 Head Mounted Display

3.2.1 Background

A head (or helmet) mounted display (HMD) is a head-worn video display device, commonly used for Virtual or Augmented Reality applications. Recent technological advances (including the introduction of OLED microdisplays[13]) are contributing to the ongoing

[13] OLED - organic light-emitting diodes, based on light emissive properties of certain organic compounds. As opposed to liquid crystal based microdisplays used in some HMDs, requiring

transformation of HMDs from the heavy, uncomfortable and prohibitively priced contraptions of several years ago, which offered poor image quality at low resolutions, to the increasingly affordable, ergonomic and portable devices available on the market today, illustrated by offerings from companies such as eMagin (Z800 3D Visor), i-O Display Systems (i-Glasses PC SVGA) or Microvision (Nomad Display) and Cybermind (Visette45 SXGA) High-End Displays (see Figures 3.3 and 3.4.).

a) i-Glasses PC SVGA b) Z800 3D Visor c) Visette45 SXGA

Figure 3.3. Examples of binocular Head Mounted Displays.

3.2.2 Prototype Requirements

In general, Head Mounted Displays are a wide category of display devices encompassing varied techniques of presenting the image to the user. There are many technical, ergonomic and perceptual considerations when selecting the best display for a given application.

For the developed proof-of-concept prototype, the most important factors considered were the price of the HMD device, the display's field of view and the screen size/resolution match with users visual acuity (for optimal image quality). As of January 2005 when the 'consumer-priced' market section was examined, the I-glasses PC-SVGA Pro 3D was (in author's opinion) the best available display in that price range, and so it was selected for the prototype

---

separate light source, OLED microdisplays emit their own light, offering decreased power consumption and better image color and brightness.

work. The I-glasses technical specifications are provided in Table 3.2. As of the time of writing (second quarter of 2006) many new products are now available which offer potentially competitive power/price ratio, for example the eMagin Z800 3D Visor or the near-eye display devices described in Section 3.2.4.

3.2.3 I-glasses PC-SVGA Pro 3D Specifications

Display Resolution	800x600
Field of View	26 Degrees Diagonal
Virtual Image Size	70" at 13'
Color Depth	24 Bit Input
Inter Pupillary Distance (IPD) Adjustments	None Required
Eye Relief:[14]	25mm
Exit Pupil:[15]	17mmH x 6mmV
Convergence:	7' 10", 100% Overlap, TBR
VGA / SVGA / XVGA Input:	Scaled to SVGA (800 x 600)
Refresh Rate:	Flicker Free 100hz
Weight:	7 Ounces
Power Supply:	100 - 240V AC Power Cube

Table 3.2. ioDisplay I-glasses Technical Specifications

[14] "The **eye relief** of a telescope, a microscope, or binoculars is the distance from the eyepiece the eye can be placed at which the exit pupil is approximately the same size as the eye's pupil." [Wikipedia]

[15] "In optics, the **exit pupil** is a virtual aperture in an optical system. Only rays which pass through this virtual aperture can exit the system. The exit pupil is the image of the aperture stop in the optics that follow it. The same term is used to refer to both the virtual aperture and its diameter." [Wikipedia]

3.2.4 New Form Factors in HMD Design: Near-Eye Displays.

With the majority of Head Mounted Displays, there is an inherent problem of user's diminished situational awareness when wearing the display. Even in see-through (optical or video) HMD configurations, such as the one used in the prototype, occlusion of substantial areas of the visual field usually occurs. Peripheral areas of human visual field are crucial for situational awareness – although the visual acuity in these areas is low, the motion sensitivity is relatively high (due to dominance of rod light sensors in the periphery of the retina), providing important cues regarding motion in the environment. In typical Head Mounted Displays the periphery of vision is often occluded by the device casing, making it more difficult for the system's user to move around without walking into objects in the environment. Aside from the visual field occlusion, the typical HMD device size and weight is detrimental to user's comfort. The cumbersome appearance of the device goes a long way against widespread acceptance of HMDs for use in everyday life situations. (This is somewhat less of an issue for military or first response applications where additional gear is usually required). New generations of display devices attempt to address these shortcomings.

The motivation behind the particular approach described here can be traced back to Minolta's prototype forgettable near-eye display (based on a holographic element embedded in the eyeglasses lens) described in [Kasai00]. The device's form factor and weight is very similar to standard reading glasses and does not interfere with user's normal routine while giving no indication to external observers that the display is on, hence the "forgettable" moniker. There are several technological approaches possible to achieve that effect. Available for several years now, the Microvision Nomad[16] displays paint monochromatic image (800x600 resolution) directly on

[16] [www.microvision.com]

the user's retina using a low-power laser. Kopin Microdisplays[17] recently announced high resolution postage-stamp-sized displays (up to 1,280x1,024 resolution) which are to become available in the third quarter of 2006. Lumus[18] recently introduced its LOE (light-guide optical element) technology, in which a thin transparent lens which collects light from a tiny projector on the side of the lens is used.

Consumer priced Personal Displays based on these companies offerings are already available, as exemplified by some of the exhibits at the recent Society of Information Display International Symposium expo (June 2006). Two products based on Kopin technology are the MyVu "personal media viewer"[19] (see Fig 3.4c) offering QVGA resolution (320x240) designed for use with the popular Video iPods and the Icuiti DV920[20] (Fig 3.4b) offering 640x480 resolution. Lumus introduced its PD-20 Series (Fig 3.4d).

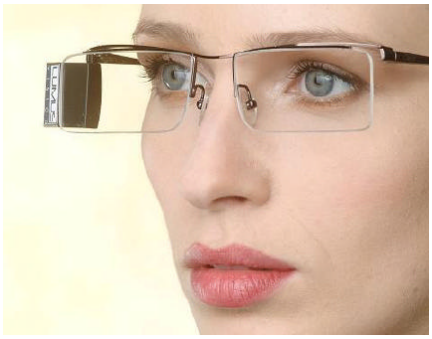
The ergonomics offered by the above devices are a significant improvement over previous generations of bulky HMDs: they are relatively unobtrusive, lightweight and have reduced impact on wearer's situational awareness. The increasing visibility and variety of such displays in the consumer priced market range can be seen as a sign of the maturing of the HMD technology. Although the image resolutions offered by most of these near eye displays are lower than that of the I-glasses display used in the implementation, the ergonomic advantages should warrant evaluation of the near-eye displays in the first response application context.

[17] [www.kopin.com]
[18] [www.lumusvision.com]
[19] [www.myvu.com]
[20] [www.icuiti.com]

a) Microvision Nomad display

b) Icuiti DV920 display

c) MicroOptics MyVu display



d) Lumus PD-21 display

e) Icuiti M920-CF. (Weight: 3.5 ounces.)

Figure 3.4. Near-Eye Displays.

3.3 Head Mounted Camera

The operation of the video-see-through AR device described here relies on providing the user with a digitally altered view of the environment. Since the view should be as consistent with the original (unobstructed with the HMD) view as possible, in the prototype a video camera was

mounted in front of the Head Mounted Display. The camera used was a Unibrain Fire-I IEEE1394 camera, two other camera hardware options were also examined. Two of the examined cameras were essentially good quality webcams, the third one was a miniature analog surveillance camera used together with an analog-digital video signal converter. The following sections describe the findings.

3.3.1 Unibrain Fire-i IEEE 1396 Camera

3.3.1.1 Fire-i Camera Features

Unibrain's[21] Fire-i camera is a CCD (Charge Coupled Device) sensor based camera is marketed for home and office use. However, the good quality of image it produces, fast framerates, low power consumption, and convenient form factor make this camera a frequent choice for augmented reality, as shown by examples of systems described in [Vlahakis03] or [Siegl03], machine vision and robotics systems. The camera is pictured on Fig. 3.5, technical specifications are contained in Table 3.3.

The camera can be powered directly from the 6-pin FireWire connector, or using an additional jack input that can accept any 8 - 30V DC power adapter. Unfortunately in most mobile computers equipped with FireWire connectors, including the Amilo-D 1840, the connector is of the 4-pin variety and does not provide power to the FireWire device. In order to interface with the notebook computer, a Newnex power injecting 4-to-6 pin FireWire

[21] [www.unibrain.com]

connector cable[22] was used (see Fig. 3.6). The cable, besides acting as a 4 to 6 pin adaptor, provides an additional receptacle for DC power. The receptacle is located near the 4-pin connector end of the cable, which allows to avoid placing an additional length of cable between the head mounted camera and the mobile device worn by the user. In the setup used for prototype demonstration a mains power adaptor was used to provide power to the camera through the Newnex cable receptacle.

Figure 3.5. Unibrain Fire-I camera. Bottom: daisy-chaining of the Fire-i cameras.
Source: [www.unibrain.com].

Figure 3.6. Newnex Firewire 4 to 6 adaptor cable with power injection.
Source: [www.newnex.com]

The lightweight design (weight is approx. 2 oz.) and small dimensions make the Fire-I quite suitable for a head mounted camera application. The camera's flat shape allowed it to be mounted on the front face of the HMD without problems, allowing to align the camera'a optical axis with the general direction of user's gaze.

The camera's average typical power consumption of 0.9 Watt is relatively low and well-suited for a mobile application. The unavailability of direct power supply from the notebook computer is not a significant drawback, since regardless of the computing platform in future

[22] Newnex iPod™ Firewire Power Cable. Part number: CFA-64P. [www.newnex.com]

iterations of the prototype, an external battery pack is likely to be provided for the head mounted display – such power source could be used to power up the Fire-I camera as well. Alternatively the camera could be customized to utilize the computer's USB port.

Additional features include the ability to daisy-chain up to 32 Fire-i cameras thanks to a second pass-through port provided on each camera. If dual head mounted cameras in a stereoscopic setup were ever to be used in the device, this feature would greatly simplify integration of the two cameras with the Head Mounted Display, reducing the number of connectors and cables needed.

Optional 2.1mm wide-angle lens compliant with the M12x0.5mm standard used in the Fire-i are available in Unibrain's Digital Camera Wide-Lens Kit. (Other focal lengths available include 8mm medium telephoto lens and 12mm telephoto lens). Used in conjunction with a wide visual angle Head Mounted Display, such wide angle lens may be used to provide the user with a wider angle of view on the surroundings. In such case, however, the effects of lens distortion on the marker detection algorithm effectiveness would need to be examined.

3.3.1.2 Performance

The camera delivers sharp and clear images, and very fast frame rates, allowing to capture VGA resolution video (640x480 pixels) at the rate of 30 frames per second (capture was performed using a third-party application). The video stream is provided to the computer at 400 Mbps through the IEEE 1394a port. The quality of optics and the CCD sensor is very good in our experience – edge sharpness is excellent, which is important both for planar marker recognition as well as for the system's end-user experience. The automatic gain, shutter and balance mechanisms proved adequate over a range of lighting conditions. With lens focus manually set to infinity, the depth of field was sufficient for the application.

3.3.1.3 Problem Encountered: OpenCV Interoperability

A problem of software interoperability between the camera's driver and the OpenCV library was encountered, resulting in the unavailability of Fire-i video capture resolutions greater than 160x120 for the purposes of the prototype. While this low resolution setting still allows for correct functioning of the prototype, it effectively reduces the maximum viewing distance (or increases the minimum physical size) of the fiducial marker needed for successful detection. Low resolution of the captured video frames results in noticeable blurring of the imagery viewed by the user, as texture filtering is applied to the camera capture texture in every frame.

Several unsuccessful attempts to address the problem were made, including testing with alternative software drivers for the camera, applying a patch to the OpenCV's camera capture function (CvCapture) code, and changing the cameras default resolution using third party software. The problem is not specific to the particular camera hardware, as it was observed on two tested units, and reported on online OpenCV forums. An incompatibility problem with the OpenCV code is the most likely cause, since video resolution setting and capture provided by OpenCV's alternative CvCam library does not cause any problems. Availability of a commercial Unibrain Fire-I Software Development Kit, and the open-source philosophy behind the OpenCV library allow the author to remain hopeful the issue can be resolved with additional effort, either though using Unibrain's SDK to perform the capture or by modifying the video capture code of the OpenCV library.

Fire-i™ Digital Camera Specifications	
Interface	IEEE-1394a (FireWire) 400 Mbps, 2 ports (6 pins)
Camera Type	IIDC-1394 Digital Camera, V1.04 Specification compliant
Sensor Type	Sony™ Wfine* ¼" CCD Color, progressive, square pixels
Resolution	VGA 640 x 480
Optics	Built-in f 4.65 mm lens, anti-reflective coating
Video Modes	YUV (4:1:1, 4:2:2, 4:4:4), RGB-24bit, Monochrome-8bit
Frame Rates	30, 15, 7.5 and 3.75 frames per second
Gain	Automatic or Manual Control 0-30 dB
Shutter	Automatic or Manual Control 1/3400s-1/31s
Gamma	ON / OFF (visual use / image processing use)
White Balance	Automatic or Manual Control
Color Saturation	Adjustable
Backlight Compensation	6 modes + OFF
Sharpness	Adjustable
Special Features	Software sleep mode, Color bar generator
Power Supply	8 to 30 VDC, by 1394 bus or external jack input Consumption 1W max, 0.9 W typical, 0.4 W sleep mode
Dimensions (WxHxD)	62 x 62 x 35 mm
Housing & Weight	Semi-transparent polymer, 60 gr
Included Accessories	Spring clip, tripod adaptor, mini-tripod, 2m FireWire cable
Included Software	Fire-i™ Software

Table 3.3. Fire-i™ Digital Camera Specifications.
Source [www.unibrain.com]

3.3.2 Logitech QuickCam Pro 4000 USB2 Camera

The Logitech QuickCam Pro 4000 (pictured Fig. 3.7) was used as an alternative video source during the development of the prototype. This CCD sensor based webcam class camera captures at the maximum resolution of 640 x 480 pixels (VGA) with 30 fps framerate. The interface used is USB2.0, power is provided to the camera from the computer's USB port.

The camera allowed for testing the software application at the VGA resolution, which was unavailable with the Unibrain camera due to software related issues already discussed. The camera performed well with the fiducial marker detection algorithm, and the subjective quality of the video presented to the user was in author's experience very good compared to other selected webcams within that price range (below 100$), due to the CCD sensor and effective automatic gain and shutter mechanisms. In video capture performed using third-party applications, the Logitech performed well, but the Unibrain Fire-I camera's output in the same lighting conditions was (in authors subjective opinion) visually sharper, with less motion blurring, which can be possibly attributed to faster transfer rates of the FireWire interface, or to better responsiveness of the Fire-I's CCD sensor.

Still, the form factor of the Logitech camera arguably prevents it from being a suitable COTS component for the prototype device. Although the camera's stand can be detached, its spherical body (50mm in diameter) and the placement of the connector cable in the rear of the sphere restricts HMD mounting possibilities. Flat-shaped cameras with connectors on the side such as the Fire-I or board cameras show more promise than this webcam designed for typical desktop use.

Figure 3.7. Logitech QuickCam Pro 4000 camera.

Figure 3.8 Belkin Hi-Speed USB2.0 DVD Creator.
Source: [www.belkin.com]

3.3.3 Belkin USB2.0 Digitizer + Miniature Analog CMOS Camera

Another hardware alternative that was briefly considered was a combination of an analog CMOS camera mounted on the HMD and a video digitizer. Potential advantages of such configuration are the ability to use smaller sized[23] analog cameras and more flexibility with prototyping, since any number of analog cameras providing NTSC video signal may be evaluated without any modification of the other system components (camera drivers etc). Miniaturized analog surveillance cameras can be used, which can be potentially easier to integrate with the HMD or helmet.

The digitizing hardware used was a Belkin Hi-Speed USB2.0 DVD Creator external video capture card, pictured on Fig. 3.8.

[23] The need to provide analog-to-digital conversion results in increased camera's size - this is true for CCD sensor digital cameras as well as CMOS-sensor digital cameras, although in the latter the A/D converter electronics can be integrated within the same CMOS chip as the light sensor, making this issue less significant.

Video Input:	composite and S-video
Video Capture Size	160x120, 320x240, 640x480
Performance:	30 frames per second at CIF (352x288) and at VGA (640x480)
Power Source and Consumption:	Self powered through USB port, +5 VDC @ 240 mA max
Signal Input Connectors:	Composite Video-Industry Standard RCA jack S-video-industry standard female S-video connector

Table 3.4. Specifications of the Belkin DVD Creator device.
Based on information provided by the manufacturer [www.belkin.com]

3.3.3.1 Performance and Remarks

The above configuration was found to be slightly inferior to the Fire-I and QuickCam options in terms of video capture speed and image quality and contrast (inferior contrast can be possibly attributed to the camera's CMOS sensor quality). Although the video capture device is powered from the USB port, the analog camera requires an additional power source, which complicates deployment similarly as in the Fire-I configuration. The examined analog camera offered wide angle of view, however this advantage can only be fully capitalized on when the head mounted display device has sufficiently wide viewing angle, otherwise the result can be disorienting to the user. The camera is lightweight and small, but it's tube-shaped housing is elongated, calling for camera placement on top of the HMD instead of in front of it. Such placement results in a slight parallax error experienced by the user, as the camera's optical axis is paralell but shifted away from the user's eye optical axis. Concluding, the particular analog camera and A/D converter combination didn't compare well with the other options, however in general such setup using different hardware may still be an interesting option. For example, the

Thermite TVC mobile computer described in section 3.1.2 provides an integrated video capture card. If this platform is used in future iterations of the prototype this feature can be leveraged on, allowing to use a lightweight, small form factor analog camera for the device.

3.4 Glove Input Device

3.4.1 Motivation

In traditional computer interfaces, the action of selecting an object in the GUI is typically performed using a pointing device such as a mouse, trackball, pen, stylus, joystick, touchpad or touchscreen – all of the listed input devices being manually operated. An important requirement for the first-response reference device is hands-free or near hands-free operation, making it possible to use the device while carrying objects or operating other machinery. Therefore, in the proposed system, user's head motions relative to external fiducials are used as the primary method of input instead of manual actions. This was described in detail in section 2.1.2 in Chapter 2. Yet, in order to support a complete range of interactions with the system, it is also necessary to provide an additional input method. For example, to access information pertaining to a fiducial marker, the user needs to activate the content selected using head movements, an action which can be seen as analogous to double-clicking an icon in a traditional GUI. This calls for at minimum one additional switch/button or discrete gesture to be supported by the system. Additionally, it could be argued that functionality incorporated into the system in future (for example teammember communications, headcam video relay or environment annotation) would most likely benefit from a straightforward 'push-button' approach for each of those features, since navigating a complex user interface would be too time consuming and would introduce too much cognitive overhead.

The interaction design for the particular device in question appears to lend itself well to use in conjunction with one or more single degree of freedom (1D) sensor based hardware input devices. Such input devices would allowing the user to quickly provide a value from a range – for

example to selecting the desired visual magnification of a map, the opacity of an overlay, or audio volume. The following sections discuss some of the design considerations examined, and describe the rationale behind the implemented solution as well as technical details.

Wearable computing interaction paradigm calls for unobtrusive input devices which are useable in a wide variety of situations. In general, there is a considerable variety of possible input methods for wearable computing devices, including 'deskless' (off-table) versions of traditional pointing devices, such as the Finger Mouse[24], wearable keyboards, inertial motion sensing devices for gesture recognition. Other approaches include vision based methods involving fiducials, glove controllers (pinch gloves[25] or finger bend sensors), or voice recognition. An example of an AR system which uses a combination of simple custom build pinch gloves and an ARToolkit-based fiducial placed on the glove is described in [Piekarski02],[Piekarski03]. The hands-free operation requirement of the first response application examined here limits the design space considerably. Multiple modality system using voice activation as an input method is a possibility, though the current state of the art in speech recognition is still limited (recognition error rates, throughput of the user-computer communications) and introduces additional challenges (increased cognitive load), as reported in the literature ([Christian00] and others). A voice activated system is likely to create conflicts with voice communications between responders, especially when the system's user is trying to look up information while communicating with another team member.

In the search of appropriate input method a number of alternatives were considered, finally a finger gesture recognition based method was settled for. Although this method does involve the users' hands, it doesn't preclude performing other manual activities at the same time. The envisioned input device is a set of 1D finger bend sensors, which can be easily embedded

[24] [www.aocusa.com]
[25] Pinch gloves detect contact between two or more fingers thanks to embedded electrical circuits (conductive materials may be used). Typically used for manipulating 3D objects ('grabbing'), or triggering actions using predefined gestures (finger 'snap').

into protective gloves worn by the responder. Glove based input devices (often referred to as 'wired gloves'[26]) are nothing new in context of virtual and mixed reality computing, where tracking devices, finger bending sensors and other functionality (pinch sensors, tactile feedback) are often used. Wired gloves are typically used for manipulating virtual objects in 3D in modeling or simulation, or for applications such as robotic control for teleoperation or entertainment industry (puppetry). An early but comprehensive survey of glove based input devices can be found in [Sturman95].

3.4.2 Implementation

In the developed system, a 'stripped down' wired glove was used, sensing in real time the current pose of fingers of one palm. Simple gesture recognition was used to prevent accidental triggering of system commands. Since responders inevitably perform a variety of manual tasks, the gesture chosen had to be unique – easily distinguishable from typically occurring hand movements. A simple gesture which fulfills this requirement is a rapid bending of a finger – for example, an AR object highlighted by the users' action of gaze centering can be now 'activated' by flicking the index finger. The finger movement speed must exceed a predefined threshold in order for the gesture to be register. The direction of finger movement is not important, since different starting positions of the finger (straight finger, fully bent finger) should be supported. The finger velocity threshold used for the gesture detection needs to be carefully selected to ensure robust detection and minimize false positives. Other possibilities for gesture recognition here include using multiple-finger static gestures, multiple finger flicks etc.

Besides being used as a basis for simple gesture detection, in some situations the raw finger bending information from the 1D sensors is used directly in the interface as well, allowing the user to provide values from a range to the system easily. An example would be a 'zooming in'

[26] Equivalent terms in use are 'cyberglove' or 'dataglove', but the terms are trademarks of Sun Microsystems. (Source: Wikipedia).

action through the bending of middle finger (again, a predefined bending threshold may be specified, below which no action is taken) and 'zooming out' by bending the ring finger. Such interactions, being more manually involved, are of an optional enhancement character and cannot be crucial to the basic functionality offered.

While the potential for utilizing sensors for fingers of both hands is there, in this implementation a glove controller was used only with the user's right hand because of the available hardware specifics. In common tasks the non-dominant hand is used to a lesser degree, usually for supplemental sub-tasks typically associated with setting a frame of reference - such as holding in place or repositioning the object which is being manipulated on by the dominant hand. Therefore using the controller with the non-dominant hand could be a good idea, since activation gestures would be potentially less likely to conflict with other manual manipulations.

3.4.2.1 P5 Glove Overview and Specifications

The prototype uses a modified off-the-shelf P5 glove input device manufactured by the (now-defunct) Essential Reality LLC. The P5 is a low-cost device which, though no longer manufactured, is still available from various retailers (approx. 25$). The product, though innovative and well-engineered, was unsuccessful as the game controller / mouse replacement device it was marketed to be, arguably due to human factors issues[27]. The provided finger bending sensing functionality and a software development kit makes it interesting to researchers, given the very low cost and the lack of similar devices on the market within this price range. Senseshapes [Olwal03] is a project utilizing a combination of the P5 glove (also modified) and an Intersense IS900 tracker for precise position tracking. An earlier computer game controller used

[27] In the most simplistic use of the glove as a mouse pointer, the user's hand is moved in a 2D plane in front of the infrared detector tower placed on the desk. Understandably, this quickly leads to fatigue, as the hand needs to be held up in the air for prolonged periods of time.

by many researchers was the much less sophisticated PowerGlove manufactured by Mattel for Nintendo game consoles.

There exists an online community[28] of P5 enthusiasts, who develop software for the P5 based on the original SDK and driver code released into the public domain by Essential Reality, and are experimenting with using the controller for robotic control or as a musical instrument (MIDI controller).

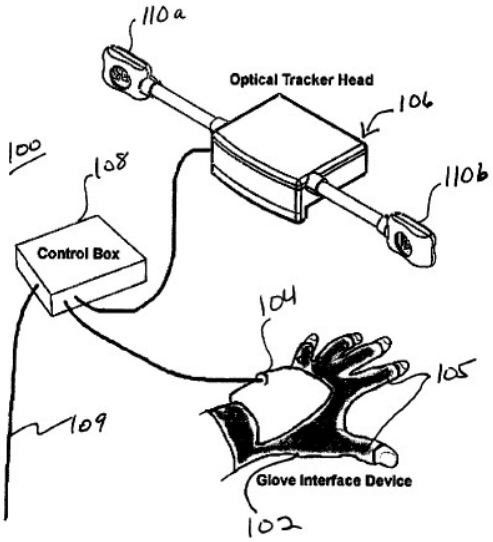


Figure 3.9a. Illustration excerpted from the P5 glove technology patent application[29]. [P5Patent]

Figure 3.9b. The P5 Glove. The hand-worn controller is shown alongside the infrared tracking device (shown off-scale, top right).

The P5 controller (pictured on Figure 3.9) provides six degrees-of-freedom (6DOF) hand pose and position tracking (X,Y,Z,yaw,pitch,roll) and finger bending sensing (five 1DOF sensors). Technical specifications are provided in Table 3.5. The glove's position tracking is

[28] [www.p5glove.pbwiki.com]

[29] [v3.espacenet.com/textdoc?IDX=WO0237466&CY=gb&LG=en&DB=EPODOC]

based on an array of infrared LEDs on the glove controller being sensed by a tracking head (Fig 3.10a, marked as '106'). The tracking head is encased together with control electronics within a 'tower stand' (Fig 3.10b – top right), which interfaces with the computer using a USB link. During normal operation of the P5 device the tower needs to be placed on the desktop in front of the glove controller, as there is a line of sight requirement between the LEDs and the infrared detectors. The finger bending is sensed using one electro-resistive element for each of the fingers.

Four additional buttons on the glove body are available to software implementors, intended for supplemental actions such as glove calibration or temporary suspending of position tracking when the glove needs to be repositioned in space (a variant of 'clutching', an action associated with relative position input devices. Picking up and repositioning the mouse when one runs out of desk space is a typical form of 'clutching').

In author's experience with the P5 glove, the 6DOF spatial tracking worked reliably only within a limited space volume, and great care was needed in setting up the operating environment, as reflections of the infrared radiation from nearby objects interfered with the precision of the tracking considerably. Finger-bend sensing was reliable, though an initial calibration step was necessary in order to prevent strong sensor biases. The calibration is performed using the provided driver software and involves making a fist with the hand's fingers, pressing a button on the glove and then completely straightening the fingers.

The necessity of line of sight with the stationary desktop tower makes an unmodified P5 device unsuitable for a mobile application such as an AR system. The specifics of the device's firmware make it impossible to track finger bends when the glove is 'out of range' of the infrared tower.

Finger Sensor Specifications:	5 independent finger measurements 0.5 degree resolution (0-90 degree range) 60 Hz refresh rate
Tracking System Specifications:	3-4 foot range from receptor 45 Hz refresh rate 6 degrees of freedom (yaw/pitch/roll/x/y/z) XYZ - 0.125 inch resolution @ 3 foot range from receptor XYZ - 0.5 inch accuracy @ 3 foot range from receptor
USB Specification:	USB 1.1 compliant HID specification compliant
Weight:	4.5 ounces

Table 3.5. Essential Reality P5 Glove – technical specifications.

This means that a straightforward repackaging of the tower electronics to make the device more portable is not possible without additional modifications to the device. In the Senseshapes project, the problem was solved with support from the manufacturer, and the firmware programming of the controller was replaced with a version without that limitation. Because of the lack of manufacturer support, technical and time restrictions, a different workaround was used here, which proved to be effective, although it is of a 'temporary solution' nature. The modification made to the glove is described in the following section.

3.4.2.2 Necessary modifications to the P5 glove.

The device enters the 'out of range' state when the infrared radiation from the glove's LEDs is not detected by the detector heads. In this state, beside the unavailability of 3-space tracking data, no finger bend data is sent to the computer, due to particulars of the controller programming. The infrared light output of the LEDs placed around the glove body is modulated individually using a specific frequency, allowing the detector to track individual LEDs (necessary for robust hand pose registration) and to filter out external infrared sources. Although the patent information hints at the technique above, no documentation is available for the P5 glove

disclosing the specifics of the modulation used. By measurements of the glove's LED voltage waveform and trial and error methods the frequency and duty cycle for a square waveform was identified which, when supplied to an external infrared light emitter in view of the detector heads, imitates the presence of the glove in the P5 infrared tower's line of sight.

A simple battery-powered electronic circuit was constructed based on a 555 astable integrated circuit and an infrared LED. It was packaged together with the detector head and controller circuitry into a smaller, more portable chassis in a manner allowing for the detector heads to be constantly illuminated by the LED. As a result, the modified glove controller device is permanently in the 'tracking' state, however it no longer provides meaningful position sensing information. On the other hand, its finger bend sensors may now be successfully used in a mobile context.

3.4.2.3 Gestures used: Ergonomics

An important human factors consideration when designing the glove input functionality is the 'separability' of the 1DOF finger bend sensors. A naive approach in designing the interface would be to assign independent gestures to each of the five fingers - however, the specifics of human hand musculature and neuromuscular control limit the usefulness of such design.

The wide range of movements and tasks performable with a human hand is made possible by a complex system of muscles, responsible for the variety of extension, flexing, adduction and abduction joint movements of the digits and wrist. In medical literature the muscles are grouped according to the placement of muscle origin and 'belly':

- extrinsic – muscle originates outside of the hand
- intrinsic – muscle origin and belly within the hand

or according to their function:

- extensors/supinators – straightening out of fingers

- flexors/pronators – bending of fingers

or according to the digit the movement of which they affect:

- thenar – movement of the thumb
- hypothenar – movement of the little finger
- short finger muscles – movement of fingers 2-5 (index-little finger)

(source [http://www.dartmouth.edu/~anatomy])

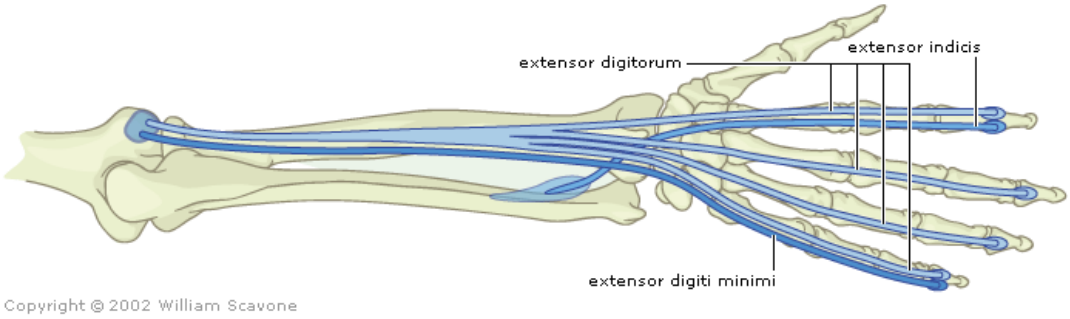


Figure 3.10. Muscles of the posterior compartment that act on the fingers (digits 2-5). Source: [http://www.dartmouth.edu/~anatomy] website

As already stated, the finger flicking motion was selected as a characteristic gesture to be used with the system. The next stage involved determining which fingers are best suited to be used with such gesture recognition scheme. There are two main cases of interest here, in one of them the palm of the hand is not involved in another task at the same time, in the other the responder is using the hand i.e. grasping an object (axe, ladder rung, compressed air bottle, other equipment) or performing other actions such as pushing and pulling. The following suggestions should be verified by a usability experiment on human subjects involving a variety of real-world tasks once the prototype iterations reach the necessary maturity.

The thumb has a specialized role in human hand, it is opposable to the rest of the fingers in an evolutionary adaptation resulting from the freeing of the hand from walking requirements [Wikipedia]. As such, it makes precise grasping actions possible, usually working in accord with the index finger (writing, picking up a small object). Thus, although the thumb is mobile and

quite capable of precise and fast movements, using it for interface-critical gestures may not be a good design decision. A responder holding a fire hose, or an axe handle cannot easily utilize the thumb for anything else, as it cannot be easily substituted with another digit).

The sensor used with little finger, in the author's experiences with the glove, proved difficult to activate without affecting the other finger sensors. Whether it is caused by biomechanic factors (muscles involved, their placement and insertion point), or ones related to motoric functions in the brain, both the agility and the little finger's movement separability seemed inferior to that of the thumb, index finger and middle finger. It is possible that this observation will not hold for other subjects in general, and trained or innate manual agility is the key factor here. The finger's relatively small length also complicates the design of the finger sensor due to smaller absolute displacements involved. Because of this, some challenges are to be expected in incorporating a finger bend sensor for this digit into a protective rubber glove as it may be necessary to provide adjustment for a variety in user hand sizes. Because of the above, the use of the little finger in the gestural interface is not recommended. If the P5 glove's little finger sensor is to be used to detect finger flicks regardless of this recommendation, it would be suggested to adjust (reduce) the threshold speed for gesture detection as compared to other fingers.

The index, middle and ring fingers are good candidates for rapid flexing gestures. According to author's observations, the ring finger proved slightly but noticeably inferior in mobility/separability. There is however a more serious problem affecting the middle and ring fingers, which is related to the fact that the long extensor muscles (extensor digitorum – see Fig 3.10) driving their straightening-out movement share a common muscle belly. Said muscle group affects all four fingers excluding the thumb. While the index and little finger have the benefit of additional extensor muscles (extensor indicis and extensor digiti minimi) which allow for individualized extending movement, the ring and middle finger are much less separable in their extension movement. When one of the two fingers is held down, the other is very difficult to lift

up. This may lead to problems in some situations, such as when the hand is grabbing an object (all fingers bent), and one of the fingers needs to be lifted in order to perform the flicking gesture.

3.4.2.4 Final Finger Gesture Assignment

Based on the rationale presented in in previous sections, fingers and gestures were selected for the prototype's interface which are listed in Table 3.6.

Finger	Action	Result	Rationale
Index finger:	Rapid finger flick	Activation/deactivation of content (depending on situation).	easy activation intuitive gesture
Middle finger:	Finger bend exceeds a predefined value (threshold)	The difference between actual finger bend value and the threshold is provided to the system as scalar input. The middle and ring finger are meant to have complementary effect, for example "rotate object right" with middle finger (speed depending on the resulting scalar value), "rotate left" with the ring finger. Used for less often needed functionality, difficult to perform when grabbing objects.	ease of sensor incorporation complementary action mirrors the natural 'toggling' action between the two fingers threshold prevents unintended triggering
Ring finger:			

Table 3.6. Finger gestures used with the prototype.

3.5 Data Communications Infrastructure for the First Response AR Device

3.5.1 Limitations of 802.11 Networking Hardware

At the current stage of the prototype development, wireless connectivity between mobile responder devices and the central database is provided using NICs (Network Interface Cards) supporting the popular and widespread Wi-Fi (IEEE 802.11) networking protocol. Since the Amilo-D 1840 computer does not provide an integrated Wi-Fi NIC, a PCIMCIA extension card

(D-Link Wireless-G WPC54G ver. 1.2) was used, allowing for testing of networked operation of the system. The card supports the IEEE 802.11 standard operating in the unlicensed 2.4 GHz radio band. Both 802.11b standard is supported (maximum throughput 11Mb/s, with 5.5Mb/s effective throughput offered to the user due to protocol overheads), as well as 802.11g offering 54 Mb/s throughput when all devices in the network support 802.11g.

This technology, used as-is, unfortunately does not satisfy the requirements of the envisioned system as it requires all participating devices to be within the range of a central infrastructure node (in 'infrastructure' mode) or within each other's range (in ad-hoc mode). A more in-depth discussion of the effectiveness of IEEE 802.11 technology in described first response context can be found in Section 3.5.2.2.

An ideal networking technology for the first response system would have to be autonomous, infrastructure independent and offering robust and wide coverage in a wide range of indoor deployment environments. While emerging mobile telecommunications technologies such as third-generation (3G) cellular telephone networks already offer data transfer rates which in near future could potentially allow for supporting a video-enhanced first response application used by multiple responder teams, the availability of such services is still limited to selected metropolitan areas. The strength of signal received within buildings can vary widely depending on radio propagation properties of the environment, and in the event of a bigger cataclysm such as an earthquake or a flood, damage to the local cell tower would render the system useless, therefore the need for autonomy has to be stressed.

Mobile Ad-Hoc Networks (MANET[30]) are an emerging technological approach showing much promise with regard to reliability and robustness within the first response context. An

[30] "A **mobile ad hoc network** (MANET) is an autonomous system of mobile routers (and associated hosts) connected by wireless links--the union of which form an arbitrary graph. The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a standalone fashion, or may be connected to the larger Internet." Source: [Baker02]

overview of the characteristics of MANETs (often referred to as ‘mesh networks’) and related issues related to their design can be found in [Frodigh00] and [Baker02]. The next section contains a discussion of the applicability of mesh networks for the designed first response device and points out a direction for further development.

3.5.2 Mobile Ad-Hoc Networks

3.5.2.1 Background Information

MANET ad-hoc, peer-to-peer network architectures offer dynamic self-configuration of routes between nodes. Such approach allows for minimizing necessary set-up configuration during emergency deployment and for extending the wireless network’s coverage in areas with limited radio frequency propagation. As seen on Figure 4.23, this allows for full connectivity in radio environments with obstacles, as certain network nodes can relay network traffic between other nodes which would otherwise be out of each other’s range (“multi-hopping”). For infrastructure-based wireless networks such conditions would require deliberately placing wireless signal repeaters to extend the spatial extent of the wireless communications medium.

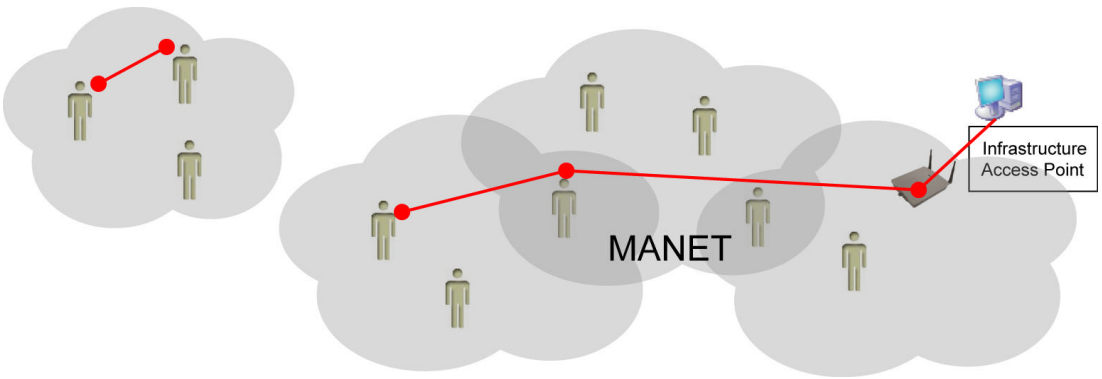


Figure 3.11. Automatic self-organisation and routing in a Wireless Mobile Ad-Hoc Network. Grey clouds represent individual radio propagation domains. **Left:** communications are possible within separated domains of the network.

Mobile computing is power-consumption sensitive, since mobile units typically operate on battery power. It can be argued that multi-hopping can reduce the overall energy which needs to be expended on transmissions, making multi-hop MANETs more energy efficient than single-hop networking topologies. Data rates can be kept high even over long distances without increase in energy expenditure per individual nodes, as low transmit power can be used for each of the individual 'hops' making up the total path (although a tradeoff with total transmission latency is to expected). Lower transmit power can also mean smaller interference and better overall spectral reuse density. (Based on the description of Motorola's Mesh Enabled Architecture at [www.motorola.com]).

Another advantage of using a MANET for a first response system is the possibility of information updates and other communications between nodes within a cluster separated from the rest of the network. Even in situations when the central database servers are not within reach, position tracking information could be still exchanged between nodes which are within each other's wireless connectivity range (see left side of Figure 3.11). Such distributed database behaviour would, for example, allow first responders trapped within a collapsed basement to locate each other, increasing their chances for survival together.

The use of MANETS for emergency response was examined as one of the focus areas of the Distributed Testbed for First Responders (DTFR) effort undertaken by the Building and Fire Research Laboratory of the National Institute of Standards and Technology with the cooperation of other NIST laboratories. In that project (see [NIST_DTFR1] and [NIST_DTFR2]), commercial off-the shelf hardware, including PDA devices, tablets and laptops, was integrated into a mesh network providing responders with multimedia communications capability. In July 2003 the deliverables of DTFR were demonstrated to a number of fire/police chiefs from across the country, and the feedback was generally positive.

3.5.2.2 IEEE802.11 and MANETs

The popular “WiFi” 802.11 networking standard includes an ‘ad-hoc’ Distributed Coordination Function (DCF) mode alongside the ‘infrastructure’ Point Coordination Function (PCF) mode. The DCF mode is based on a carrier sense multiple access with collision avoidance (CSMA/CA) scheme. The IEEE 802.11 protocol by design operates within the lowest two layers of the OSI Reference Model layers, the physical and the Data Link layers. This allows TCP/IP networks to operate seamlessly over the 802.11 wireless medium, but as a consequence, the routing (typically performed by the IP protocol) taking place in the third (network) layer cannot utilize the node availability information encapsulated within the lower layers. Therefore radio communications are limited only to devices currently in range (‘single-hop’) and no ‘multihop’ routing is possible between nodes.

The IEEE’s ESS Mesh Networking Task Group is working on a (unapproved yet) 802.11s extension (according to a Wikipedia article, it is expected to be ratified by the year 2008), which is expected to address the problem outlined above by utilizing "radio-aware metrics over self-configuring multi-hop topologies.”

Even as of now, however, it is possible to overcome the 802.11 “single-hop” limitation by using a specialized routing protocol on top of 802.11, effectively achieving multi-hop routing between multiple “WiFi” “clouds”. Some of the protocols associated with mesh networks over 802.11 radios are the DSR (Dynamic Source Routing) protocol [Johnson03], AODV (Ad-hoc On Demand Distance Vector) and others. The prototype mesh network of the DTFR project mentioned above was based on 802.11b-equipped hardware and NIST-developed research implementations of network protocols [NIST_DTFR1].

An example of a commercial technology which showcases this approach and which may be considered for integration with the developed system is outlined in the following section.

3.5.2.2.1 PacketHop's TrueMesh Technology

PacketHop[31] provides software-based solutions allowing for creation of an ad-hoc mesh with 802.11 radios, targeted at public safety, government, enterprise and consumer sectors. PacketHop's offering includes the TrueMesh software-based mesh networking technology, and the Aware Communications Suite software integrating distributed (server-less) real-time multimedia communications applications. The obvious advantage of this open-standard based, hardware non-proprietary approach is its compatibility with off-the-shelf 802.11 equipment as well as flexibility and cost savings - since existing infrastructure and mobile devices can be leveraged on. Possibility of integration of the designed first response system with the TrueMesh technology as the underlying transport mechanism should be investigated in more detail as one of the next steps in system development. The availability of an existing communications suite could be considered a potential added value, since such ready-to-use functionality would complement the position tracking and content browsing functionality of the designed device well.

3.5.2.3 Non-802.11 MANETs

Considering the DARPA origins of the MANET concept it shouldn't come as a surprise that some of the most mature MANETs already in operation today are the military JTRS (Joint Tactical Radio System) and NTDR (Near-Term Digital Radio) systems. In the public sector, however, there is also recently growing interest in employing mesh networking in municipal 'last-mile' or 'hot-spot' internet provision scenarios as well as providing connectivity for public safety and public works services. A proprietary wireless ad-hoc networking technology offered by Motorola is described in the following section.

3.5.2.3.1 Motorola's Mesh Enabled Architecture (MEA)

Motorola's MEA technology encompasses a family of mobile broadband radio devices which support instant ad-hoc network formation. MEA radios are available in versions for both

[31] [www.packethop.com]

the unlicensed 2.4GHz and the 4.9GHz public safety band. For example, Motorola's WMC7300 PCMCIA card for use in notebook computers, is pictured on Fig. 3.12.

MEA radio users can leverage Motorola's unique Multi-Hopping® capabilities that turn each MEA-equipped user into a router/repeater. This allows a user's traffic to be propagated through other users' transmitters to reach MOTOMESH access points. As a result, every user makes the network stronger – extending network coverage and creating more data paths through the network.

3.5.2.3.1.1 MEA location tracking

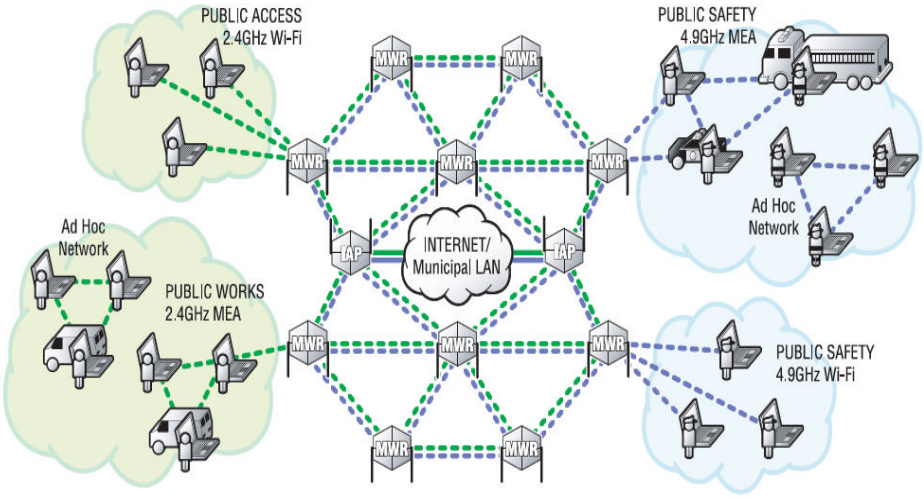
"Non-GPS location and asset tracking" is mentioned as one of the five key benefits of mesh networking in a whitepaper by Motorola [Motorola05]. In fact, all MEA devices can be tracked via Motorola's proprietary Mesh Positioning System (MPS) technology, suitable for use inside buildings and in other types of environment where GPS is unavailable. The position tracking is based on 'time-of-flight' calculations and triangulation, and the results can be provided in geo-referenced coordinates for interoperability with existing GPS and GIS systems.

The precision and robustness of the MPS positioning technology and its potential usefulness for the first response device need to be evaluated in comparison with the optical fiducial scheme proposed in this Thesis. If Motorola's MEA is selected as the wireless networking technology for the first response system described, there is a possibility of utilizing a dual approach to the position tracking, where the fiducial-based tracking information can be fused with the MPS data.

Figure 3.12. Motorola WMC7300 public safety 4.9GHz band MEA radio modem card for a notebook computer.

3.5.2.3.1.2 MOTOMESH – Motorola's Multi-Radio Technology

Although MEA radios cannot directly communicate with 802.11 standard based wireless networking devices and form ad-hoc networks with them, Motorola has recently (second half of 2005) introduced a new network architecture solution, MOTOMESH, which integrates the disparate municipal wireless networking technologies, including 2.4GHz and 4.9GHz MEA and 802.11 networks. Each MOTOMESH Mesh Wireless Router device incorporates 4 transmitters: 2.4GHz WiFi and MEA and 4.9GHz WiFi and MEA radios and provides internal routing and network management, allowing for greater flexibility and availability.



MEA – Mesh Enabled Architecture

MWR- Mesh Wireless Router

IAP – Intelligent Access Point

Figure 3.13. Deployment overview of Motorola's MOTOMESH multi-radio broadband municipal networking solution.
Source: [www.motorola.com]

CHAPTER IV

SOFTWARE IMPLEMENTATION

4.1 Introduction

4.1.1 Development Platform

The prototype system was developed in C++ for the Windows operating system using Microsoft Visual Studio 6.0 as the main development environment The prototype was developed and tested on a mobile computer with Windows XP Home Edition operating system installed. The prototype application developed for the purposes of this project is a multi-threaded executable application utilizing dynamically and statically linked third party libraries. The code contributed by the author integrates the functionality provided by those external libraries into a cohesive whole implementing the envisioned system functionality and behaviors described in the previous chapter and elsewhere throughout this Thesis.

4.1.2 Third-party Software Libraries

The main third-party software components utilized in the project are as follows:

- **Ogre3D**, developed by the OGRE team at [http://www.ogre3d.org], including contributions from the Open Source community (see section 4.3 “Graphical Engine” further in this chapter)

- Mark Fiala's **ARTag** library [Fiala04] (see section 4.4 "Planar Marker System")
- Open Computer Vision Library (**OpenCV**), developed by Intel and the Open Source developers
- **MySQL C language API**, provided by MySQL AB (vendors of the MySQL database server software).
- Hovik Melikyan's **Portable Types** helper library [http://www.melikyan.com/ptypes/]

The above packages are all cross-platform, which makes porting the prototype to other operating systems feasible, should it be required. The libraries can either be built for other operating systems (*nix, Mac OS) from the source code provided, or the developers provide ready to use binaries for other platforms.

The Ogre3D library is available in Linux/Mac versions, which are limited to using OpenGL as the rendering subsystem, as DirectX requires Windows. ARTag Revision 1 is provided in the form of Windows, Linux and Mac binaries, Revision 2 is for Windows only as of the time of writing. OpenCV is also possible to build on Linux and Mac OS Tiger (both for the PPC and Intel processor based architectures), thanks to contributions from the open-source community.

Unibrain, the manufacturer of the Fire-i IEEE1394 cameras provides drivers for Linux and Mac OS. The P5 glove input device has originally been provided with drivers and SDK for Windows only. It is no longer manufactured, however its drivers' source code has been released into the public domain, and has since been ported to Linux and Mac by the community.

However, the overall level of support for the above third-party code appears to be highest for the Windows platform, if criteria such as the maturity of code, completeness of functionality and thoroughness of bug-testing are evaluated. The amount of additional work required to

successfully integrate the libraries on a different platform is difficult to estimate, but it could be considerable given the amount of third-party code being leveraged upon, and diversity of its sources.

Some existing attempts at AR for handhelds, such as projects related to the ARToolkitPlus Library demonstrate that it is possible to create functional systems on embedded/handheld platforms (Fig. 4.1). However, the computational and hardware requirements of the prototype (marker detection, image processing, 3D graphics) mean that a handheld/PDA device implementation would be very challenging to implement given current state of the technology. The functionality would need to be scaled down considerably, and most likely the software would need to be redesigned and created from scratch using handheld-specific set of libraries and/or custom low-level code. Another possibility which could be worth considering is a handheld AR scenario with a high-bandwidth wireless connectivity available, with the handheld device acting as a 'thin client', capturing and sending the camera imagery to a remote processing unit and receiving a stream of computed AR video. While such approach certainly relieves the handheld from computationally intensive marker detection and image synthesis tasks, the responsiveness of such a system would strongly depend on the wireless connection's throughput, latency and reliability.

Figure 4.1 Examples of Handheld Augmented Reality projects.
Source: [http://studierstube.icg.tu-graz.ac.at/]

4.2 Software Architecture

4.3.1 Development Philosophy

Mixed reality software applications share many similarities with real time interactive ("human in the loop") simulations, such as driving or flight simulators. Their main purpose is to provide visualization, respond to user actions, and allow for real time interactivity. Another closely related category of software is computer games, as these often feature advanced simulation of real world phenomena by means of sophisticated physics and artificial intelligence modules. While AR applications do not incorporate such advanced simulation modeling as a rule, the software code flow, typical requirements and challenges associated with them are usually still very similar to those characteristic of a flight training simulator or "first-person-shooter" game development. The majority of 'traditional' software applications, such as those associated with the realms of home and office productivity, are usually based on the notion of a document and document views. Such applications allow the user to create, view or process information in one

form or another, usually using the WIMP[32] desktop metaphor in the Graphical User Interface (GUI). As the user manipulates the data, the corresponding 'views' of the data are refreshed accordingly. In AR, as well as in simulations and games, the GUI is "seamed" with, or becomes, the user's environment, such direct interaction metaphor requires a different approach. The visualization of the simulated environment (or augmentations) needs to be continuously updated to reflect user's actions (change of viewpoint, action performed on the input devices) as well as the passage of time within the simulation. As pointed out in [Singhal99], since the user's actions are not possible to predict in general, the views of the synthetic 'world' cannot be precalculated, and need to be created anew with each frame of the animation. Thus at the heart of most mixed reality systems lies a processing loop ('event loop') in which the system inputs are analyzed, the world-model recalculated accordingly and then rendered, resulting in current graphical image frame. The frames need to be displayed fast enough, and often enough to provide an illusion of smooth movement to the user. The software architecture ramifications of this fact are discussed in the following paragraphs.

4.3.2 Display Latency/Framerate

Human factors studies suggest that the display framerate should be optimally kept at at least 24 frames per second for a smooth simulation. An example target goal of 24 frames per second means that the system needs to finish its complete 'frame' of processing in under 40 milliseconds. Depending on the amount of processing power available, such a goal may or may not be realistic, so implementations of Mixed Reality systems often settle for a lower framerate value. A related and equally important quantity is the system's total end-to-end latency, corresponding to the delay between user's action and the observed result (sometimes referred to as 'transport delay')[Schneidermann98]. In order to prevent user fatigue and disorientation, low

[32] WIMP – abr. for Window, Icon, Mouse, Pointing device, the typical building blocks of graphical user interface

display refresh latency is required. This is not merely a matter of the user's frustration with an unresponsive application and 'sluggish' interaction experience [Wloka95]. Analogous to motion sickness experienced in vehicles, "simulator sickness" is a physiological phenomenon due to the disparity between the visual information observed on the display and the movement/position information from the user's proprioceptive sensors and sense of balance, which can trigger nausea and cause fatigue [Wloka95]. An additional timing-related issue, particularly important for AR video-seethrough systems, is the timing synchronization of synthetic augmentation imagery with the camera-based video stream. Lags resulting from the tracking system latency and other processing can lead to the augmentations 'lagging' behind the real-world images, destroying the intended effect of coexistence of virtual object's with real objects.

All the above considerations combine into a difficult design problem. This calls for architecting the system in such way so that the system latency can be minimized.

4.3.3 Sources of Lag

(based on [Singhal99])

To investigate the main causes of lag in the system, let's consider a very straightforward implementation of a VR or mixed reality system, as described in [Singhal99]. The implementation in question is a single-thread single-process networked VR system. The computing steps are illustrated on Fig 4.2. The initialization phase, usually performed only once at the beginning of runtime, includes a number of setup operations, such as initial database transactions, setting up of network sockets, graphics rendering parameters and objects, initializing input devices etc. After the system has been initialized, it enters a main event loop, cycling through a number of operations repeatedly until the application is terminated. The input device state is read and virtual 'world' state changes are calculated based on this input. In an AR application this stage includes the camera video capture and the fiducial detection, pose, and

position registration. Next, as the network is read, network packets received from other systems are analyzed. Simulation-related computations may then need to be performed, such as collision detection between objects, physics modelling, etc. The state changes resulting from the computations are then sent to the network. Finally, a new frame of the animation needs to be generated by the graphics hardware and displayed, and the loop cycle starts over.

Such sequential approach to a VR/MR application may be sufficient, if the delays accumulated from all the stages of the event loop don't exceed approximately 100ms, after which both the decreased framerate and the delay in perceived imagery become unacceptable.

Unfortunately in the majority of real-world systems this will not be the case, as there are several potential bottlenecks in the discussed event loop sequence. Communication with input devices, whether it's a transmission over a serial port link to and from a tracking device, or a video camera capture in an AR application is typically a multistage process, often including a data processing stage within a controller embedded within the device, and a time-consuming transmission of the input data.

The "Compute State changes from Inputs" phase may be seen as particularly time-intensive in the case of a computational vision based Augmented Reality system, since image processing is performed in order to detect fiducials and extract their poses from the camera capture frames in that step.

The "Computational Modeling" step, not present in simple systems, can be very computationally intensive, for example when scientific visualization, or advanced physical modeling is involved

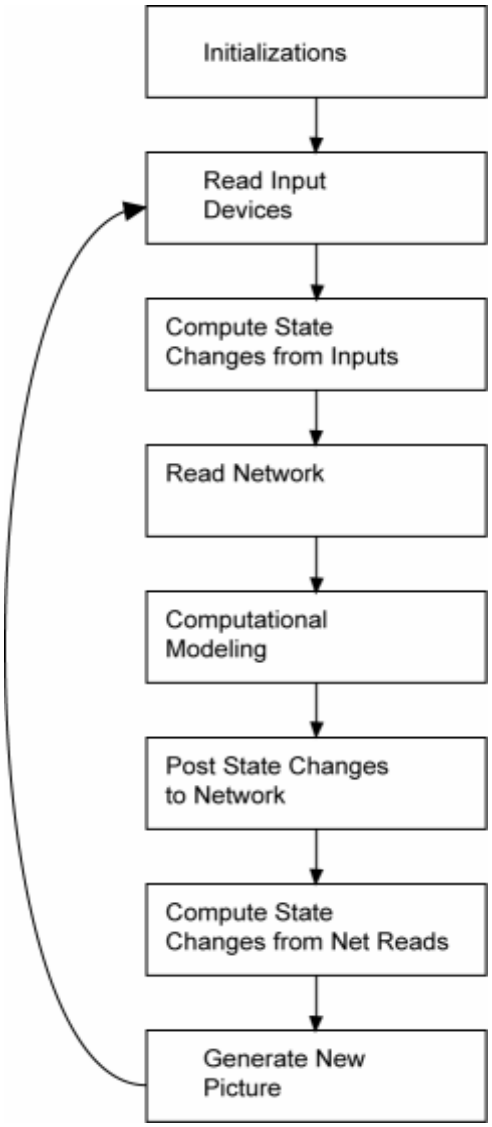


Figure 4.2. Program flow in a single-thread networked Virtual Environment. (Source: [Singhal99])

Another culprit is the "Read network" block, as data needs to travel through the kernel's protocol stack. In a single-threaded application no-wait function calls have to be used, returning immediately if there is no data awaiting. If data is present, it needs to travel up the protocol stack to the application layer, which introduces delay. Sending data to the network in the "Post state changes" block can be just as costly. For simple systems, UDP packets can be sent out without waiting for acknowledgement of successful transmission.

Depending on the level of realism or complexity required by the simulation, the "Generate New Picture" block may or may not have a significant effect on the total event loop cycle length. Graphics processors are currently capable of performing a lion share of the computational tasks related to rendering concurrently with the central processor; however software rendering may still be needed for portable devices which do not possess the necessary accelerating graphics hardware. Typically AR does not require displaying very complex 3D scenes with many objects, as this is the domain of VR, but in general the complexity of the depicted scene corresponding to the number of polygons displayed has a strong effect on rendering time. (To ensure performance, optimization techniques need to be employed, typically

based on effective reduction of the number of polygons which need to be displayed, this is done via polygon culling [Clark76], mesh optimizations and use of multiple Level-Of-Detail meshes).

The total execution time of the above sequence can easily exceed the acceptable delay threshold for a given application and is dependent on many (often unpredictable) outside factors (network congestion when sending, complexity of a particular 3D scene, number of fiducials detected in the scene etc). The existence of blocking function calls further aggravates the problem, as there may be segments in time during which no processing at all is performed while the system is waiting for some event. For example, in one of the first attempts at integrating the video camera footage into the 3D scene, the camera capture was performed before every frame of the animation. The delay involved in the camera image capture reduced the rendering rate to 10 fps from what was 60fps before the inclusion of the video texture. This was not an acceptable result, as one of the goals was to preserve the maximum possible overall rendering update frequency. The answer to the problem was to introduce elements of concurrent processing into the software design. Instead of performing video frame capture at the start of rendering of each animation frame, a separate programmatic thread was created, in which the camera is queried in a continuous loop at the rate allowed by the devices throughput. This way, even though the video texture in the prototype is updated at a relatively slow rate, the global fps rate isn't reduced significantly.

4.3.4 Multiple Thread Software Architecture

[based on Singhal99]

Although the overall speed of a simple AR/VR system with a sequential processing loop can be increased in a trivial fashion by using more powerful hardware, this still does not eliminate the blocking delays related to the I/O device or network. A better solution is to employ a multithreaded architecture [Singhal99]. Concurrent processing makes it much easier to reduce the lag to levels acceptable within the human factors requirements. The generalized application

described in section 4.3.3. is redeveloped into a multithreaded application as depicted in Fig. 4.3. The operation blocks were grouped into partitions, or subsystems, which need to be processed sequentially and whose partitions are put within separate programmatic threads, communicating with each other by means of shared memory. While care does need to be taken to balance resource usage between the threads, such an architecture allows the overall system lag to be reduced and the frame rate can be increased and kept at an approximately constant value regardless of external conditions. Blocking function calls may be now used without the danger that an unusual condition such as network congestion or failure will slow down the processing of other subsystems.

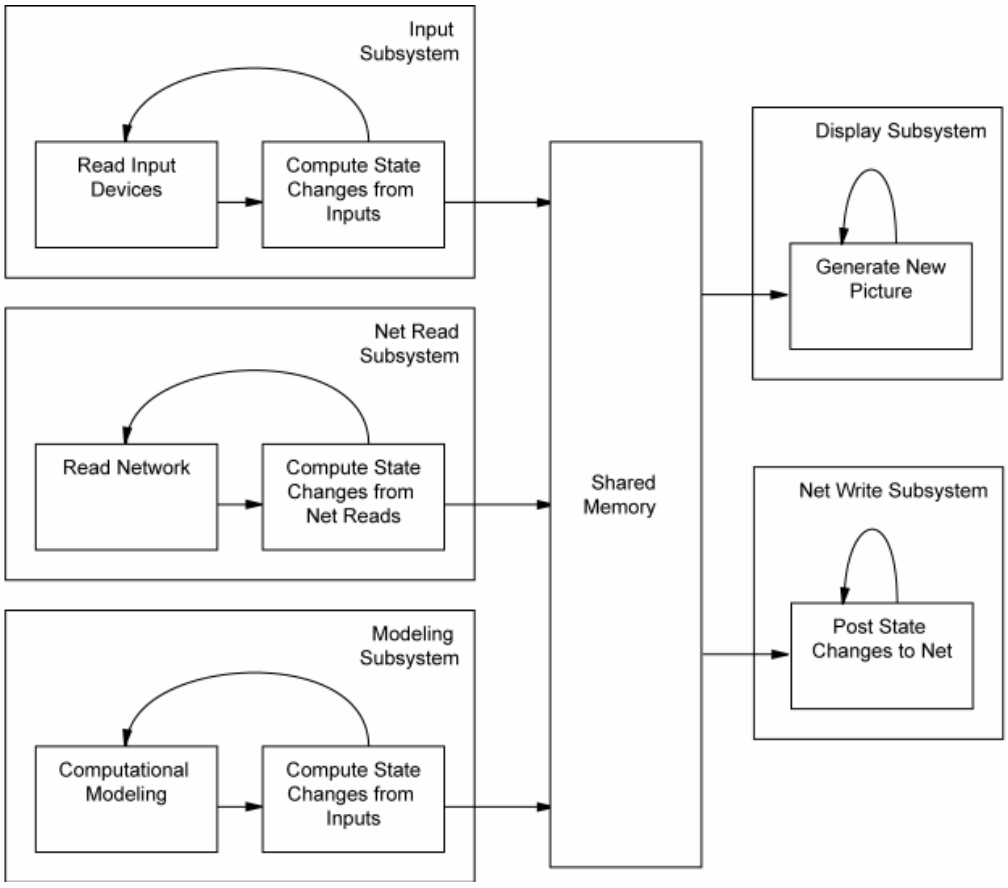


Figure 4.3. Multiple-threaded virtual reality system.
After [Singhal99].

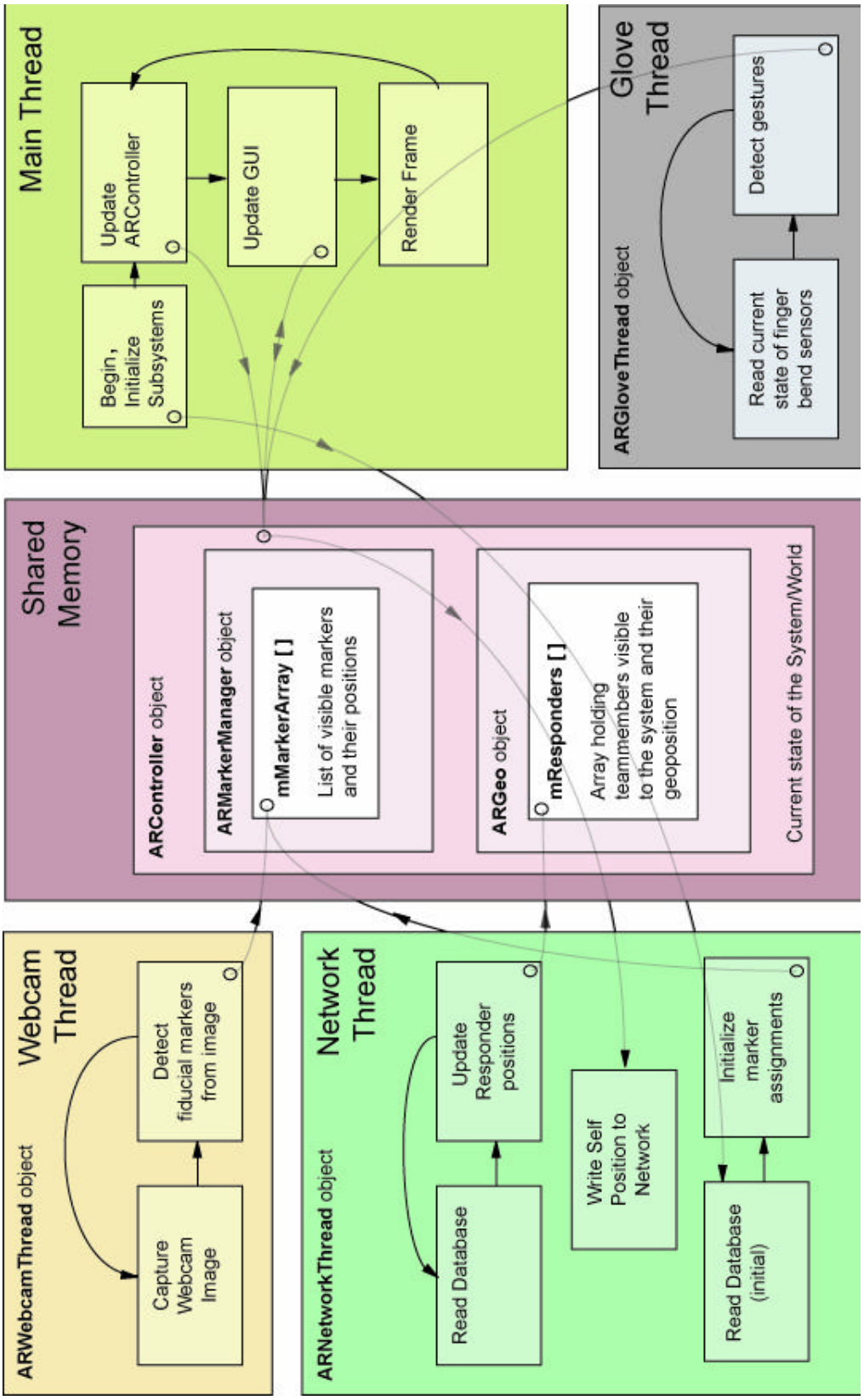


Figure 4.4. Multithreading in the prototype's software architecture

4.3.5 Multithreading in the Prototype

In the executable implementing the prototype application, the overall processing burden is divided among several 'concurrent' threads, as pictured on Figure 4.4. An overview of those threads and their integration is discussed, followed by the description of the OOP class structure and functionality of code contributed by the author.

The development of code implementing the programmatic threads leverages on the Portable Types (ptypes) helper library, which provided an OOP wrapper allowing to create each thread as an object derived from the ptypes class pt::thread.

The Webcam thread handles the image capture using OpenCV's *cvQueryFrame* blocking function. When the captured video frame buffer is available for processing, it is forwarded to ARTag planar marker recognition routine *artag_find_marker*.

The Glove thread starts by waiting for P5 device detection. Once the glove is detected in the system, the thread activates the glove indicator, and begins querying the fingerbending sensors in predefined time intervals. Simple gesture recognition is performed based on sensor readout delta and internal timer. Additionally, the thread updates the 3D scene object representing the glove indicator in the GUI directly, to save some processing. To save resources, the thread is "relaxed" (put into idle state) for a predefined interval of time.

The Network thread contains MySQL database query handling routines. Since all the SQL function calls are blocking (stop the flow of execution), it was logical to separate them from the main rendering thread. At initialization, the locally used marker array containing fiducial descriptors is populated with entries from a remote database table, while other initialization tasks can be performed within the main thread. After that, the database is cyclically queried for the positions of responders. When the information about new self position is acquired based on the currently visible fiducials, the network thread sends an UPDATE query to the database. Occasional lack of database connectivity affects only the teammember positioning functionality,

processing in other threads continues unaffected. Thread idling is used to conserve processor cycles.

4.3.6 MVC Design Pattern

A generalized Model-View-Controller design pattern, as described in [Buschmann96] is widely considered an appropriate metaphor for any GUI interface. Elements of this approach can be seen in the implementation of the handling of fiducial markers detected in the camera image and corresponding labels displayed on screen, which the user interacts with using the glove gestures ("Model" would correspond here to ARMarker class, "View" to ARLabel and ARObject class, and "Controller" to ARController class). A more strict separation of code into Model, View and Controller blocks would not be appropriate here, since the overhead resulting from variable passing might outweigh potential gains in program structure clarity/reusability. One of the programming style rules decided on at the beginning of the development of the software code was to minimize the repeated passing of parameters through multiple layers of nested code, and to prevent unnecessary data duplication, when possible. Such approach doesn't necessarily correspond well with the OOP programming style which is based on encapsulating the variables together with the methods operating on it within classes (thus limiting the scope and accessibility of the variables).

4.3.7 Class Structure and Interactions

The overall class structure had to be planned in such a way so that the required AR functionality is seamlessly integrated with the structures imposed by the Ogre3D graphical engine framework (described in Section 4.3). What follows is an overview of the structure and program flow of the prototype's software application, with a brief discussion of the developed classes and the relationship and communications between them.

4.3.7.1 Overview of Class Objects

The author's primary coding contribution consists of 17 C++ class objects which as a whole implement the prototype's functionality. These classes are listed below in roughly in order of significance, alongside a brief description of the functions they provide.

- *ARPrototype Class*

  (ARPrototype.h, ARPrototype.cpp)

  Provides the topmost entry point for the application, implements the WinMain() method, instantiates the ARApplication object and executes it's go() method. A good place to perform early stage initializations.

- *ARApplication Class*

  (ARApplication.h, ARApplication.cpp)

  The main application object which encompasses the functionality of the other classes. The main Ogre3d framework objects are instantiated and initialized here with values provided in a configuration file or using an interactive startup dialog. The thread performing webcam video capture is created and started.

  After necessary initialization steps required by the Ogre3D engine (creation of frame listener object, cameras, 3d scene setup, resource allocation) the rendering loop is started in a blocking call to the Ogre3D startRendering() function. From that moment, all events such as the computations performed before each frame of animation, handling of inputs etc. are handled from within the ARFrameListener object code, which is described next.

- *ARFrameListener Class*

  (ARFrameListener.h, ARFrameListener.cpp)

  This class implements the Ogre3D frame listener object which handles events in the system. Callback functions for events such as start and end of frame rendering, and input device

related events are defined in this class, specifying the behaviours required by the first response application. ARController object which encapsulates functionality related to the GUI state machine is instantiated here, and updated whenever the application receives new inputs.

- *ARController Class*

  (ARController.h, ARController.cpp)

The ARController class object's purpose is coordination of the subsystems, and handling of the state machine of the graphical user interface, based on the ever-changing information being provided to the system. It initializes and communicates with threads performing network and glove input device related operations, it also queries the webcam thread for fiducials detected in the camera imagery and updates the graphical display accordingly.

- *ARNetworkThread Class*

  (ARNetworkThread.h, ARNetworkThread.cpp)

Implements network database client functionality in a separate programmatic thread. Network connections are set up allowing to read and write information to and from the MySQL database on a remote server. The information transmitted includes fiducial marker related information as well as the responder location tracking information.

- *ARGloveThread Class*

  (ARGloveThread.h, ARGloveThread.cpp)

Once the glove device is detected in the system, this thread provides glove monitoring and simple gesture recognition.Graphical indicator of glove status is also updated here.

- *ARWebcamThread Class*

  (ARWebcamThread.h, ARWebcam.h)

The thread responsible for continuous capture of headcam video frames, leveraging on OpenCV image handling routines. Each incoming frame is processed using the ARTag fiducial

detection algorithm, resulting in a list of fiducials present in the camera view, and their detected position and orientation which is forwarded to the ARMarkerManager object. The video frame is then used for updating the graphical Ogre3D texture object used in the composite GUI scene, providing the video background.

- *ARMarkerManager Class*

  (ARMarkerManager.h, ARMarkerManager.cpp)

  The object of this class maintains the data lists identifying the fiducial markers detected in the environment at any given moment, and keeps track of marker status. ARMarker objects are activated when they are needed, and deactivated after a predefined 'dormancy' period of inactivity. Visible marker 'weight' parameter used for visualization is determined based on a predefined weight map.

- *ARMarker Class*

  (ARMarker.h, ARMarker.cpp)

  An object corresponding to an unique individual fiducial marker ID. Each ARMarker object stores the geolocation and content-specific information related to a given ID. When a fiducial with a specific ID is detected in the environment, the corresponding ARMarker object is notified about it by the ARMarkerManager class, and subsequently updated with fiducial position information. An active ARMarker manages it's visual representation's behaviours by retrieving an ARLabel object pointer from the ARLabelManager class and updating it when needed. In a similar fashion, when the user requests multimedia content associated with a given fiducial, the ARMarker object retrieves and initializes an ARObject content object of required type through the ARContentManager.

- *ARLabel Class*

  (ARLabel.h, ARLabel.cpp)

  An object of this class corresponds to a graphical fiducial marker label presented on screen. Methods for updating the label's position and changing its appearance are provided. Objects of this class are managed by ARLabelManager class, which performs all the necessary resource bookkeeping.

- *ARLabelManager Class*

  (ARLabelManager.h, ARLabelmanager.cpp)

  Primarily a resource management class, the purpose of which is to minimize resource allocation delays related to the creation of graphical label augmentations. A pool of ready to use Ogre3D graphical objects is maintained and made available on request to ARMarker objects. New resources are allocated when the pool is insufficient, and deallocated when they are no longer required.

- *ARContentManager Class*

  (ARContentManager.h, ARContentManager.cpp)

  Another, alongside ARLabelManager, resource management class responsible for the creation of multimedia content objects (2D and 3D graphical entities). At present, only rudimentary resource allocation is performed here.

- *ARObject Class*

  (ARObject.h, ARObject.cpp)

  A polymorphic class providing a unified interface for handling a multitude of different content types. Presently, four types of content objects are implemented: Text, 2D bitmap, 3D model, and 360 degree panoramas. Adding code for a new content object type is straightforward

and required creating a class inheriting from the base ARObject class and overriding its virtual methods responsible for updating the object based on system inputs.

- *ARGeo Class*

  (ARGeo.h, ARGeo.cpp)

  The ARGeo object keeps track of the responder location information currently known to the system, including the location of the local device based on tracked fiducials.

- *ARTextOverlay Class*

  (ARTextOverlay.h, ARTextOverlay.cpp)

  A class simplifying the creation of text captions seamlessly integrated with the graphical interface – it is used for displaying descriptions for fiducial marker labels and for text content objects. A number of parameters such as color, opacity, size or customizeable motion flicker is supported for increased flexibility.

- *ARTimer Class*

  (ARTimer.h, ARTimer.cpp)

  A helper class utilized throughout the application, wherever precise timing, tracking of timeouts or predefined time intervals are needed. A typical usage example would be the use of timing for glove sensor gesture detection – the increases in finger bend sensor values need to be compared with absolute time measurements in order to determine whether threshold finger flexing speed is exceeded.

- *ARBitmapOverlay Class*

  (ARBitmapOverlay.h, ARBitmapOverlay.cpp)

  A helper class implementing a graphical overlay layer using Ogre3D graphical rendering capabilities. The overlays can be rotated, scaled and blended using a variety of techniques. This

functionality is not utilized in the prototype in its present form, but was included for demonstration purposes or future use.

4.3.7.2 Code Flow Overview

In this section, the Reader is familiarized with the programmatic flow of the applications software. The following narrative describes in an approximately sequential manner the events taking place during the execution of the application code, the data structures involved and the methods operating on them. It is hoped that through this approach the Reader may gain a clearer understanding of the context in which the objects described above coexist and interact. The following sections should provide a good starting point for understanding, modifying and expanding the application's code when need arises.

As stated before, the ***ARApplication*** class provides the overall application framework. A single object of this class is instantiated within WinMain[33]. From there, its *go()* method is invoked, which performs the necessary initialization of Ogre3D objects (including ***ARFrameListener***) and parameters for the hierarchical scene manager (cameras, frame listeners, objects), spawns the thread for webcam image capture and processing (***ARWebcamThread***), and finally starts the rendering loop by invoking the *startRendering()* method of the topmost object in Ogre3D scene manager hierarchy. The *startRendering()* method, once called, doesn't exit until the application is halted. Any subsequent processing of external inputs to the system from then on is provided by specialized callback functions defined within the ***ARFrameListener*** object, triggered by events such as keyboard or mouse input, beginning of frame rendering or end of frame rendering.

---

[33] The WinMain function is called by the system as the initial entry point for a Windows-based application. Source: MSDN (Microsoft Developer Network) [msdn.microsoft.com].

When ***ARFrameListener*** object is initialized, its fields are initialized; one of them is the important ***ARController*** object. ***ARController*** , amongst other functions, holds the data structures and flags reflecting the state of the applications GUI, owns the ***ARMarkerManager*** object (which keeps track of detected fiducials), and starts the glove input device thread (***ARWebcamThread)***. Keypresses received by the ***ARFrameListener*** callback functions or fingerbending gestures detected within the glove thread ***(ARGloveThread)*** need to be sent to the ***ARController*** object and interpreted, as an action of a keypress can have different outcome depending on the current state of the GUI. For example, when a fiducial is within the active central region of the screen, the label it is overlaid with is in 'selected' state indicating that content activation is possible. It is then possible to start viewing content associated with a label by flicking the index finger. When there are no 'selected' labels, such action will not be triggered.

The *update()* method of the ***ARController*** is called at the beginning of each frame in the rendering cycle. Within this method, several operations are performed. Most importantly, the ***ARMarkerManager*** object queries the ***ARWebcamThread*** for the list of fiducials detected within the most recently captured camera image and performs the required marker bookkeeping. The ***ARGloveThread*** is also queried, returning the values corresponding to current fingerbending sensor readouts and flags corresponding to detected gestures.

The ***ARMarkerManager*** class calls for a more detailed description at this point, as it encapsulates most of the functionality related to displaying augmentations based on fiducials present within the field of view of the user. The object of this class is instantiated within the ***ARController***, and updated with each new frame of the animation. ***ARMarkerManager*** holds a data structure *mMarkerArray* which is an array the indices of which correspond to all possible fiducial marker IDs (ARTAG's marker numbers). The array contents are of type ***ARMarker*** - objects of this type correspond to physical fiducial tags and hold relevant descriptive information. ***ARMarkerManager*** initializes the fields of the *mMarkerArray* at startup, using the information provided by the network/database module. This information includes:

- text for label descriptions,
- information needed to retrieve content associated with a marker from a local file (bitmap, 3D mesh):
- type of content
- unique name identifying the content
- geopositioning of the real-world marker, (x,y,z coordinates and facing direction information)
- additional fields which state whether:
    - the ARTAG marker with a given ID is in use,
    - the marker has any content associated with it

The *mMarkerArray* array is a static lookup structure which doesn't change often, or at all. However, the ***ARMarkerManager*** object holds also dynamic data structures, *mVisibleSet, mPrevVisibleSet, mDormantSet, mTempSet*, implemented using the Standard Template Library *set* container. These data structures are updated with each new frame of the camera video capture as a result of querying the ARWebcamThread object with its *getlist* method, and reflect the changing set of currently visible markers and markers that were seen very recently but are currently not visible (dormant). A dormant marker's ID is kept within the *mDormantSet* structure for a predefined amount of time, after which it exits that state (is removed from the dormant marker list). The need for a list of *visible* markers is obvious: if a fiducial becomes visible, the graphical augmentation needs to be shown to reflect that. Additional resource management actions need to be performed when a marker which was not visible before appears in the field of view – a comparison with the *visible* marker set corresponding to previous frame is performed. The *dormant* marker list was introduced for two main reasons. First, it prevents the distracting flicker of augmentations in case the real world fiducial was obscured temporarily (for example by

a passing person) or failed to be detected because of a motion blur caused by an abrupt head movement. The second reason is the resource management overhead related to allocating and freeing up graphical resources. To reduce repetitive resource allocation/destruction a "hysteresis" behavior was implemented in the form of the already described timeout period when the marker remains in *dormant* state, and the additional mechanisms of providing a ready-to-use pool of ***ARLabel*** graphical resource objects in advance, which will be described later.

ARMarkerManager also keeps track of the presence of a single *selected* marker. In proposed scheme of interaction, illustrated with the storyboards presented in Chapter 2, selection is performed simply by centering the view on a single marker, thus placing it within the central ellipsoidal 'active' area. In the implementation this behavior is customizeable, as it is based upon a weight-map which may be provided as a PNG graphical grayscale bitmap. The bitmap can be easily edited, allowing to change this behavior without recompiling the software module, for example to provide several "active" areas, placed in separate regions of the screen. In case of several markers occupying the inside of the active area, the one which has screen position attributed with the highest weight is selected.

When ***ARMarkerManager*** is signalled by the ***ARController*** object that an 'activation' gesture was performed by the user, it checks for presence of a 'selected' marker. If such marker is present, the 'content browsing mode' is activated, which means the marker labels are removed from the display, and contents associated with the selected marker are displayed. Types of content and methods of browsing will be discussed later, when the ***ARObject*** class is introduced.

Two resource management classes were created for the purposes of intelligent handling of graphical Ogre3D objects used for the necessary augmentations. The first, ***ARLabelManager*** manages the ***ARLabel*** objects used to augment the detected fiducial with a graphical icon and short text description clarifying the type of content associated with this fiducial. Second, the ***ARContentManager*** handles the ***ARObject*** class objects holding graphical resources used for displaying the informational content associated with a fiducial (text, 2D or 3D objects). Both

manager objects are declared as static members of the ***ARMarker*** class (meaning that there is only one static instance of either of them which can be accessed by any ***ARMarker*** object) and are initialized by the ***ARMarkerManager*** object during its initialization. An ***ARMarker*** object, depending on the situation, may call the aforementioned manager objects' *fetch* and *release* methods whenever resources are required or no longer needed, respectively. ***ARLabelManager***, ***ARContentManager***, ***ARContentManager*** and ***ARObject*** classes encapsulate most of the necessary Ogre3D graphical engine related function calls, while the ***ARMarker*** class is more abstract and separated from the graphical scene management functionality.

ARLabelManager object holds an array of pointers to ***ARLabel*** objects. The size of that array corresponds to the predefined maximum to the number of graphical labels which can be displayed at any given time. To conserve memory, unneeded ***ARLabel*** objects are created and destroyed dynamically when needed. However, in order to increase application responsiveness a preinitialized pool of a small number of ready to use ***ARLabel*** objects is kept by the ***ARLabelManager*** , which should be sufficient for most typical scenarios where a large number of physical fiducials is unlikely to be found within a small physical area. ***ARLabel*** objects themselves, once assigned to a given ***ARMarker*** object, implement their own visual behaviours, based on position and other information received from ***ARMarker***. They are based on Ogre3D **OverlayContainer** objects which may contain text, graphical and 3d mesh objects and are overlaid on top of the existing 3D scene being rendered.

ARContentManager class manages Ogre3D resources such as 3D meshes depicting buildings, 2D graphical elements such as floorplans or diagrams, or other types of multimedia content. In order to create a system which is extensible with regard to the available types of media content while still providing a unified programming interface, the polymorphism feature of C++ was utilized. A base class ***ARObject*** was developed, containing virtual methods, and a set of extended classes ***ARObject2D***, ***ARObject3D***, ***ARObjectText*** were derived from it, overriding the virtual methods. New types of content object classes can be added on at will, and no significant

changes need to be introduced in other parts of the applications' code. This allows exposing a unified interface for handling different types of content, while the specific behaviours most suitable for a given type of content can be implemented individually. As an example, let's consider viewing a 3D wireframe model of a building (implemented in ***ARObject3D***) as opposed to a 2D map of one of the floors (***ARObject2D***). Users modification of the display viewpoint in both cases is performed by repositioning the fiducial within the field of view (through head movement), the vertical and horizontal components of this movement are interpreted differently by the polymorphic classes. A wireframe mesh may react to user looking to the left by modifying its yaw, while a flat map may simply pan to the left. More advanced behaviours are possible as well.

Throughout the code, a number of predefined values were used to initialize sizes of arrays, specify durations of timeout, default screen size and many other parameters. In most cases they were specified using define directives, which can be all found in **ARDefines.h** header file and modified if needed.

4.3 Graphical Engine

4.3.1 Choice of Rendering[34] Engine

The prototype application's graphical frontend (GUI) was conceptualized (see Chapter 2) as a mixture of video, 2D and 3D graphics, in a combination which is typical for many Augmented Reality systems. It was then succesfully implemented using an open-source rendering

[34] "[Rendering is] the generic term for "creating the picture" from a scene's models, surfaces, lighting, and camera angle. Basic aspects of any rendering algorithm include visible surface determination, shading, scanline conversion ..." - Terrence Masson, "CG 101: A Computer Graphics Industry Reference" (New Riders, 1999).

engine[35] OGRE3D. In this section the considerations and rationale behind the selection of this particular graphical visualization library in the light of the application's specific requirements are discussed.

4.3.2 Requirements of the AR Application

To illustrate the requirements considered in the early prototyping stages, a concept design in the form of a sequence of "mock-up" storyboards from a first-iteration paper prototype of the GUI will be used. An example storyboard is shown on Fig 4.5, it depicts several distinct elements overlaid on top of the camera-provided image of the environment. The 3D[36] wireframe mesh in the bottom left corner shows the layout of the building and positions of teammembers within it. Whenever applicable, the elements of the display should be correctly oriented with regard to the direction the user is facing (determined by the detection of fiducials within the field of view). A pseudo-radar 2D display, placed in the upper-right corner, shows a mini-map of immediate surroundings, also incorporating mobile indicators of teammembers. The fiducial placed on the wall is augmented with a graphical element, and an indicator of the fingerbending sensors is displayed below right. A separate 'window' displaying some text and a photograph or a video transmission is displayed in a semi-opaque fashion, allowing the user to still discern some of the visual features of the video background. This concept design was the starting point for determining the functionality required from the rendering engine used.

---

[35] Rendering engine – a loosely defined term used usually in computer entertainment / simulation industry to denote a software API providing the rendering of complex scenes. Here this term is used in the sense of the code providing the management of geometry. It allows the programmer to operate on a higher level of abstraction, manipulating 3D scene building blocks such as meshes, lights, cameras, while the data structures containing the graphical primitives are transformed, culled, partitioned (octet trees, portals) internally by the engine before relaying them to the graphics hardware for display. In contrast, a *game engine* usually also provides other game-related functionality (sound, network, artificial intelligence, physics etc.) besides the rendering.

[36] The "3D" description used in this chapter refers to three dimensional scenes projected onto a 2D plane using perspective projection, not stereoscopic, truly three-dimensional imagery.

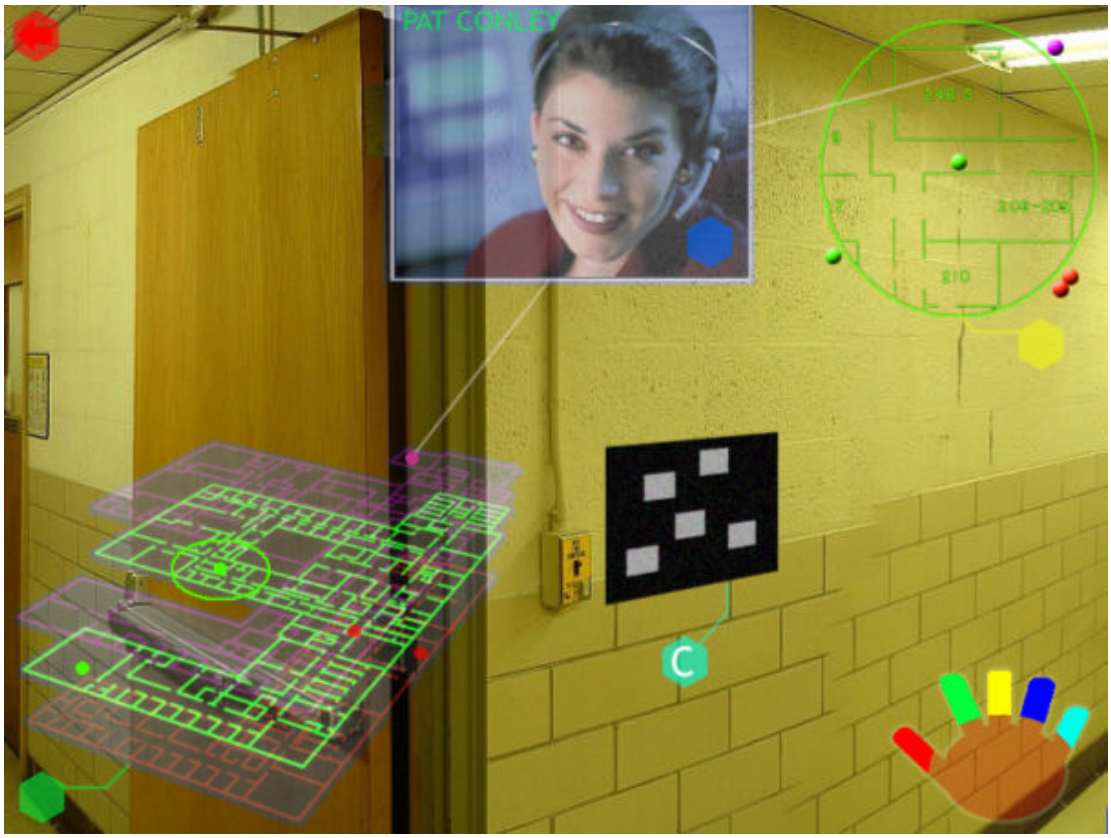


Figure. 4.5. Mock-up screen from a first draft paper prototype of the system.

4.3.2.1 Rendering of Scenes Composed of 3D Objects

Implementing a complex real-time synthesised GUI benefits from having a unified approach to all the disparate graphical elements (2D bitmaps, 3D objects, text). Such abstract and flexible perspective is possible when every individual element of the display is considered an object in a 3D scene. Although in the example given in Fig 4.5 only the building mesh appears to require third dimension, there are substantial benefits to using a 3D renderer to generate the GUI in its entirety. Today's graphics cards (GPUs) provide hardware acceleration of sophisticated 3D rendering operations, including texturing and lighting, geometry calculation, perspective

transformations to name but a few. Although 2D operations (blitting[37], simple shape drawing) are hardware accelerated in modern graphics cards as well (mainly for the purposes of windowed operating systems, and applications such as office productivity tools), that functionality is usually limited. On the other hand, under specific circumstances (parallel/orthographic projection used instead of a perspective projection) the 3D graphics pipeline can discard the third dimension information and emulate anything that 2D accelerator can offer. In the light of this, it becomes possible, and often desirable, to use 3D acceleration even when creating plain 2D[38] graphical interfaces for applications requiring more sophisticated graphical functionality (blending multiple layers, interpolation and filtering, scaling) and efficiency. Such approach is already used in Apple OS X operating system's Quartz compositing engine, which provides support transparency and 'fisheye' scaling in the interface. Similar approach will be utilized within the upcoming Windows Vista operating system, where the 'Avalon' graphical interface will be entirely 3D-accelerated. The shift to unified 3D rendering can be also observed in Microsoft's handling of their game programming API: DirectX. Older versions of DirectX SDK(prior to 8.0) contain separate APIs for handling 2D and 3D operations: DirectDraw and Direct3D. As of version 8.0, the two APIs were integrated into one, DirectGraphics API, with strong emphasis on the functionality formerly contained in Direct3D.A generalized diagram on Fig. 4.6 illustrates how a 3D renderer can be used for an AR application. Objects (a),(b),(c),(d) are geometrically simple, texture-mapped[39] rectangles, placed within the field of view of a virtual camera, they correspond to the various GUI

---

[37] Bit blit (bitblt, blitting etc.) is a computer graphics operation in which two bitmap patterns are combined into one. (source: Wikipedia)

[38] The distinction between a purely 2D and a 3D interface is not very clear-cut, as even typical window-based environments feature a very strong 3D cue: occlusion – which leads to such interfaces being often referred to as 2.5 dimensional.

[39] "Texture mapping is a method of adding realism to a computer-generated graphic. An image (the texture) is added (mapped) to a simpler shape that is generated in the scene, like a decal pasted to a flat surface. (Source: Wikipedia).

elements present on screen. The viewpoint and field of view of the camera, as well as the placement and order of the rectangles determine the final result of rendering, shown in the top left inset (i). The various features relevant to the presented design problem will be discussed individually in the sections to follow.

4.3.2.2 Hierarchical Scene Manager.

Creating and controlling complex 3D scenes containing numerous objects calls for some way of specifying groupings and relationships between objects. Typically this is provided by code called a hierarchical scene manager (sometimes called a scenegraph). Let us consider objects (e) and (f) from Fig. 4.6, two models placed within a 3D scene, viewed through the virtual camera number 2. Typically, each of the objects would have an individual coordinate space associated with it, for the purposes of self-scaling or rotation (for example, when the helicopter model needs to be rotated along its own axis). However, modifying each and every object individually whenever the whole scene needs to be rotated, while preserving the correct relative positions and orientations of the objects, would be problematic to say the least. Treating the whole scene as a graph, with objects belonging to nodes of such graphs greatly simplifies such an operation. A root node in the hierarchy (Fig 4.6g) provides a coordinate system (shown as the large axes on Fig. 4.6) which acts as a global reference frame. The two objects within the example scene can be associated with two children-nodes of the root node. Rotating, scaling, or deforming the root node will now have the effect of rotating, scaling and deforming its children nodes as well. A scene hierarchy may be very complex, expanding into a large node tree, allowing for intuitive and realistic handling of scenes consisting of many objects (for example in a simulation of a multiple-

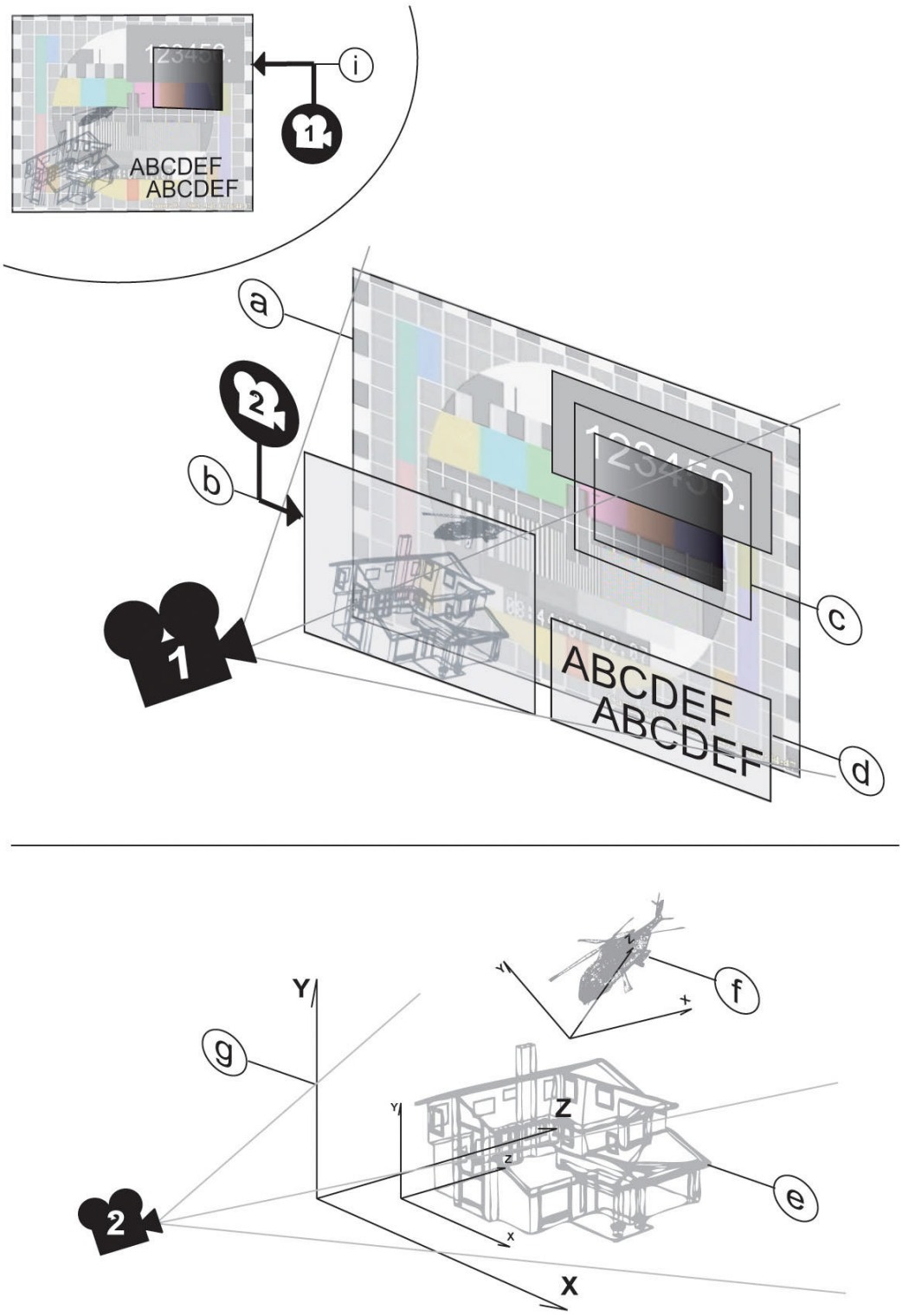


Figure 4.6. Use of a 3D graphics renderer for a mixed reality graphical user interface

-joint robot arm). The scenegraph's principle of operation can be described in a simplified manner as traversing the node tree and successive multiplication of matrices corresponding to affine spatial geometric transformations. Such node orientation inheritance approach allows for a hierarchical cause-and-effect behavior – such as, for example, when the table on which the robot arm is rotated, all joints of the arm follow suit.In a mixed reality application such as the one presented here, though the complexity of the 3D scene is kept at a low level compared to that of, for example, a flight simulator, using a scenegraph is nonetheless still required for hierarchical scenes such as the GUI layer structure pictured on Fig 4.6, or a 3D building display showing locator dots representing the continuously changing position of responders alongside a reference geo-navigational grid.

4.3.2.3 Rendering of Multiple Scenes / Multiple 'Virtual Cameras'. Render to Texture Target (RTT)

The diagram on Fig 4.6. depicts two virtual cameras, one (nr 1) is used to observe the layers combined into the final GUI, the other (2) is pointed at another set of objects. The rendering engine used for the application should allow for multiple camera viewpoints for such configuration to be available.

Furthermore, the multiple renderings obtained in this fashion need to be presented to the user on the same display. The simplest approach is to render to a rectangular subsection of the display device, however the most flexible and powerful way to achieve that effect is by using Rendering to Texture. Instead of rendering to the full-screen or windowed display device, the bitmap image produced by the renderer is used to create a dynamic texture, which is then used for texture-mapping an object in another 3D scene (Fig 4.6b). This allows unparalleled level of control over how the mini-scene will be incorporated into the main GUI. This is due to the fact that the textured object, acting as a small virtual 'projection screen', can be rotated, scaled or

distorted. It may be placed in a fixed location on screen as an interface element, or used as a dynamic augmentation over a fiducial marker viewed through the camera, precisely reflecting the markers pose (for example viewing an animated rendering of a device displayed over a paper fiducial placed on a wall). If the renderer allows for it, the texture itself may be alphablended with the background. All these operations are possible regardless of the properties of the original 3d scene (lowest section of Fig. 4.6), such as its position and orientation and lighting. It would be possible to composite a hybrid GUI without Render-To-Texture targets just by careful placement of all models within the within the same camera space, but this introduces a slew of problems, such as clipping of a 3D model by the background rectangles (unless depth buffer checking is selectively turned off for some objects), and considerable limitations to lighting and blending techniques available. Having the intermediate texture stage potentially allows for further image processing to be applied to the results of the rendering. Rendering to texture targets thus was high on the requirement list for a suitable rendering engine for implemented application.

4.3.2.4 Incorporating a Live Camera Video Stream into the Scene. Dynamically Generated Textures.

As in most AR applications, in the described system it was necessary to incorporate the incoming headcam video as the background layer of the display (Fig 4.6a). Capture of video from a digital camera is typically not a part of a graphical rendering engine (with some exceptions). The ease of potential implementation of camera-to-texture functionality had to be considered. An enabling feature of a renderer which is a first step toward implementing a camera texture is the accessibility of textures in general. If an engine supports textures which can be freely accessed via a buffer in memory, and updated with each frame of the animation, then more likely than not a video texture is possible for that renderer.

4.3.2.5 Alpha Blending

Supported by most of current hardware 3D accelerators, alpha-blending allows for textures and geometry of varying pixel/vertex/triangle opacity. In general terms, alphablending extends the color space used by the renderer. The colour of a vertex (a point in the rendered geometry) or a texture pixel is usually specified by providing the R, G, B values corresponding to contribution of red, green and blue in the final color. When alphablending is used, yet another value (A) needs to be provided for each vertex or pixel, specifying how opaque that vertex or pixel is. High alpha values translate to large contribution from that pixel/vertex color in the color of the pixel in the final rendering, low values mean that the final screen pixel color will be mostly based on what was below the transparent object. This is illustrated in Fig 4.6c, a rectangle is texture mapped with a texture using nonuniform alpha values, starting with low alpha on the left and gradually increasing to full opacity (maximum alpha) at the right border of the rectangle.

In the first-response application, lack of alphablending support would mean, for instance, that a building floor blueprint shown in an area of a screen would have to obscure the camera video in the background plane, which would be unacceptable for situations requiring increased situation awareness. When nonuniform alphamask is used with the blueprint, the contours of the blueprint can be viewed directly over the camera view.

Results of several studies show that, when used properly, alphablending provides an effective method of achieving good tradeoff with regard to perceptual interference between foreground and background visibility [Harrison95], [Baudisch04]. This is important in situations requiring attention switching between two or more sources of information, which is precisely what the envisioned usage pattern for our application is.

Besides alphablending, the availability of other blending capabilities, if offered by a graphical engine, would be advantageous. Such a capability could be for example blending of layers by multiplying or subtracting foreground pixel color values from the background, this could be used for customized overlay element blending.

4.3.2.6 Miscellaneous

Some other features which are not critical, but could facilitate development as well as content creation for the developed application:

- import of multiple graphical bitmap formats. Some of the widely used image formats are:
  - Windows Bitmap (.bmp),
  - Portable Network Graphics (.png),
  - Adobe Photoshop (.psd),
  - JPEG File Interchange Format (.jpg),
  - Truevision Targa (.tga),
  - ZSoft Painbrush (.pcx)
- import of 3d meshes in popular formats / availability of converting tools (mesh importers). Popular mesh formats include:
  - VRML
  - Maya (.obj),
  - 3DStudio (.3ds),
  - Milkshape (.ms3d),
  - Microsoft DirectX (.X))
- easy and flexible text rendering with control over font, size and other properties of rendered text

4.3.2.7 Ease of Use and Flexibility

The learning curve characteristic of a particular rendering API is another important property that had to be considered given the project scope and timeline. A simple to use programming interface is always welcome, unfortunately there is often a tradeoff with regard to

ease of use and the engine's power and flexibility. A very abstracted, simplified programming rendering interface may be perfect for setting up a simple 3D scene, but when advanced functionality is required, such renderer may be simply too limited. A good example of the tradeoff is the two 3D programming interfaces offered by Direct3D (in versions of DirectX SDK prior to 8.0), referred to as the Retained Mode and the Immediate Mode. Retained Mode is a high level API which can be treated as a rendering engine in that it provides a geometry engine (management of vertex database etc.), object management (scenegraph) and scene animation. It is simple to use, but also very limited and inefficient in terms of rendering speed. Functionality such as variable alpha textures cannot be implemented using DX3DRM (DirectX3D Retained Mode) functionality alone. On the other hand, there is Immediate Mode, which provides a 'thin', hardware independent layer between the developer and the graphics hardware. In this mode the application needs to handle the scene geometry resources on its own, allowing for creation of highly optimized rendering engines but also requiring a significant code development effort.

Well organized and complete documentation of the rendering API is a must, as well as support available for developers. This is particularly true for open-source solutions, where an active and helpful community of engine developers and engine users can sometimes rival commercial solutions with regard to resolving bug issues or other problems, while lack of such support provides a strong argument against using that particular solution.

4.3.2.8 Code License

The limited budget of the project, as well as the potential for further unhindered expansion was the motivation behind implementing the system based on mostly open-source libraries. Within the open-source licensing world, a multitude of license types exist, differing in requirements and limitations with regard to modifying and redistributing the code (copyleft[40],

[40] Copyleft: a general method for making a program or other work free, and requiring all modified and extended versions of the program to be free as well. (Source: [http://www.gnu.org])

source code disclosure). Besides the GPL[41] and LGPL[42] licenses and several other variations, there are custom licenses allowing unrestricted use of code for research and noncommercial purposes but involving a licensing fee when the product is used commercially. Some code is released into the public domain – uncopyrighted.

4.3.3 Alternatives Considered.

With the above requirements in mind, a number of available 3D engines was examined, using information provided by the engines' vendors and found in an extensive database of 3D engines found at [http://www.devmaster.net/engines/] . In some cases, sample code and binaries demonstrating the use of the engine were examined, but no systematic evaluation of all engines involved was performed.

There were several possible paths with regard to type of 3D library used. On one end, there were low-level graphics programming APIs like OpenGL or DirectX which could be used directly. Simple helper libraries exist which facilitate development by wrapping some of the more tedious setup operations in that case. In another category there were high level 3D graphics engines such as the open-source OGRE, CrystalSpace or Irrlicht, or the commercial low-cost Torque engine. Yet another alternative is provided by AR-specific software libraries, such as the ARToolkit or ARTag SDKs which encapsulate the fiducial recognition functionality together

---

[41] GPL: GNU Public License: "When code licensed under the GPL is combined or linked with any other code, that code must also then be licensed under the GPL. In effect, this license demands that any code combined with GPL'd code falls under the GPL itself." (Source [http://www.openoffice.org/FAQs/faq-licensing.html])

[42] LGPL: GNU Lesser General Public License. "Code licensed under the LGPL can be dynamically or statically linked to any other code, regardless of its license, as long as users are allowed to run debuggers on the combined program. In effect, this license recognizes kind of a boundary between the LGPL'd code and the code that is linked to it." (Source [http://www.openoffice.org/FAQs/faq-licensing.html])

with visualization of the augmentations overlaid on top of the camera image. Each of these alternatives presents a different set of drawbacks and benefits, which will be discussed here.

4.3.3.1 Low-level Graphical Libraries

They provide hardware independent API for accessing the functionality offered by 3D acceleration hardware, but not much beside that. Some implementations may provide software low-level renderer if an accelerator is not present.

This category is listed here for primarily for the sake of completeness, and because all of the 3D engines considered by us are based either on OpenGL or DirectX3D. Although it is possible to implement an AR system from scratch using a low-level graphics API, the development effort would have been prohibitive in case of the described system.

4.3.3.1.1 OpenGL

The open-platform, vendor-neutral industry standard graphics API, with implementations offered by SGI, Microsoft and others is similar in concept to the DirectX Immediate Mode discussed previously in its low-level approach tightly coupled with the operation of the graphics hardware's state machine.

4.3.3.1.2 DirectX3D

Part of Microsoft's DirectX programming API, already discussed above. The platform is limited to Microsoft Windows operating system. Current version of DirectX SDK as of April 2006 has version number 9.0c.

4.3.3.2 Open-source Engines

4.3.3.2.1 Ogre3D Rendering Engine: [http://www.ogre3d.org/]

The OGRE3D engine is a powerful, to a large extent platform/library independent 3D engine, allowing for rapid development of 3D applications. Unlike many other available free open-source engines, it has not been designed with game development in mind. In direct contrast to strictly game-oriented engines which incorporate functionality related to sound, artificial intelligence, physics, input-output devices, OGRE3D is limited to graphics and is designed to be open-ended and flexible, making it very suitable for non-game applications such as scientific visualizations. It conceals much of the complexity related to setting up the devices and scenes for rendering, although lower level operations are available when required by advanced programmers. It also supports several 'state-of-the-art' rendering techniques, making it competitive with some commercial products. Required features such as Render-To-Texture and Alphablending are supported, as well as additional features listed above as optional.

OGRE3D can render using OpenGL as well as DirectX, acting as a higher level wrapper for those libraries. Using the OpenGL rendering subsystem of Ogre3D enables development for other platforms as well, making it potentially much easier to port the application if desired.

It provides GUI building tools by means of 2D and 3D overlay functionality and an event driven GUI module, reducing the amount of time and code needed to implement a custom user interface. A great majority of parameters and settings for the devices, objects in scene and types of rendering techniques involved can be specified by easily editable script files, allowing for experimenting and fine-tuning the results without recompiling source code, thus facilitating shorter development time.

Good programming design practices in case of OGRE3D extend to documentation which appears extensive, complete and up-to-date. A user's manual is available as well as a full API

reference describing all members of the class hierarchy. OGRE public forums are very active and there is a good level of support in case of problems.

4.3.3.2.2 Crystal Space: [http://www.crystalspace3d.org]

An extensive game development framework based on OpenGL, it has a plugin-based architecture in which individual modules providing various functionality may be linked in dynamically. Windows, Unix and Mac platforms are supported in varying degrees. All the functionality critical to the AR application is supported, such as render-to-texture, multiple cameras, GUI building, as well as the non-critical features are present (fonts, mesh importers).

Documentation is extensive, both a manual and an API reference are available, however they are reported not to be up-to-date in all areas. Additionally, the overall design philosophy and offered functionality was found to be more game-specific and broader in scope than was required for the project. The engine usage appeared less intuitive than in OGRE3D, however that is only the author's subjective opinion based on limited exposure to this software.

4.3.3.2.3 Irrlicht: [http://irrlicht.sourceforge.net/]

A graphics oriented engine supporting OpenGL, DirectX and software renderer as rendering outputs, it allows textures as render target, alphablending and mixing of 2D and 3D graphics, as well as some functionality for building GUIs,. As of version 0.14.0 (30 November 2005), all 2D drawing functions can draw into textures, and multiple scene managers can be used side-by-side.

Irrlicht is free, open-sourced and licensed under the "zlib license": it can be debugged, extended and modified without the need to publish the changes. Documentation and tutorials are available, an active community provides support. Engine is reported to be extremely easy to use, but lacking in some areas of the featureset. An attractive, although not critical feature with regard to multimedia content is the native import of a large number of bitmap and mesh file formats.

4.3.3.3 Low-cost Commercial Engines.

This category contains 3D engines geared specifically for low budget (“indie”) game development. Some of the engines in this category were developed as in-house tools by computer game development companies, and later released to third-party developers under various licensing conditions.

They usually incorporate a wide range of functionality and tools for content generation, level editing etc., most of which is not relevant to this project’s needs. Being game oriented, they may also not be flexible enough with regard to the limited but quite specific visualization needs of this project.

4.3.3.3.1 The Torque Game Engine [http://www.garagegames.com /]

Advanced engine suited for development of computer games, very extensive feature set which in this case is reported to leads to a steep learning curve. Judging from user feedback, it is a very robust and powerful solution, for a comparatively very low license price of 100$ per programmer, no royalties. Supports Render-to-Texture.

4.3.3.3.2 TV3D SDK 6 [http://www.truevision3d.com/home.php*]*

Another engine for game development. Reported to be very fast and the API simple to learn, however it doesn’t offer Render-To-Texture support. Free for non-commercial uses (watermark displayed), commercial usage license price is 150$.

4.3.3.4 AR-specific libraries.

Several of the fiducial planar marker systems which are discussed in Section 4.4 of this chapter are provided together with rendering code which allows for rapid development of Augmented Reality applications. Indeed, many of the prototype systems presented at ISMAR conferences [ISMAR] are based on ARToolkit’s rendering capabilities without any additional customization. Such libraries usually offer video capture, detected fiducial marker position and

pose calculation, camera calibration and facilities for correct display of geometry overlaid over the camera video.

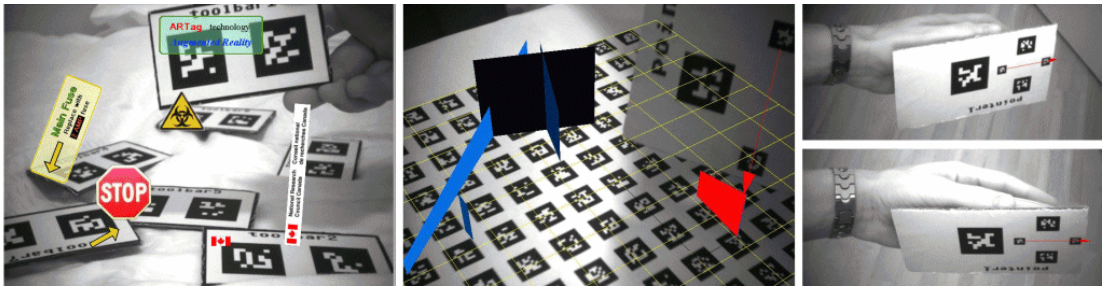


Figure 4.7. Graphics-related features of ARTag Rev.2 SDK

4.3.3.5 ARToolkit [http://www.hitl.washington.edu/artoolkit/]

ARToolkit code is available in two versions, one includes an OpenGL-based renderer. There is an extended version of ARToolkit which has inbuilt support (based on Open-VRML library) for reading and displaying 3D models stored in VRML97 format, which facilitates displaying 3D models superimposed over recognized fiducial markers. Camera image capture and display is handled using Microsoft DirectShow library.

4.3.3.6 ARTag SDK [http://www.cv.iit.nrc.ca/research/ar/artag/]

The ARTag library, which started as an extension to ARToolkit, now comes bundled with an OpenGL-based SDK which makes it a standalone solution similar to ARToolkit. Support for camera capture is based on OpenCV's CvCam library for USB2 cameras, IEEE-1394 cameras from Point Grey research are also supported. 3D objects in WRL (VRML), OBJ (Wavefront, Maya), ASE (3D-Studio export) files can be loaded from disk and displayed as augmentations. Additionally, Open-GL based support for alpha-blended 2D augmentations has been provided.

While both libraries provide a comprehensive solution, the use of such all-in-one solution has several drawbacks. The requirements regarding the fiducial planar marker system

performance become entangled with the requirements pertaining to graphical rendering, which often leads to a less favourable set of tradeoffs. In Section 4.4 the features and characteristics of ARTag and ARToolkit are reviewed from the standpoint of detection effectiveness, and ARTag emerges as the winner given the requirements.

ARTag SDK, due to the inclusion of ready-to-use alpha-blended 2D overlays, indeed seems a more appropriate option for development, although at the time of graphical engine selection the ARTag Revision 2 containing the full SDK hasn't been created yet (it was released in February 2006). The lacking features, such as a fully featured scenegraph, rendering to texture targets to name the most important few, would need to be implemented based on plain OpenGL, as already mentioned this is a serious drawback due to the development effort required.

Ultimately, independently developed libraries for fiducial marker recognition and for graphical rendering were opted for. This decision guaranteed bigger flexibility with regard to further system expansion in the future, such as the freedom of using a different planar marker system or expanding the system with complex visualizations not considered at this stage. The thrust in the development of fully featured, general purpose open-source 3D engines such as OGRE or IRRLICHT by multiple developer teams and communities guarantees better chance of continued support and growth, whereas Augmented Reality specific code, being much more specialized, is usually developed by small or even single-person (in case of ARTag) development teams.

The cost of this flexibility in this case came down mainly to the necessity to implement camera-to-texture rendering functionality. Each camera video frame captured for the purposes of marker detection by the fiducial module needs to be reused by the graphical rendering subsystem and displayed as the background layer in the GUI. This was implemented succesfully, giving us extended flexibility with regard to how the video may be displayed, since textures can be manipulated quite easily by 3D engines. This opens up potential possibilities for video image

enhancement (low light conditions), stereoscopic displays, or other features in systems built upon the work presented here.

4.3.4 Initial Results with Ogre 3D SDK

From the available alternatives, OGRE3D library was selected to be the 3D engine used in the project. From the perspective of the several months spent developing the system, one can say that the decision paid off on a number of occasions. OGRE3D proved to be flexible, stable and intuitive to learn and use.

Before OGRE3D-based development began, a simple application was developed for the purposes of testing and demonstrating selected techniques natively available using the Ogre3D engine. The results of this evaluation will be briefly discussed now below, providing additional insight into development with Ogre.The source code for the test application is included on the Appendix CD. Please see Appendix A for setup instructions.

4.3.4.1 Scene Manager

In the example pictured Fig 4.8, a scene was set up containing several objects (3D meshes loaded from files (Fig 4.8d,g), generic plane meshes generated on the fly (Fig. 4.8c,e,f) and two cameras. In Ogre, the scene hierarchy is based on scene nodes, which can be joined together in a parent-child relationship, creating a node tree. An individual node can be manipulated (translated, scaled, rotated), the children nodes of that node will inherit these transformations. Entities (3D meshes), cameras and lights may be attached to nodes. Entities are assigned materials, which contribute to the objects' appearance. The example framework code provided with Ogre may be modified to use one of the several available scene manager implementations, depending on the application's requirements (optimizations when displaying large terrains, etc), for this application's purposes the default scene manager was sufficient.

4.3.4.2 2D and 3D Overlays

Ogre3D 2DOverlay object class (example Fig 4.8a) allows for overlaying of text windows (with a texture background or transparent) on top of the scene. This is helpful for displaying text information (selected parameter values, runtime information), as well as bitmap graphics (2D maps, diagrams) which can be rotated and scaled easily. In the example a simple text overlay is defined within the overlay script text file, the background texture is animated (rotated).

3D objects may be incorporated into the overlays as well, 'floating' in front of the scene (useful for map displays, interactive 3D and 2D schematics viewed superimposed on the external camera view, “HUDs”). In the example, a file-loaded mesh is used as a 3D overlay (Fig 4.8g), its orientation changes to reflect the current camera orientation.

4.3.4.3 Viewports

Each virtual camera has a viewport object associated with it. First viewport in the example application shows the scene observed by the first camera, it is set up to be displayed within the entire application window rectangle. Its background color is red. The second viewport (b) is displayed in front of it within a smaller rectangle, showing the same scene viewed from a differently placed second camera, with a differently colored background (green).

4.3.4.4 Render-To-Texture.

Rendering to a texture is quite straightforward, in the described example the first viewport’s output is used to create a texture used for two of the 2D plane objects in the scene (Fig. 4.8 f,c). Using textures as render targets allows to set up complex scenes utilizing all the benefits of a scene manager, render those scenes to textures which are then used for 3D objects (for example on augmentations over optical markers) or 2D objects (interface elements).

4.3.4.5 Dynamic Textures/Nonuniform Alpha Blending.

Besides render-to-target textures, it is possible to create dynamic textures. The contents of such textures may be accessed and modified on-the-fly. One of the possible relevant uses is the display of camera video or other types of 2D data through a texture applied on an object, applying sharpen filtering to video image, indicating movement in a video scene by highlighting sections of image changed since last video frame, drawing simple geometry or procedurally generating textures. In the example application, a dynamic texture is used on two of the scene objects (Fig. 4.8g, e). The texture is procedurally filled with a semi-randomized pattern changing with time. The texture is 32-bits per pixel, which includes 8 bits of alpha (opacity) information per pixel - using non-uniform alphablending for a single texture is demonstrated in the example. Such a technique can be used for the display of text or drawings over an image, displaying sun flare light effects in games etc.

4.3.4.6 Material Blending.

Besides texture alpha blending, different blending properties may be set for materials, determining how the object is combined with the background scene (including objects placed behind it). It is possible to define the material blending behavior in a highly customized manner. In the additive blending mode, pixel values of the source (the object in front) are added to the values of the target (the scene behind the object). In the modulate mode (shown for the c plane in the example), values are multiplied. Several other possibilities exist.

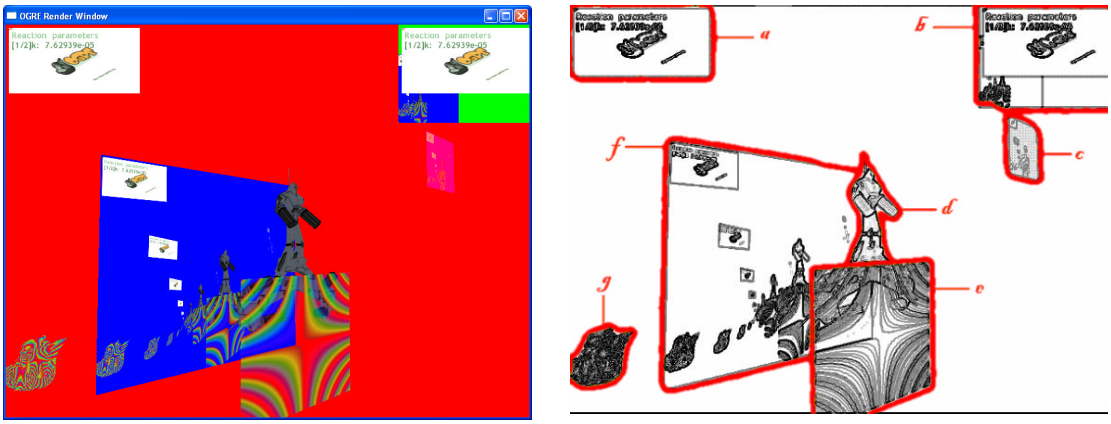


Figure 4.8. Output of a test application showcasing graphical functionality available through OGRE3D.

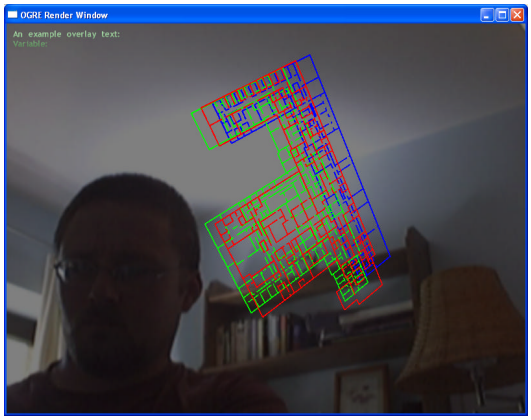


Figure 4.9. Overlaying of 3D geometry over video camera imagery.

For the purposes of examining the material blending capabilities of OGRE3D, a simple scene was set up, in which a camera captured texture was used as the screen background, and a wireframe mesh model of a building (old Kingsbury Hall) was overlaid on top of it, alongside a simple text overlay (See Fig 4.9). The model can be manipulated (rotated,scaled) in real time using the mouse. In order for the model to always remain on top of all the other 3D scene elements, the materials used for each the building floors had to be assigned a “depth_check off” attribute in the material script. Self illuminated materials were used to make the mesh appearance independent of any light sources in the scene (if there were any). No blending method was used at this stage, the mesh is fully opaque and obscures the video image pixels underneath.

Fig. 4.10 shows the building mesh assigned with a material which was additively blended with the video image using the “scene_blend add” attribute for the material. In this mode of blending, the color of the rendering output is added to the scene.

In the next example, pictured on Fig. 4.11. “modulate” blending parameter was used. In this mode, the pixel color values of the blended object are multiplied with the background pixel colors.

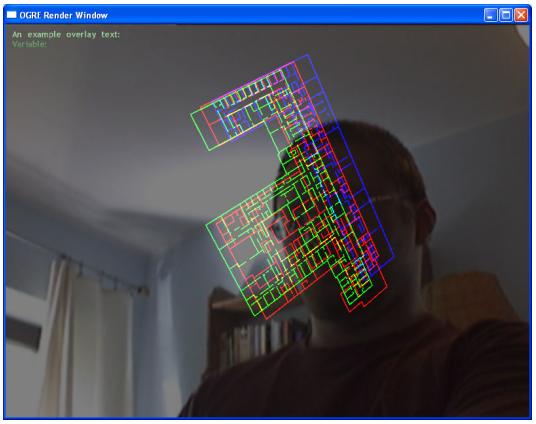


Figure 4.10. Additively blended material.

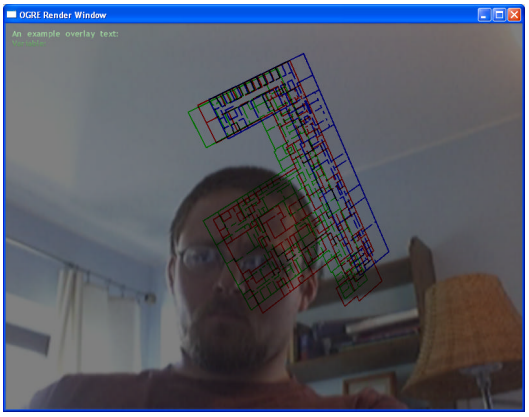


Figure 4.11. Modulation blending.

Another blending possibility is through the use of use of simple (uniform) alpha blending. Using a range of alpha values for materials assigned to individual scene objects allows to highlight some information, while keeping non-critical information filtered out. Higher alpha value corresponds to a more visible object.

4.3.4.7 Camera Based Texture.

Since Ogre3D doesn’t provide video capture from imaging devices, an external library had to be used in order to display live video on a texture. Feedback from the online OGRE forums, and code snippets provided by another user of OGRE3D led to the successful solution to the problem of incorporating a video stream into the 3D scene in form of a dynamic texture. Intel *OpenCV* image processing library provided video capturing functionality and was successfully

interfaced with Ogre3D texture manager. A demonstration of the result is pictured on Fig 4.12. Multithreading was necessary in order to maintain high rendering frame rate, due to the inherent latency in the camera video capture process, which conflicted with the requirement of keeping the rendering loop refresh rate as high as possible. The *ptypes* library was used to simplify the creation of programmatic threads.

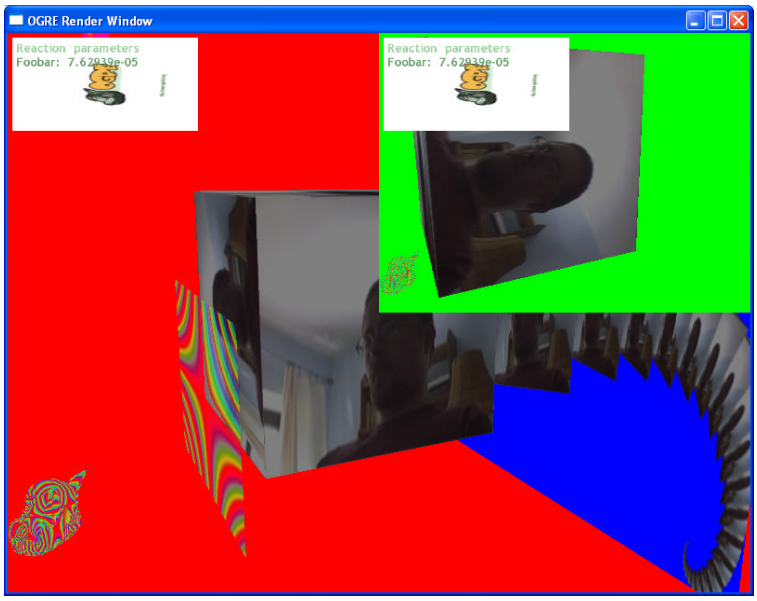


Figure 4.12. The cube in the center of the screen is texture-mapped with a continuously updated (at a slower rate than the rest of the scene) video camera based texture.

4.4 Planar Marker System

4.4.1 Registration in Augmented Reality

Implementing Augmented Reality requires a means of registering (tracking) of the location of physical objects relative to the user's position and point of view. This allows for the display of synthetic augmentations superimposed onto and annotating the real environment. For example, user's position and direction of gaze may be used to render an X-Ray vision view of the environment, showing location of obscured objects (like electric wires inside a wall). The environment's appearance can be altered, allowing to simulate an emergency (as illustrated on Figure 4.13) Another possibility might be to place a virtual building object in the environment in

order to examine potential visual interactions between an architectural design and already existing buildings. Yet another may be adding virtual descriptive labels 'hovering' above parts of a complicated device and reacting to users' actions in realtime.

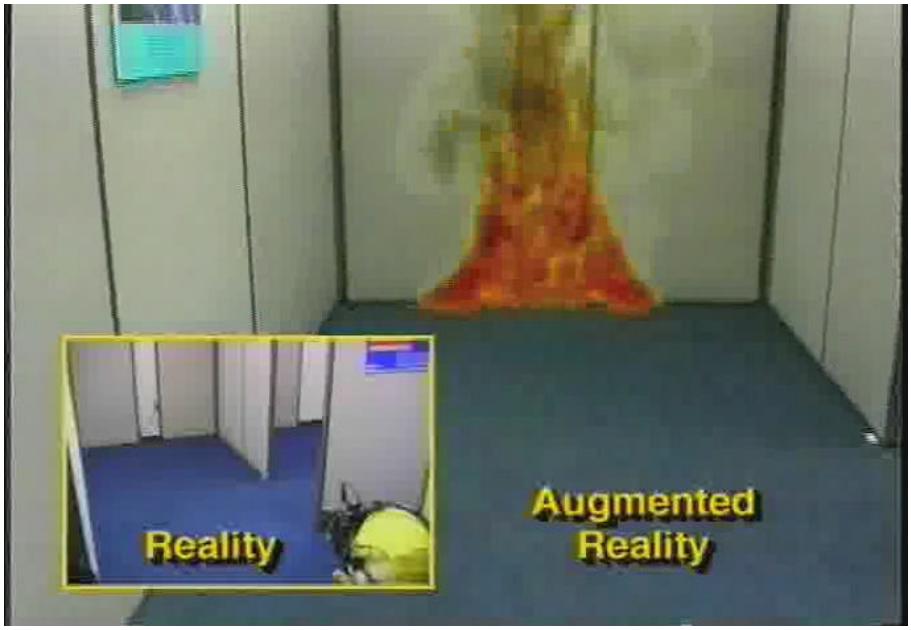


Figure 4.13. Example of an Augmented Reality firefighter training application.
The inset shows an external view of the scene (the AR-equipped trainee can be seen). The rest of the figure shows the trainee's view augmented with a simulation of fire in a predefined location.
Source: [http://www.harmlesshazards.com/].

In both cases, in order to provide effective and seamless blending of the real and virtual scenes, specific real-time information about the user's current viewpoint is needed. For video-seethrough HMD-based applications such as the device proposed here, this translates to the following information about the head mounted camera:

- intrinsic (internal) information, such as:

  focal length and lens distortion

- extrinsic information (with regard to external environment), such as:

  Six degrees of freedom information about the camera position in three dimensions (x,y,z coordinates) and pose (orientation: often expressed as pitch,yaw,roll).

There are several possible approaches to providing the above information, some involving electromagnetic field sensors (as used in Polhemus Liberty trackers [www.polhemus.com]), inertial, accelerometer based methods (InertiaCube,InterTrax by Intersense [www.isense.com]), infrared sensing (Intersense Constellation), and vision-based methods, which will be discussed in more detail in this chapter.

As noted in [Azuma97], Augmented Reality applications are particularly demanding with regard to the accuracy and speed of registration (compared to Virtual Reality), as visual-visual[43] conflicts brought upon by registration errors are very easily detected by the human eye, especially when motion is involved. Static registration errors are often related to inaccuracies in the intrinsic camera information - such as not accounting for lens distortion effect, or the error introduced within the tracking/sensing system - as a result the augmentation does not appear in the correct position/pose. Dynamic registration errors are related to the discrepancy between the current extrinsic information provided by the sensors and the real camera position. For example, if a tracking/rendering system delay in an HMD-based AR application causes the virtual object's movement to be delayed with respect to the real environment's movement, the illusion of the

[43] **visual-visual** conflicts - involving contradictory information provided by the sense of sight exclusively, as opposed to **visual-kinesthetic** and **visual-proprioceptive** conflicts, related to contradictions between visual information and the perceived information about body position, velocity and stance (provided by inner ear and stretch receptors in muscles and joints). [Azuma97]

coexistence of the real and the virtual is dramatically diminished. Similar effect is observed when, while user viewpoint is relatively static, the augmentations appear to move 'on their own' due to imperfection in the position tracking (this type of registration error is referred to as jitter).

4.4.2 Vision-based Tracking Methods

A promising family of registration methods involves using computational vision methods to extract camera (or object) pose and position information directly from the video image. Such approach can be considered technologically enabling for AR in general, as relatively low-cost video cameras may be used in place of other much more expensive types of tracking equipment. In a typical video see-through AR system which already contains a camera, no additional hardware is required. Image-processing based techniques provide the benefit of an always closed feedback loop, compared to methods such as inertial tracking where errors may accumulate over time (sensor drift) leading to gross inaccuracies in tracking. Several commercially available hybrid tracking systems use visual methods for that very reason: to periodically correct tracking device output based on visual features detected in the environment (such as in the hybrid inertial-visual IS-1200 system by Intersense).

Within the vision-based tracking methods category, a number of subcategories exist. All of the vision methods are similar in that they involve extracting salient visual features (edges, corners, spatial frequencies etc) from each video frame, and analyzing them in order to determine the required pose and position information. What those methods differ in is the type of those features, their origin in the environment, and the image processing algorithms used for decisionmaking. Some approaches do not require any modifications to the environment; for example *template matching* based on previously collected imagery of the objects to be recognized and/or tracked, or *optical flowfield* analysis to estimate motion through space. Most methods however require incorporating referential salient features into the environment in a systematic manner, in the form of fiducial markers (*fiducials*). Fiducials can take a variety of forms and

shapes, like a bright light source, a predefined constellation of colorful dots, or a planar black and white pattern placed on a surface.

Of biggest relevance to the described application are systems utilizing 2D planar fiducials. A discussion of the considerations in designing the optimal planar marker system for a given application can be found in [Owen02].

4.4.3 Existing Planar Fiducial Marker Systems

Planar fiducial marker systems (encompassing the set of fiducials as well as the algorithms of identification and pose extraction) can be seen (after [Fiala04]) as an evolution of the concept of *bar code pattern* used in commerce, with added provisions for *localization*. Some examples of planar marker systems used predominantly for carrying information, are Maxicode (US Postal Services), DataMatrix, QR (shown in the top row on Fig. 4.14 [Fiala04]). All of those systems depend on high-contrast, bitonal, 2D encodings of binary data.

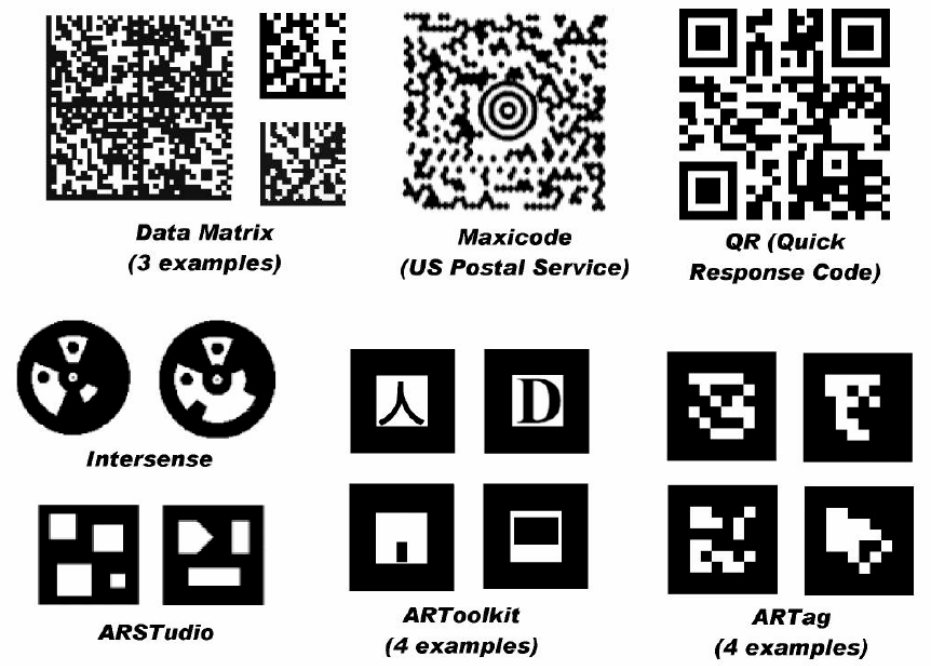


Figure 4.14. Types of planar fiducial markers used in various applications. (After [Fiala04])

Data Matrix, Maxicode, QR are industrial systems for carrying data, Intersense is used in position tracking, ARSTudio, ARToolkit and ARTag are augmented reality planar marker recognition libraries described in this section. Source: [Fiala04]

As noted in [Fiala04], in most cases the information provided by these systems about the location of the marker is insufficient for localization (determining the full 6DOF pose of the marker), making them inappropriate for AR tracking. Additionally, the wide field-of-view required by typical AR systems introduces perspective distortion, which strongly interferes with correct detection of the markers in such systems.

AR-specific planar marker systems need to address the above requirements in their design. There are some existing commercial systems such as IS-1200 by Intersense, and several systems which are open-source or freely available for research purposes, such as ARToolkit, Cybercode [Rekimoto00] and ARTag. Examples of markers used within some of those systems are presented in the bottom row on Fig 4.14. As the proceedings of AR-themed conference ISMAR indicate, such systems have already reached sufficient level of maturity to be successfully applied in research and the creation of prototypal AR systems, such as the one proposed and implemented in this Thesis.

4.4.4 Brief Description of Systems

A number of AR-specific planar marker systems was examined and evaluated with the criterion of usefulness for the intended first response application. A brief introduction of the investigated libraries is presented in the next paragraphs. The most likely contenders were ARTag and ARToolkit, and they will be discussed in greatest detail.

4.4.4.1 ARToolkit

ARToolkit, developed at Osaka University and the HITLab at University of Washington, is a software library providing a whole range of AR functionality from camera capture, through marker pose detection to geometry rendering, and is arguably the most widely used by

researchers among the considered alternatives. Some of the projects (prototype and commercial) implemented using ARToolkit are listed on the project webpage[44]. The stages of the marker detection algorithm used in ARToolkit are illustrated on Fig 4.15. The original greyscale image (a) is thresholded; pixels darker than a specific threshold parameter are connected (b). External border contours of connected regions are extracted (c). In the next step, quadrilateral region borders are identified and their corners located (d). Original contents of quadrilateral regions are unwarped to canonical square form, compared to pattern templates by convolution, and in case of successful detection overlaid with requested augmentation (e)

4.4.4.2 ARTag

ARTag ([Fiala04]) is a library originally designed to be used in conjunction with ARToolkit; however the core functionality provided by this library is useable in standalone mode. Further revisions also included a full SDK allowing for rapid development of applications without using ARToolkit. ARTag's planar marker recognition is based on digital coding theory, allowing for error correction. Unlike ARToolkit, where after determining the boundary box for a particular marker the contents are sampled and compared to bitmap templates by using convolution, ARTag after unwarping and sampling on a 6x6 grid treats the contents as codewords, calculating checksums and applying forward error correction. Another important difference is that ARTag finds quadrilaterals by linking edge pixels, as illustrated on Fig 4.16.

4.4.4.3 ARToolkitPlus

ARToolKitPlus[45] is a reworking of ARToolKit's vision code extending it with id-based marker functionality inspired by ARTag. Since this library was developed for a project involving handheld devices (see Figure 4.1), many speed optimizations were introduced for devices with less processing power. The code was also repackaged, allowing for greater use flexibility but also

[44] [http://www.hitl.washington.edu/artoolkit/]
[45] [http://studierstube.icg.tu-graz.ac.at/handheld_ar/artoolkitplus.php]

breaking compatibility with the original library. ARToolKitPlus is not meant as a complete rapid prototyping solution, as all camera capture and geometry rendering functionality have been stripped from it. The library was not tested here due to apparent lack of documentation and support for compiliing and incorporating the code in an application. Subsequent updates to the code posted on the developer's website may have changed the situation, which would make this system worthy of investigation based on the performance benefits it claims.

4.4.4.4 ARStudio

ARStudio is a technology demo of a planar marker system, rather than a complete development solution. It is acknowledged here for completeness, as it was in fact the first marker system examined by the author, which allowed for instant setting up of a demonstration AR setup showing real-time augmentations. The ARStudio demo incorporates image and video file overlaying, loading of VRML object (VRML animations are supported). It is interesting to note, that unlike in ARToolkit and ARTag, the marker detection is based on corner detection, and each marker can be considered as a local constellation of corner-based features. Such approach alows for detection even in the presence of partial occlusions (see Fig 4.17). Apparently, the work on the ARStudio has been discontinued since September 2002.

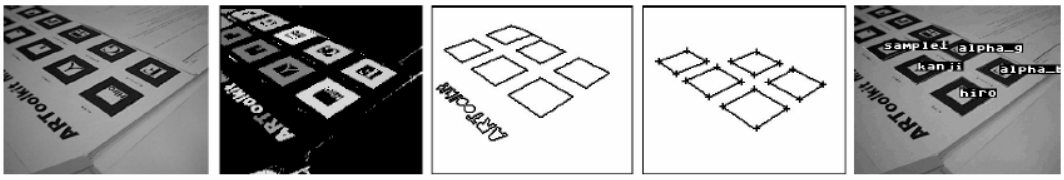


Figure 4.15. ARToolkit method of marker detection.

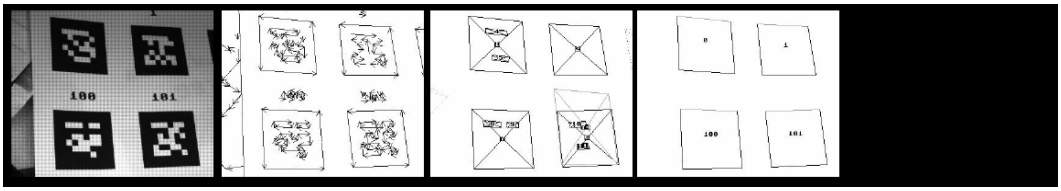


Figure 4.16. ARTag's method of marker detection.

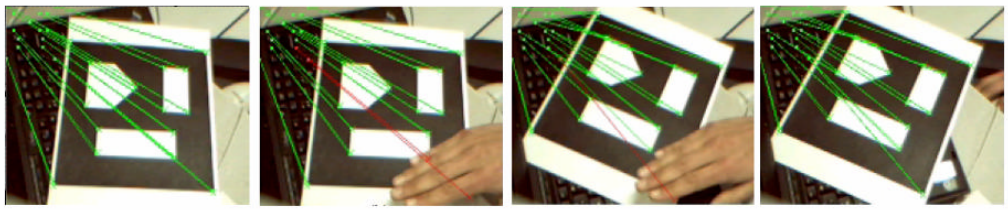


Figure 4.17. ARStudio's corner based method of marker detection and occlusion handling.

4.4.5 Specific Requirements of the First Response Application

In choosing the optimal planar marker system library for the developed first response application, specific requirements were considered. Different tradeoffs presented by the methods of marker tracking used in the investigated libraries had to be examined, as well as software library interoperation with other components of the application. The following paragraphs present a discussion of the relevant properties of each of the libraries, based on the information contained in the libraries' documentation, the author's findings and experiments involving installation, compiling and running the provided examples and attempts at integrating provided functionality into existing code. Another source of information was the systematic evaluation of the performance of ARToolkit and ARTag libraries contained within [Fiala04].

4.4.5.1 Scalability of Marker Library

The target goal of the proposed application is to annotate large environments: buildings, warehouses, campuses. This calls for an extended marker library in order to provide reasonable saturation of the physical space with fiducial markers, particularly important for the purposes of personnel tracking. Unfortunately some planar marker systems offer an unfavourable tradeoff between the unique number of marker IDs and both error rates and latency.

- ARToolkit

In ARToolkit, there is no imposed upper bound on the quantity of markers which can be handled by the system, aside from the theoretical combinational bound based on the size of the

grayscale marker templates used for detection are 16 by 16 pixels (sometimes modified to 32 by 32). In order to use a marker within ARToolkit, the fiducial pattern needs to be printed, and a pattern file needs to be created, containing pre-rotated versions of the fiducial. Rotational symmetry needs to be considered when creating a library of markers, to ensure uniqueness of the patterns regardless of marker/camera orientation. Additionally, it is recommended to optimize the pattern files for each marker by using actual camera captures of the printed patterns under various target lighting conditions and distances. Such approach significantly reduces error rates for ARToolkit, unfortunately also greatly increasing the effort needed to create and maintain a larger library of markers. After detecting and extracting the contents of a marker quadrilateral, the contents are unwarped and need to be compared to the pattern files in library, resulting in an array of confidence factor values for each of the pattern files used in comparison. The pattern file which provides the highest confidence factor (a measure of similarity to the unknown marker being recognized) corresponds to the marker id of the unknown marker, but only if the confidence factor is higher than an arbitrary confidence threshold value provided by the user (to prevent high false positive rate). Visual distinctiveness of markers in the library decreases with the increase in library size. There are no definite guidelines for creation of a large number of markers, although some approaches of systematic generation of markers by varying spatial frequencies have been proposed [Owen02].

- ARTag

On the contrary, ARTag's library of markers contains a fixed quantity of 2002 items, where half of them are black rectangles (with bit-encoded patterns inside) on white background, and the other half has inverted luminance: white boxes on black background. Each marker's pattern is a bit-encoding of the markers ID number, making generation of a marker for printout very straightforward in an automatic fashion. No bitmap templates need to be stored or maintained.

- ARToolkit+

ARToolkitPlus is advertised to support maximum 4096 unique markers, where 512 markers can be used without any performance penalty whatsoever. Though the performance tradeoffs for larger library sizes haven't been tested by us, the claims should warrant a closer inspection for future work on the device, if the marker library used in the implementation needs to be expanded for larger areas or if more precise and robust personnel tracking is required.

4.4.5.2 Marker reuse methods

The limitations of ARToolkit for the purposes of indoor location-based systems are also reported in [Reitmayr03]. In order to reduce the inevitable performance hit on the system related to increasing the size of the library, the authors propose a method of fiducial marker reuse, based on accessibility graphs of the environment. A marker is uniquely identified on the basis of currently detected fiducial ID and the ID's of markers observed immediately prior to that. It can be argued that such an approach is not feasible in this case, as it requires a degree of continuity in observing the markers placed throughout the environment for the accessibility matching to be reliable. In a likely situation when a user of the device would traverse the environment in a hurry, they would certainly skip a number of markers encountered on the way. One of the goals of the system is to give assistance when required, but not interfere with normal activity of the user. Requiring mandatory observation/registration of markers passed along the way doesn't fit this objective. Additionally, assigning new markers to specific locations is made much more complicated, as a reliable accessibility graph needs to be created for a given building - this conflicts with the goal of creating a system which allows for straightforward addition of new content by non-specialists.

A potentially more manageable method of effectively expanding the marker library size could be to use combinations of markers, for example utilizing a two marker combination in

which first marker signifies the space partition (floor, building, etc.), and the second corresponds to the content. Such methods may be also used with ARTag, if the default library size proves insufficient, for example in very tall buildings.

4.4.5.3 Marker detection errors

Some of the most important metrics describing a planar marker system are the error rates, notably:

- False negative detection rate
- Negative detection rate refers to the probability of a situation where a marker present in the environment is not detected at all.
- False positive detection rate
- Positive detection rate describes the likelihood of erroneous detection of a marker when there is no marker present.
- Inter-marker confusion
- Inter-marker confusion rate refers to the probability of a situation where one marker is erroneously recognized as another marker.

In the context of the developed application, these error rates translate to undesirable system behaviors such as problems accessing information associated with a fiducial and diminished robustness of position tracking (for a false negative), unreliable position tracking where a person is reported to be present in an area that they are not, as well as providing irrelevant misleading information to the user (high false positives, and inter-marker confusion). All of the above situations can be potentially very misleading and dangerous in an emergency situation, therefore the marker system selected should be characterized with very low detection error rates.

A comparative experiment concerning efficiency of ARTag and ARToolkit's marker recognition techniques presented in [Fiala04] shows that ARTag has many advantages over ARToolkit, notably the detection robustness is much higher in situations involving bigger number of markers in the active library. Although a mathematical estimate of the false positive error rate is provided for ARTag, experiments using various types of video footage reported no false detections at all. Unfortunately the same cannot be said about ARToolkit, where errors were quite numerous. Simply increasing the confidence threshold to reduce false positives results in increasing false negatives. Such tradeoffs are not necessary with ARTag. Similarly, intermarker confusion is much lower with ARTag. For applications in which a high number of markers is not needed, the ARTag marker set has been ordered in such a way that subsets of the library starting at marker number 1 are optimal as far as codeword distance for a given number of markers, allowing for scalability with regard to marker distinctness.

4.4.5.4 Precise Pose Registration

A major proportion of Augmented Reality systems rely on precise position and pose registration in order to provide an illusion of realism for the virtual augmentations. This is certainly desireable for applications where such illusion is required, for example in a tangible user interface, or a military/medical/emergency training simulation, and usually is achieved inside a limited and controlled environment within which precise tracking can be achieved, such as a lab room equipped with sophisticated trackers. While vision-based methods do enable a more 'roaming' approach to augmenting reality, as fiducial markers can be easily and cheaply produced and placed within any environment, they are not without some limitations. An indoor location-based application based solely on visual methods, such as the one proposed here, cannot offer continuous camera position/pose tracking unless at any given time there is at least one fiducial in the line of sight, large enough or close enough and properly oriented with respect to the camera. The smallest and largest distances of a fiducial from the camera at which the fiducial is still

correctly detected is a function of several variables, including the physical size of the marker, camera focal length, video capture resolution, lighting, type of marker content etc. For ARTag, the minimum pixel width of the marker in the captured video frame needs to be approx. 13 pixels at a 640x480 resolution for detection to be possible at all, while a high detection rate is achieved starting from approx 17 pixel widths. For ARToolkit it is harder to estimate the minimum, as it is dependent on the characteristics of the visual pattern used for the marker, and the arbitrary detection threshold selected by the user.

For precise pose detection however, even more information is necessary than for straightforward detection. Thus in order to be able to provide, for example, a jitter-free 'X-ray' view of the building correctly overlaid on camera imagery, the fiducial markers would need to be placed in large amounts throughout the environment, and would need to be relatively large in size to be successfully tracked from a distance. From author's observations, when augmenting a fiducial with a 3D object, most realistic results are obtained when the scale of the object is small relative to the marker, as presence of jitter and pose inaccuracies are not very noticeable in such case. For objects larger than the marker or located at a distance from the marker's center, even subpixel-scale motion of the fiducial leads to very annoying large displacements of the augmented virtual object which can be disorienting and annoying.

Therefore early on in the design process the notion of providing functionality strongly dependent on precise marker tracking was abandoned in favor of a simplified but more robust approach in which the marker just needs to be detected to provide location tracking. The position of the marker's center is used as a control for browsing the available information content, as illustrated in the storyboards and screenshots presented in Chapter 2. In such approach, there is no need for pose registration at all (just position is detected), therefore the requirements on fiducial markers' size, video capture resolution and maximum distance from marker could be relaxed.

4.4.5.5 Low Latency Requirement

As noted in Section 4.3.2, the overall system end-to-end latency is crucial for an AR device. Marker detection computations need to be performed for each incoming camera video frame, thus significantly contributing to overall delays.

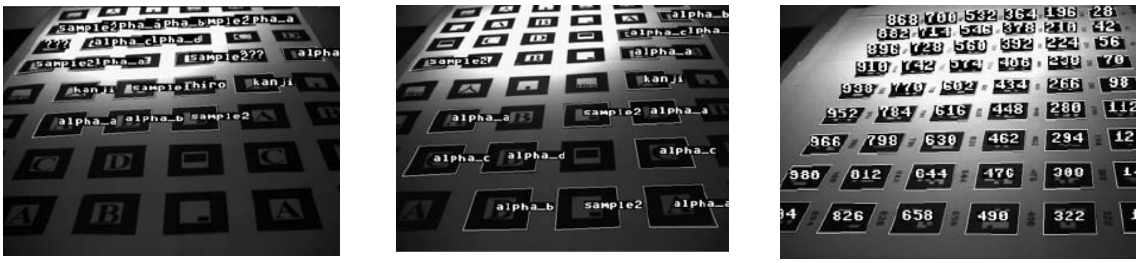
The detection process for a single ARTag quadrilateral takes exactly the same time regardless how many of the 2002 markers are actually used by the application. ARToolkit processing is faster for small pattern library sizes, ARTag's performance in half resolution mode is equivalent to half-resolution mode ARToolkit loaded with a library of 30 markers, while the performance of half resolution ARToolkit loaded with 200 markers corresponds to ARTag in high resolution mode.

4.4.5.6 Robustness

4.4.5.6.1 Lighting Conditions

One of the apparent weaknesses of ARToolkit, as pointed out in [Fiala04] is the sensitivity of the contour detection and marker recognition technique to the lighting conditions in the environment. ARToolkit's grayscale thresholding method requires picking an arbitrary value which is then used for binarization of the image. Unfortunately, in naturally occuring lighting conditions such thresholding can be too restrictive, resulting in failed detection of markers located in shadowed areas, or areas with gradation of light intensity, for example within the penumbra of a spotlight.

Since ARTag uses a localized edge detection scheme rather than global thresholding, it is not suprising that it performs better in more challenging lighting conditions. See Fig, 4.18 for an illustration of ARTag's advantage.



a) ARToolkit, higher binarization threshold value

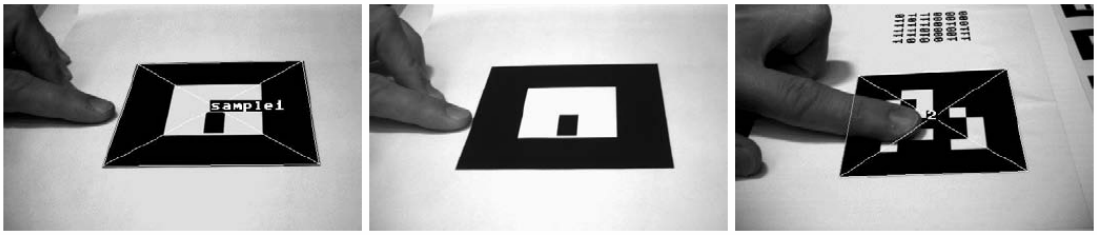
b) ARToolkit, lower binarization threshold value

c) ARTag

Figure 4.18. Comparison of marker detection susceptibility to local lighting condition variations. Overlays with name or ID of marker indicate correct detection.

4.4.5.6.2 Partial Marker Occlusion

Of all considered marker tracking libraries, ARToolkit is the most susceptible to false negative detections due to marker occlusion, as it doesn't incorporate any mechanisms to compensate for them. Even a small occlusion of the border prevents marker detection - see Fig. 4.19b. This also means that the whole contour of the marker has to be well within the camera frame for successfull detection. On the other hand ARTag, ARToolkitPlus and ARStudio provide methods for handling partial occlusion. ARTag's occlusion handling is possible due to error correcting properties of the code utilized in the encoding of the marker IDs in the patterns (also true for ARToolkitPlus, but not tested) as well as the particulars of the quadrilateral detection employed in ARTag (the border can be partially occluded, unlike in ARToolkit). The advantage of this approach over ARToolkit's is illustrated on Fig 4.19c. This also means that within the implemented system, indication of a marker present in the environment will be displayed faster that with ARToolkit, before the marker has completely entered the user's field of view, and remain visible longer when the marker moves out of the display. This could be considered an important feature from the standpoint of situation awareness.



a) ARToolkit, before occlusion

b) ARToolkit, occluded border, failed detection

c) ARTag, successful detection despite partial occlusion of border and several bits of the pattern

Figure 4.19. Effects of marker occlusion

As a side note, ARStudio's detection technique appears to be geared particularly toward tangible reality interactions, in which the user hand occludes the marker to a greater extent [Malik02], and special care has been given to ensure the system performs very well with respect to occlusion. The author's experiences with the technology demo confirm that notion.

4.4.5.6.3 Camera Focus

One definite observed advantage of ARToolkit over ARTag is higher immunity to out-of-focus imagery, with regard to both negative error detection and presence of jitter in pose registration. This is due to the specifics of the quadrilateral region detection methods of the two libraries: ARTag's edge detection is based on luminance gradients.

4.4.5.7 Ease of Library Integration / Dependencies

Prototypal systems such as the one developed by us are often by necessity strongly characterized by the reuse of existing code. Bringing together vision-based methods, camera video capture and processing, 3d and 2d graphical overlays combined with video, databases and input sensors would be a downright impossible task under the circumstances (time restriction, single developer, low budget), if all the code handling low-level operations such as graphical

rendering, image processing, networking etc. had to be created from scratch. The enabling factor, allowing for rapid development of the application, was the high availability of quality, dependable open-source libraries abstracting a large proportion of the abovementioned functionality. Part of the challenge with such 'standing on the shoulders of Giants' approach however was to integrate code which originated from different developers, who also used various third-party lower level libraries (graphical libraries, video capture, image processing etc). Conflicting or redundant dependencies needed to be addressed.

As described in Section 4.3, early on in the design process it was established that the developed application will require relatively sophisticated graphical features for implementing the GUI, leading to the selection of Ogre3D graphical visualization engine for the implementation. One of the benefits of this was the potential platform independence, as Ogre3D can be used in OpenGL mode, making it potentially much easier to port the application to a Unix/Linux based platform, if required in future. However, using this fully featured engine to full extent enforced a complete separation of the graphical rendering from the marker tracking and augmentation, and required the shaping of the whole structure of the applications code around the Ogre3D framework. This strongly influenced the final choice of planar marker library.

- ARToolkit

In order to build ARToolkit for one of the supported platforms, necessary system prerequisites need to be fulfilled. For a Windows build, several runtimes and SDKs are needed such as GLUT (OpenGL wrapper), DirectX Runtime, and some additional/optional components for video capture and VRML support. This makes ARToolkit easy to use out-of-the box and powerful, but also potentially more challenging to integrate with the rendering engine. Since graphical rendering in the developed system was to be performed solely by Ogre3D, and camera captured video needed to be handled from within the renderer as well, ARToolkit's extensive functionality would be underutilized,. Since the applications requirements with regard to marker

tracking were very specific and limited in scope, extracting only what was required could prove potentially very time-consuming.

Fig. 4.20 illustrates the ARToolkit library architecture and its dependency on other libraries.

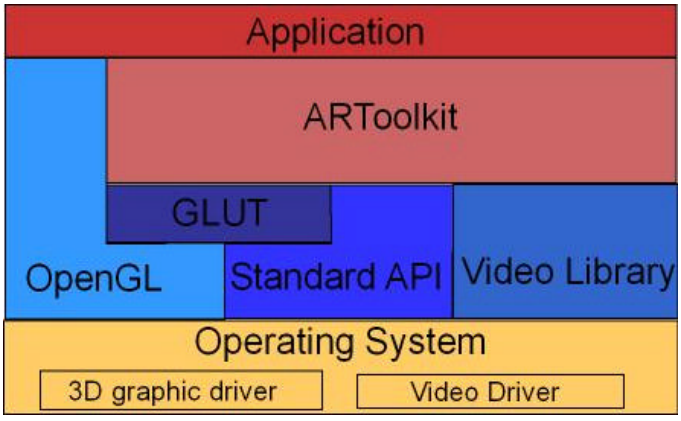


Figure 4.20. ARToolkit software architecture and dependencies.
(Source: [http://www.hitl.washington.edu/artoolkit/documentation/devframework.htm])

- ARTag

ARTag was originally designed to be used only in conjunction with ARToolkit. More recent addition released by the developer in December 2005 provided an SDK allowing to use ARTag in a standalone fashion. Neither extending ARToolkit with ARTag, nor using the extended SDK suited the needs of the project well. However, the core marker detection functionality provided by this library can be used separately in the form of a stripped-down library. A simplified interface provides simple marker detection without pose/position estimation or camera space transformations, which was exactly what was required and ultimately allowed for a very straightforward integration of this library into application code.

Overall ARTag was found to provide the best solution with regard to flexibility of use of all the investigated libraries.

4.4.5.8 Ease of use / Documentation / Support

- ARToolkit

ARToolkit is undoubtedly the most widely known solution available for researchers, recognized and popularized at the ISMAR conference, actively developed and supported by a growing community of contributors. The documentation reflects this fact, providing extensive information about creating and using new markers as well as creating custom applications based on the ARToolkit code. Sample programs have been provided.

- ARTag

The stripped-down subset of ARTag chosen to be utilized in the project takes a video frame as input and returns an array of detected marker IDs and their corner coordinates. As noted before, such functionality was exactly what was required for the purposes of the project. Code is well commented, and usage is explained in the provided documentation and by means of sample programs. The library is continually updated, revisions are occasionally posted on the internet. The library author was contacted with a specific problem and provided assistance.

- ARToolkit+

As noted by the developers of ARToolkit+, it was developed internally for the purposes of a handheld device prototyping, therefore documentation is targeted at experienced AR system developers well versed with the workings of the original ARToolkit library.

4.4.6 Conclusion

4.4.6.1 Choice of the ARTag system

The ARTag planar marker library matched the application's requirements more closely than ARToolkit, which was the main contender (widely considered the "state-of-art" system for research purposes). Reports from the literature as well as the author's observations confirmed that the weaknesses of ARToolkit (larger error rates, high sensitivity to lighting conditions, performance reduction when working with large library of markers) might pose severe problems

for this particular application, while the chief strengths, such as providing a complete prototyping solution, and integrated rendering and camera space transformations, were not relevant to the application needs, and furthermore, they could potentially complicate software library integration. Integrating ARTag library into the application's code proved straightforward and the performance of detection algorithm was satisfactory, though testing revealed a stability issue described below.

4.4.6.2 Stability Issue

The ARTag library code subset utilized in the project was originally available only as a single-threaded static library, which conflicted with the dynamic multithreaded library runtime enforced by other components of the project. Assistance from Mr. Fiala, the code author was obtained, in the form of a multithreaded statically compiled library (available as artag2_multithread.lib binary on the CD), which can be used alongside the other components, with some reservations. The optimal form of the library would be multithreaded DLL versions of the library in release and debug variants – this is the case for most other libraries used in the project. An alternative would be to compile the library statically into the code, however in the case of ARTag the source code is not made freely available by the author. Since DLL builds of the library were unavailable at the time of development, the multithreaded static version was used. This introduced an issue into the development process, which was that the release build of the project software is not stable in current state – the ARTag function call causes a crash. Debug configuration was therefore used throughout the development, unfortunately reducing the system's performance since debug versions of libraries are always slower. In this configuration the system is generally stable. An additional issue was observed though, which may or may not be related – when the white border around the black fiducial is too thin, the ARTag processing again causes a failure. An investigation of the above effects and further collaboration with the ARTag code author would be advised, as these issues detract from the system's performance most heavily from all the issues encountered.

4.5 Network and Database Module

Centralized responder position and asset tracking require provision of network database connectivity. This section describes the current implementation and points out directions for future development.

4.5.1 Implementation

In the current implementation, mobile responder stations transmit database queries and receive results directly to/from the server software over the available wireless data communications channel (see Fig. 4.21). The server software is installed as a system service on a 'Command Center' computer placed outside the danger zone but within the range of radio frequency wireless network (IEEE 802.11x "WiFi" used in the prototype setup).

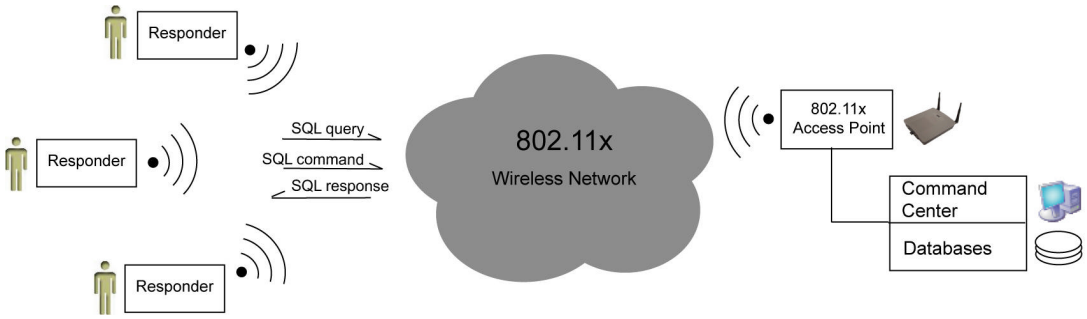


Figure 4.21. Database connectivity in the prototype implementation.

4.5.1.1 Database Mechanism

For the purposes of the prototype, MySql Server v.5.0 software was used.

MySql Server is distributed by MySql AB under dual license (GPL and commercial), and is an attractive solution for a significant section of the database market, offering stability and relatively high speed of query handling comparable or exceeding other open-source databases (PostgreSQL) as well as commercial products by Microsoft and ORACLE. MySQL offers high availability of programming interfaces for a wide range of environments and platforms, and

ACID[46] compliance. However, the speed and simplicity of MySql comes at the expense of limited compliance with MySQL standards functionality. Most notably, nested queries are not supported, so MySQL would be inappropriate for a system requiring advanced relational database operations. On the other hand, network connections are handled very fast, making it a very suitable database for internet based systems or multiple user systems such as the one described here.

The structure of the MySQL fiducial description and responder tracking database including a description of its fields can be found in Appendix B. During development, database structure initialization and data entry was performed utilizing a third-party MySQL client program, SQLYog[47].

4.5.2 Further Work Directions

The database functionality of the evolutionary prototype in its current stage embodies only a slice of the total envisioned functionality of the final system as described in Section 2.2 of Chapter 2. Furthermore, the existing implementation is not a mature solution in terms of network bandwith utilization and robustness. The following sections contain suggestions for future expansion and refinement of the system, and hint at some engineering issues

4.5.2.1 Database Proxy/Custom Protocol

As the prototype is refined and expanded, it would be advisable to implement a proxy application acting as a gateway between the mobile responder systems on the network and the database server itself (see Figure 4.21). The proxy, coupled with a custom network protocol employing compression and/or error correction, would allow to reduce network bandwidth requirements and latency, and increase robustness of the final system. Given the rather simple

[46] ACID (Atomicity, Consistency, Isolation, Durability) - a keyword referring to a set of database features which ensure continuous integrity of stored data. Such features are required in most business applications such as bank or online monetary transactions etc.
[47] available at [www.sqlyog.com].

database connectivity requirements, employing a simpler file-based database instead of a SQL-based solution is another possibility. However, the drawback of such approach is that it would prohibit leveraging on the interoperability, functionality, scalability and platform-independence of ready to use SQL-based databases. Depending on final systems needs, functionality such as web interfaces, user authorisation, database transaction logging and database replication as well as specialized client software for external database access would have to be developed internally.

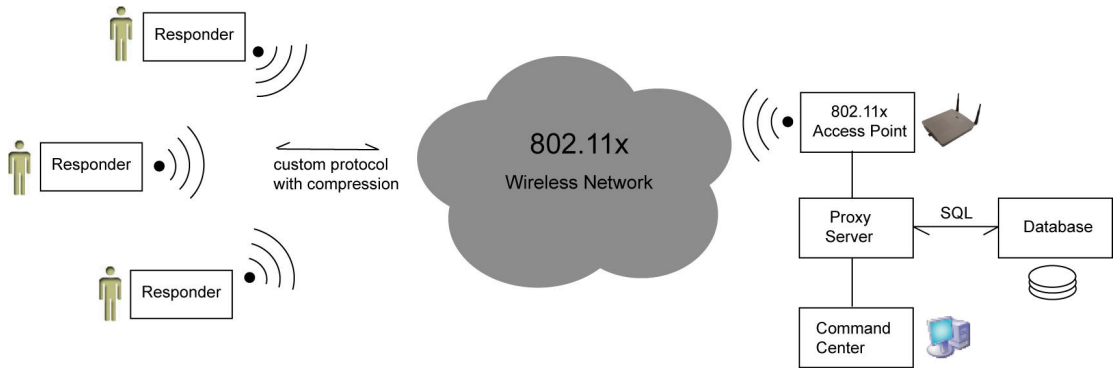


Figure 4.22. Database proxy/front-end server.

4.5.2.2 Mobile Device Database Initialization

The current prototype relies on the multimedia content data already residing on the hard drive of the mobile computer used by the responder. Bitmap imagery for maps, blueprints and environmental cubemaps, Ogre3D object files for 3d building maps and device schematics, font files and necessary resource configuration scripts need to be placed in the appropriate directory within the 'media' subdirectory of the application's installation path prior to deployment. The content cannot therefore be changed at runtime. However, the assignment of fiducials to the content objects, label descriptions and other parameters can be changed when the system is operating, as this information is stored in the database and refreshed in predefined intervals.

It can be expected that the data volume of multimedia content objects for a densely annotated building can be relatively large. Since the area affected by the emergency is not known

in advance of the emergency, it is imperative for further work on the prototype to include developing a mechanism for quickly transferring the required sets of multimedia and fiducial descriptive content to the responder's mobile computers. This initialization stage should be preferrably performed over a wired networking link or using some form of removeable media while the first response team prepares for being dispatched to the site of emergency. In the network database variant, the same database mechanism can be used as for fiducial marker description database – for example, BLOB type SQL database fields can be used to store (preferrably archived) content data files within the same SQL database alongside the fiducial descriptive information. The required databases are downloaded to the first response mobile device from a centralized repository over a wired database connection before deployment.

CHAPTER V

SUMMARY OF RESULTS AND FUTURE DIRECTIONS

The majority of the development of the first iteration of the Augmented Reality first response device prototype took place over the time of one year between June 2005 and June 2006, with a preliminary design, conceptualization and technology research stage of five months prior to that. The goal of this phase of the project, which was to deliver a functional demonstration of an Augmented Reality first response system based on commercial off-the-shelf hardware components and custom-written software, was achieved successfully.

In this chapter, the current development state of the prototype is summarized and several issues and areas which require further work are outlined.

5.1 Project Deliverables

5.1.1 Prototype Hardware Deliverable

Currently the prototype is available as a proof-of-concept setup utilizing a laptop computer, an I-glasses HMD display, a firewire camera, a modified P5 glove device, and WiFi network card, as described in detail in Sections 3.1 – 3.5 of Chapter 3. All utilized components are products available off-the-shelf, consumer range priced. The approximate total cost of all components is 3500$ based on prices at of the time of purchase.

5.1.2 Prototype Software

The prototype software code was developed in C++ for the Windows platform, the source code and binaries are located on the enclosed CD appendix. It implements the functionality of the environment annotation reader device, providing an Augmented Reality GUI (described in Section 2.1.2) which allows the user to access the data assigned to fiducial markers placed in the environment, and visualize the associated content interactively. Basic network database support is included (see Section 4.5.1 and Appendix B). The prototype leverages on a number of third-party software libraries, such as Ogre3D (Sec. 4.3), ARTag (Sec. 4.4), OpenCV (Sec. 4.3.4.7, Sec 3.3.1.3), MySQL C API (Sec. 4.3.5). All the necessary third party code and binaries are included on the CD.

5.2 Future Development

5.2.1 Issues to Resolve

Although the system in its current state is functional, two software development problems were encountered during prototype development, which need to be promptly fixed in order to significantly increase the system performance and reliability. The first one is the incompatibility of the Fire-I camera with the OpenCV camera capture routines, described in Section 3.3.1.3 (other cameras work well). The second, more serious issue is the stability of the ARTag simple fiducial recognition library, described in Section 4.4.6.2.

Another important bit of functionality missing in the current implementation is the P5 glove finger sensor calibration. Presently, calibration can be performed using a third-party configuration application, as described in Appendix C. This functionality should be incorporated into the application instead, as this would simplify the deployment and allow for re-calibration on the go if the sensors change their properties (for example due to increased temperature).

5.2.2 Suggested Areas of Concentration

The current implementation of the first response device still has several shortcomings which limit the degree to which the prototype can be meaningfully tested in the target environment. Some of them are indicated in the sections below.

5.2.2.1 Wearability

Provided that external battery packs and appropriate mounting gear (backpack or similar) are available, it would be possible to operate the device in untethered mode. The available operating time would be limited in such setup (40 minutes), and the ergonomics would leave much to be desired. Therefore, in future iterations of the design the following areas need to be investigated and enhanced:

5.2.2.1.1 Power Consumption and Robustness

The mobile computer used for the development of the prototype, Fujitsu Amilo-D 1840 (Section 3.1.1), while offering good processing capability, was not engineered to provide long battery life, or be used extensively in a wearable configuration in adverse environmental conditions. A more suitable computing platform, such as for example the Thermite TVC described in section 3.1.2, should be considered in future iterations.

5.2.2.1.2 Form Factor

In addition to a more suitable wearable computer, the form factor of the peripherals used in the system should ultimately be improved upon. The current prototype utilizes low cost components which, while adequate for proof-of-concept purposes, would not be suitable for deployment in emergency situations.

5.2.2.1.2.1 HMD

The i-Glasses PC3D head mounted display used for the prototype (Sec 3.2.2) significantly occludes the periphery of vision, thus negatively affecting the user's situational awareness. Integration with existing first response head-worn gear could pose additional

problems. A more innocuous and less invasive option is discussed in Section 3.2.4, where near-eye displays are described. Such displays have been recently made available in the consumer market, alongside more advanced, rugged, and expensive solutions useable in more demanding scenarios.

5.2.2.1.2.2 Fingerbend Sensors

The modified P5 glove (Sec. 3.4.2.1) doesn't provide much flexibility for integration with first responder's protective gear (e.g. rubber gloves), and a significant portion of its footprint is used for electronics which are no longer used after the modification. Provided the finger bending sensor based interaction is determined to be a successful technique through user evaluation study, the next suggested step could be investigating the possibility of embedding simplified fingerbend sensors directly in first responder's protective gloves. According to rationale presented in Sec 3.4.2.3, 3.4.2.4, three sensors should be sufficient. Providing wireless connectivity would be a logical next step for an input device used in this fashion, simplifying donning of the wearable system by the responder.

5.2.2.1.2.3 Camera

Since the head mounted camera needs to be carefully placed relative to the user's original line of sight, the exact requirements for the camera hardware can be determined only in conjunction with the information about the specifics of the head-mounted display and the protective head gear (helmet, breathing apparatus) to be used. The Fire-I IEEE1394 camera used currently (Section 3.3.1) provides an adequate starting point in terms of quality and footprint. However, if an even smaller footprint is required, the Reader should refer to the suggestions contained in Section 3.3.3.1.

5.2.2.2 Reliable Ad-hoc Networking

At present, the prototype supports IEEE 802.11 connections between the responder device and the network database. The limitations of this technology are discussed at length in

Section 3.5. Several possible alternatives are indicated, including PacketHop (see Sec. 3.5.2.2.1) and Motorola's Mesh Enabled Architecture (see Sec. 3.5.2.3.1).

5.2.2.3 Testing and Feedback

All aspects of the prototype design, both in hardware and software, need to be thoroughly tested and evaluated by the target group of users and experts. The feedback obtained thereby should be the fundamental basis for following iterations of the design, a process ultimately leading to successful deployment of the device in real world situations. Since technology advances and a growing knowledge base are gained through hands-on usage, this fact should warrant continuing opportunities for improvement.

5.2.3 Project Scope Expansion

As detailed in Sections 2.2.1 through 2.2.3, the prototype system will ultimately function within a broader context, including infrastructure and tools for authoring, distributing and maintaining the annotation database information. Therefore, when the first response device matures sufficiently in future iterations of the prototype, it would be adviseable to gradually define, develop, and incorporate that encompassing infrastructure into the project. Modularity and open-endedness should be a high priority in these stages, as technological flexibility will facilitate the evolution of the prototype into a complete solution.

5.3 Final Conclusion

Applications related to first response are notoriously difficult as a design problem, due to the very steep requirements with regard to robustness, reliability and resilience. Additionally, a slew of human factors issues need to be addressed, given the high stakes, cognitive challenges and complexity of emergency scenarios. While not constituting a mature solution useable in the target environment at this stage of development, the project deliverables demonstrate that the

proposed approach is viable, providing a suitable starting point for iterative design involving feedback from the intended target users of the system.

BIBLIOGRAPHY

Chapter I References

[Azuma97] "A Survey of Augmented Reality" R. Azuma; Presence: Teleoperators and Virtual Environments. vol. 6, no. 4, Aug. 1997, pp. 355-385.

[Azuma01] "Recent Advances in Augmented Reality" Ronald Azuma et al.; IEEE Computer Graphics and Applications, v.21 n.6, p.34-47, November 2001.

[Azuma04] "Overview of augmented reality" Ronald Azuma; Proceedings of the conference on SIGGRAPH 2004 course notes GRAPH '04, August 2004.

[Neumann98] "Cognitive, Performance, and Systems Issues for Augmented Reality Applications in Manufacturing and Maintenance" Ulrich Neumann, Anthony Majoros; Proc. IEEE Virtual Reality Annual International Symposium, 1998.

[CLS86] "Fire and Smoke: Understanding the Hazards" Comission on Life Sciences National Research Counci1 National Academy Press, Washington, D.C. 1986.

[Zinn77] "Investigation of smoke particulates generated during the thermal degradation of natural and synthetic materials" B. T. Zinn et al.; Fire Safety Journal Volume 1, Issue 1 , Pages 23-36. March 1977.

[Lampe04] "A Ubiquitous Computing environment for aircraft maintenance." Matthias Lampe, Martin Strassner, Elgar Fleisch; Symposium on Applied Computing Proceedings of the 2004 ACM symposium on Applied computing.

[Romero03] "Mixed reality hypermedia: HyperReal: a hypermedia model for mixed reality" Luis Romero, Nuno Correia; Proceedings of the fourteenth ACM conference on Hypertext and hypermedia, August 2003.

[Milgram94] "Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum." Paul Milgram et al.; Proc.SPIE: Telemanipulator and Telepresence Technologies, vol. 2351, Society of Photo-Optical Instrumentation Engineers, Bellingham, Wash., 1994.

[Marinelli05] "When Worlds Converge: The Coming Supernova of Entertainment and Education", Don Marinelli; Publications from the Forum for the Future of Higher Education (2005).

[SIGCHI99] "Bridging physical and virtual worlds with electronic tags" Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit. Year of Publication: 1999.

[Weiser91] "The Computer for the Twenty-First Century" Mark Weiser; Scientific American, September 1991. pp. 94-104.

[Romer04] "Smart Identification Frameworks for Ubiquitous Computing Applications" Kay Romer, Thomas Schoch and Friedmann Mattern; Wireless Networks, November 2004.

[Milgram94] "A taxonomy of mixed reality visual displays" Paul Milgram, Fumio Kishino; IEICE Transactions on Information Systems, Vol E77-D, No.12 December 1994.

[Lorincz05] "MoteTrack: A Robust, Decentralized Approach to RF-Based Location Tracking", Konrad Lorincz and Matt Welsh; Proceedings of the International Workshop on Location and Context-Awareness (LoCA 2005) at Pervasive 2005, May 2005.

[Priyantha00] "The Cricket Location-Support System." Priyantha, N., Chakraborty, A., and Balakrishnan, H.; In Proc. 6th ACM MOBICOM Conf. August 2000.

[Wong05] "An Analysis of the Human Odometer" U. Wong, C. Lyons, and S. Thayer; Tech. report CMU-RI-TR-05-47, Robotics Institute, Carnegie Mellon University, September, 2005.

[Kourogi03] "Personal Positioning based on Walking Locomotion Analysis with Self-Contained Sensors and a Wearable Camera" Masakatsu Kourogi, Takeshi Kurata; Proc. ISMAR03, 2003.

[Randell03] "Personal Position Measurement Using Dead Reckoning" Cliff Randell, Chris Djiallis, Henk Muller; Proceedings of Seventh International Symposium on Wearable Computers, 2003.

[Manesis05] "Optimising navigation: Survey of position location techniques in mobile systems" Thanos Manesis, Nikolaos Avouris; Proceedings of the 7th international conference on Human computer interaction with mobile devices & services MobileHCI '05, September 2005.

[Miller06] "Indoor Navigation for First Responders: A Feasibility Study" Leonard E. Miller, Wireless Communication Technologies Group; National Institute of Standards and Technology, February 2006.

[McKinsey02] "Increasing FDNY's preparedness" McKinsey & Company; [http://www.nyc.gov/html/fdny/html/mck report/index.shtml] (2002).

[Fiala04] "ARTag Revision 1, A Fiducial Marker System Using Digital Techniques," Fiala, M.; NRC/ERB-1117. November 24, 2004.

[Wilson05] "Design of Monocular Head-Mounted Displays for Increased Indoor Firefighting Safety and Efficiency", Proc. of SPIE Vol. 5800, 2005.

Chapter II References

[Chen95] "QuickTime VR: an image-based approach to virtual environment navigation" Shenchang Eric Chen; Proceedings of the 22nd annual conference on Computer graphics and interactive techniques, September 1995.

[Foote00] "FlyCam: practical panoramic video" Jonathan Foote, Don Kimber; Proceedings of the eighth ACM international conference on Multimedia, October 2000.

[Sun01] "Exploiting Video: Panoramic video capturing and compressed domain virtual camera control." Xinding Sun, Jonathan Foote, Don Kimber, B. S. Manjunath; Proceedings of the ninth ACM international conference on Multimedia, October 2001.

[Agarwala05] "Capturing reality II: Panoramic video textures" Aseem Agarwala, Ke Colin Zheng, Chris Pal, Maneesh Agrawala, Michael Cohen, Brian Curless, David Salesin, Richard Szeliski; ACM Transactions on Graphics (TOG), July 2005.

[Baudisch04] "Multiblending: Displaying Overlapping Windows Simultaneously Without the Drawbacks of Alpha Blending" Patrick Baudisch, Carl Gutwin; (2004).

Chapter III References

[Vlahakis03] "Design and application of the LIFEPLUS Augmented Reality System for continuous, context-sensitive guided tours of indoor and outdoor cultural sites and museums" V. Vlahakis, T. Pliakas, A. Demiris, and N. Ioannidis; VAST 2003, EG Workshop Proceedings, 2003.

[Siegl03] "Mobile AR Setups" Hannes Siegl, Harald Ganster and Axel Pinz; Proceedings of the 27th Workshop of the ¨ OAGM/AAPR, Laxenburg, 2003.

[Ware] "Information Visualization: Perception for Design, 2nd Edition " Colin Ware; Morgan Kaufmann 2004.

[Kasai00] "A Forgettable near Eye Display", Ichiro Kasai, Yasushi Tanijiri, Takeshi Endo, Hiroaki Ueda; Proceedings of the Fourth International Symposium on Wearable Computers (ISWC'00), 2000.

[Christian00] "A comparison of voice controlled and mouse controlled web browsing" Kevin Christian, Bill Kules, Ben Shneiderman, Adel Youssef; Proceedings of the fourth international ACM conference on Assistive technologies Publisher: ACM Press, November 2000.

[Piekarski02] “The Tinmith system: demonstrating new techniques for mobile augmented reality modeling” Wayne Piekarski, Bruce H. Thomas; Australian Computer Science Communications , Third Australasian conference on User interfaces - Volume 7 CRPITS '02, Volume 24 Issue 4, January 2002.

[Piekarski03] “Interactive augmented reality techniques for construction at a distance of 3D geometry” Wayne Piekarski, Bruce H. Thomas; Proceedings of the workshop on Virtual environments 2003 EGVE '03 Publisher: ACM Press May 2003.

[Olwal03] “SenseShapes: Using Statistical Geometry for Object Selection in a Multimodal Augmented Reality System.” Olwal, A., Benko, H., Feiner, S; In Proceedings of The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR 2003). Tokyo, Japan. p. 300–301. October 2003.

[Sturman94] “A survey of glove-based input” Sturman, DJ & Zeltzer, D.; IEEE Computer Graphics & Applications 14(1), pp. 30--39, January 1994.

[Baker02] "An outsider's view of MANET," F. Baker; Internet Engineering Task Force document, 17 March 2002.

[Frodigh00] "Wireless Ad Hoc Networking: The Art of Networking without a Network," M. Frodigh, et al,; Ericsson Review, No. 4, 2000.

[IEEE802.11] “Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification” IEEE Std. 802.11, 1999.

[Johnson03] "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks (DSR)", Johnson, Maltz, Hu; April 2003.

[RFC3561] “Ad hoc On-Demand Distance Vector (AODV) Routing”, Request-for-comments, July 2003.

[NIST_DTFR1] http://w3.antd.nist.gov/comm_net_ps.shtml

[NIST_DTFR2] http://w3.antd.nist.gov/wctg/NDTFR/dtfr.html

[Motorola05] “Mesh Networks: A Revolution in Wireless Connectivity”. Motorola Technology Position Paper, 2005.

Chapter IV References

[Buschman96] ”Pattern Oriented Software Architecture: A System of Patterns” Frank Buschman, Regine Meunier, Hans Rohnert, Peter Sommerlad; 1996.

[Singhal99] “Networked Virtual Environments – Design and Implementation” Sandeep Singhal, Michael Zyda, 1999.

[Schneiderman98] “Designing the user interface: Strategies for effective human-computer interaction.” Schneiderman, B.; Addison Wesley Longman Inc. 1998.

[Wloka95] “Lag in multiprocessor VR” Wloka, M.; Presence: Teleoperators and Virtual Environments 4(1):50-63, Winter 1995.

[Clark76] “Hierarchical geometric models for visible surface algorithms.” Clark, J.; Communications of the ACM 19(10):547-554, October 1976.

APPENDICES

APPENDIX A. SOURCE CODE CD

A.1 Description of CD contents

On the enclosed CD the Reader will find the electronic version of this Thesis, alongside all the binaries, source code and resources needed to proceed with the prototype software testing and/or development. An overview of the directory structure and directory contents is presented in the following Tables A1.1 and A1.2. Further information about contents of a particular directory can be usually found in a ReadmeAR.txt file, if one is present in the directory.

Directory	Description
\MSciThesis	Contains this Thesis in PDF format.
\Development	Contains all files needed for testing and development of the prototype's software compressed in a single .zip archive (*ARPrototype.zip*)
\Media	Contains additional media, including a video presentation of the system's usage.

Table A1.1. Directory contents of the enclosed CD

The directory structure contained within the *ARPrototype.zip* file is shown in Table A1.2 on the next page.

Directory	Description
\Binaries	The prototype executables and all necessary runtime library files.
\debug	Output directory for the linker, when Debug Configuration is used. Contains all necessary DLL files (in debug versions) and configuration files for Ogre3D and the prototype application. *ARPrototype.exe* is the prototype mobile responder application executable, *ARPrototype2.exe* is the sample Command Center viewer application build.
\release	Release Configuration linker output directory.
\Dependencies	Third-party software libraries used by the application's code. Software development kits, library and header files, in some cases source code and/or sample code examples are provided in individual directories listed below.
\OgreSDK	Ogre3D Software Development Kit v1.0.2 [Azathoth]. Please see the following references: *\Dependencies\OgreSDK\docs\manual\manual_toc.html* *\Dependencies\OgreSDK\docs\ReadMe.html*
\OpenCV	Intel's open-source image processing and computer vision, version 5 Beta. Please refer to documentation found in *\Dependencies\OpenCV\docs* directory for help on getting started with OpenCV
\STLport	Contains a standards-compliant Standard Template Library (v.4.6.2) implementation required by Ogre3D. The version of STL provided with Visual C++ 6.0 is not compatible.
\P5SDK	P5 Glove Software development kit 2.0 related files
\Ptypes-1.8.3	Files for Hovik Melikyan's helper library Portable Types version 1.8.3
\MySQL	MySQL database connectivity C library.
\DesignFiles	Contains various intermediary design files created during development of the prototype, including graphical files in Adobe Photoshop format.
\HelperApps	Helper applications used during development of the code. Please see the ReadmeAR.txt file for more information
\OgreResources	This directory contains resource files used by the Ogre3D engine. The relative path to this directory must be specified in a *resources.cfg* file located in the same directory as the Ogre3D-based executable (in this case in the Binaries\debug and Binaries\release directory). The resources include images, 3d mesh object files, and script files defining runtime parameters for many of the objects used by the application.
\ProjectFiles	Contains Visual Studio 6.0 workspace files and source code for the prototype application. The mobile responder application workspace file is: *\ProjectFiles\ARPrototype\ARPrototype.dsw* Please see Section A.2. of this appendix for few simple additional steps needed in order to compile and link the project sources.
\Tutorials	Contains a selection of tutorials for the Ogre3D engine provided by the community. This is a good place to start when learning how to utilize Ogre's powerful functionality.

Table A1.2. Directory structure and contents of the enclosed CD

A.2 Setting up the development environment

Although no installer is provided for the prototype's code, the process is straightforward and simple, and there is no need to install any of the dependencies (SDKs) individually, as all necessary files are provided. It is assumed here that Windows XP operating system is used and Microsoft Visual C++ 6.0 software is already present on the system.

The contents of file "***ARPrototype.zip***" (found on the CD in "Development" directory) need to be unarchived into a new directory created by the user. The name and location of that 'root' directory for the project can be anything the user wishes.

In order to tell the C++ compiler where to find the third party library header files and libraries, the path to the user created directory needs to be stored in a Windows environment variable. In Windows XP, this can be performed through the *"Control Panel"->"System"->"Advanced"->"Environment Variables"* dialog. A 'system variable' (as opposed to 'user variable') should be used if the development settings are to be used by several users of the PC (this requires System Administrator privileges). The Visual Studio may need to be restarted for the new system variable to be recognized.

The name of the new system variable should be set to ARPROTO, and the value should be the string corresponding to full path where the project directories are located.

For example, if the contents of the ARPrototype.zip file were placed in ARPrototype subdirectory of the root directory of the C: drive partition, the value of the newly created system variable should be "C:\ARPrototype\", and the "Environment Variables" dialog should appear as depicted on Fig. A2.1 below.

The ARPROTO variable is used in the project settings, specifically in the paths for 'Additional include directories' ('Project Settings', 'C++' tab, Category 'Preprocessor') and 'Additional library paths' ('Project Settings', 'Link' tab, Category 'Input').

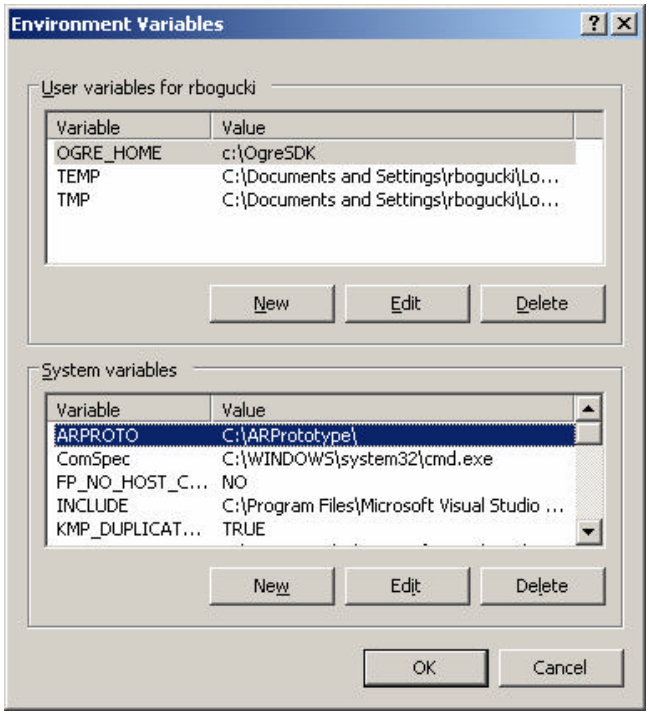


Figure A2.1. Appearance of the Environment Variables dialog after adding the new system variable ARPROTO.

The final step is to provide the compiler with the correct path where the compiled executable will be placed, and where it will be run from by the debugger. Unfortunately environment variables cannot be used in this dialog, therefore the path needs to be entered manually. In this way the built executable will be placed in the correct directory ensuring visibility of necessary resource files and dynamically loaded libraries. This is achieved by entering the required pathnames in the Debug pane of Project Settings Dialog, as shown on Figure A2.2. Settings for Win32 Debug and Win32 Release configurations (selected using the 'Settings For' pull-down list) need to be specified separately, for example if the root directory where the projects files were installed is, as previously, *C:\ARPrototype,* the directories for the Release configuration will be as follows:

- Executable for debug session:

  C:\ARPrototype\Binaries\Debug\ARProject.exe

- Working directory:

  C:\ARPrototype\Binaries\Debug\

While for the Release configuration the paths will be:

- Executable for debug session:

  C:\ARPrototype\Binaries\Release\ARProject.exe

- Working directory:

  C:\ARPrototype\Binaries\Release\

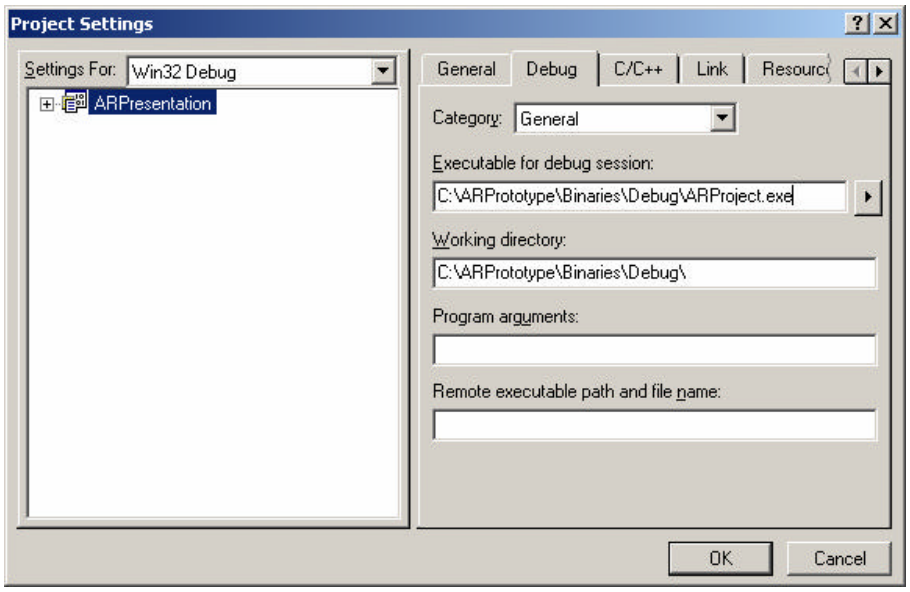


Figure A2.2 Modifying the debug session executable and working directory in the Debug pane of the Project Settings.

After applying the above settings, the Reader should be able to compile, link and execute the prototype software successfully.

APPENDIX B. DATABASE STRUCTURE AND SETUP

B.1 MySQL Database Contents

A single MySQL database was created and populated with sample data for a number of fiducials and responders for purposes of testing. Database server needs to be configured to allow connections from computer hosts participating in the testing. The network address of the database server, the database login username and password as well as an unique ID for each responder are provided to every mobile responder application at runtime in a configuration text file *database.cfg* present in the application's installation directory, otherwise default values are used.

The "ar" database contains two tables, named "responders" and "markers". The former contains tracking information individually updated by each of the mobile responder systems, the latter holds descriptive information about fiducial markers and the content associated with them. Tables B1 and B2 describe the individual fields of those two tables, the data types and the purpose of those fields.

The database's structure and data have been exported as a sequence of SQL statements and included with the Appendices CD. This allows to recreate the database configuration of the prototype easily when required, by executing the sequence of SQL commands on a MySQL server via client software.

field name	data type	description
id	int	ARTag Identification Number of the fiducial marker described by the 'marker' table's row. Primary Key of the table.
isuseable	binary	Is the fiducial marker ID useable. Allows to mark certain ARTag id numbers as not useable due to internals of ARTag planar marker system.
isused	binary	Is the fiducial marker with given ID deployed in given application (building). By default should be FALSE, set to TRUE when a fiducial is deployed during the fiducial placement stage by the building administrators.
posx	float	X geolocation coordinate of the fiducial in 3D worldspace.
posy	float	Y geolocation coordinate of the fiducial in 3D worldspace.
posz	float	Z geolocation coordinate of the fiducial in 3D worldspace.
label1	tinytext	Text description shown in first line of the fiducial's label augmentation.
label2	tinytext	Text description shown in second line of the fiducial's label augmentation. Shown only in 'expanded' mode when the label is within the active area of the display.
label3	tinytext	Text description shown in third line of the fiducial's label augmentation. Shown only in 'expanded' mode when the label is within the active area of the display.
isgeo	binary	TRUE: fiducial is tied to a definite physical location and has geolocation coordinates assigned to it. FALSE: fiducial is not tied to a location, can be carried by the responder (e.g. on the glove for quick access) or can be placed in multiple locations.
iscontent	binary	TRUE: there is additional content associated with a fiducial which can be accessed using the GUI. FALSE: no additional content is available (fiducial is used solely for the purposes of localization, or carries information in it's text label).
contenttype	tinyint	type of content. Currently implemented types:
resourcename	tinytext	Name of resource. Depending on type of content this can be a filename of a 3d object file, name of a 2D texture or virtual panorama.
iscontentgeo	binary	Specifies whether the content should be displayed in a specific manner with regard to the fiducial's geolocation coordinates. For example, a 3D model of a building should be shown properly oriented with regard to responders position and facing direction, while a schematic 3D model of machinery would be always shown from a predefined point of view.
fidazimuth	int	The facing direction (azimuth) of the fiducial marker in the physical environment. Needed in 'You Are Here'-type applications.
fidelevation	int	The fiducial's elevation. Allows for placement of fiducial on the floors, ceilings or slanted/curved surfaces.

Table B1. Description of the database fields of the 'markers' table holding the fiducial marker information.

Table B1, continued:

contentoriginx	float	Geolocation position of content object's origin. Provides worldspace coordinates which allow for several objects to be properly aligned relative to each other.
contentoriginy	float	
contentoriginz	float	
contentazimuth	int	Azimuth and scale of the content object. Simplifies editing and placement of content objects into the 3D environment. For example a building mesh may be created/edited in a local coordinate system, the facing direction and size relative to other elements in the simulation is described by the azimuth and scale fields.
contentscale	float	

field name	**data type**	**description**
id	int	Responder Identification Number. Primary Key of the table.
posx	int	"Last-seen" X geolocation coordinate of the responder.
posy	int	"Last-seen" X geolocation coordinate of the responder.
posz	int	"Last-seen" X geolocation coordinate of the responder..
locknown	binary	TRUE if the posx, posy, posz fields hold meaningful values. FALSE if the responder's position is unknown – for example when no fiducial has been viewed yet.
name	text	Name of the responder. Can be used in visualizations of responder positions.

Table B2. Description of the database fields of the 'responders' table holding the responder position information.

APPENDIX C. P5 GLOVE CALIBRATION

C.1 Glove Finger Sensor Calibration

Due to the offset and drift occurring in the electro-resistive elements used in the finger flexion sensors of the P5 glove, for best performance it is necessary to calibrate the sensors before using the glove. Optimally this procedure should be available through the AR application's interface, at present a separate application is used for that purpose. The recommended application, provided with the Dual Mode Driver package by Carl Kenner, is included on the CD in *Misc\P5GloveDriver\DualModeDriver* directory. The name of the executeable is VBControlPanel.exe. The original version of the calibration application provided by the manufacturer can be found in the OriginalDriver subdirectory.

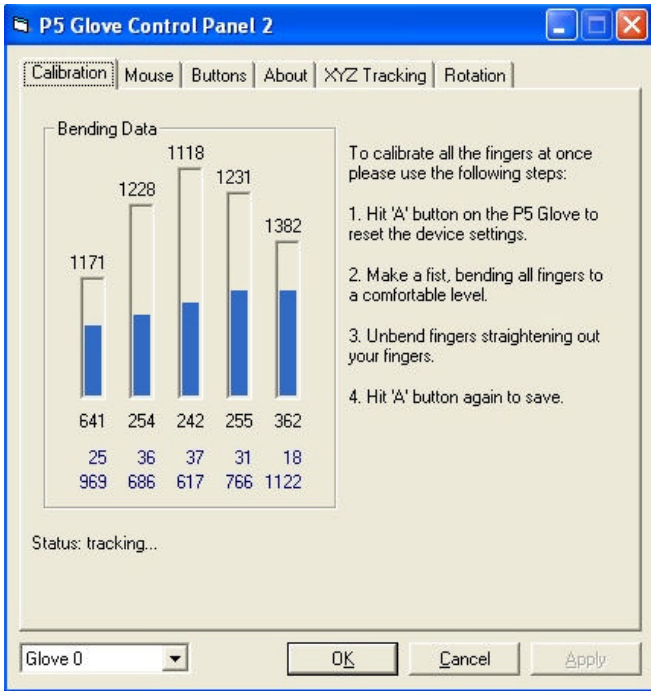


Figure C1. P5 Glove Calibration dialog

After starting the VBControlPanel application (see Figure C1), it is recommended to disable Mouse Driver functionality on the tab labelled “Mouse”. Calibration procedure is described in detail on the “Calibration” tab, and involves pressing the “A” button on the glove, making a fist with all fingers, unbending fingers and pressing the “A” button again. The calibration settings are stored internally in the glove.