An Authoring Tool for Building Mobile Phone AR Applications

A thesis

submitted in partial fulfilment

of the requirements for the Degree

of

Master of Science

in the

University of Canterbury

by

Yuan Wang

Examining Committee

Associate Professor Tim Bell	Supervisor
Professor Mark Billinghurst	Co-Supervisor

University of Canterbury

2010

Table of Contents

Acknowledgments 6
Abstract 7
Chapter 1 Introduction...1
1.1 An Introduction to Augmented Reality (AR) 2
1.2 Authoring Tool 9
1.3 Thesis Structure 10
1.4 Research Questions 11
1.5 Research Contributions 12
Chapter 2 Background Research 13
2.1 AR Authoring Tools...13
2.1.1 AR Authoring Tools for Programmers.....................................14
2.1.2 AR Authoring Tools for Non-programmers.............................18
2.2 AR Authoring Tools for Mobile Phones..25
2.3 Advertising Using Mobile AR 29
Chapter 3 AR Authoring System 34
3.1 Overview of the System 34
3.2 AR Authoring Tool...36
3.2.1 ComposAR 36
3.2.2 ComposAR Customization 38
Chapter 4 AR Viewing Tools..43
4.1 Edgelib 43
4.1.1 Multi-platform Deve1opment..44
4.1.2 High-performance Graphics...45
4.1.3 Edgelib Integration with ARToolKit on a Desktop PC 47
4.1.4 Input XML File 53
4.2 Studierstube ES...59
4.2.1 Configuring StbES 62
4.2.2 Video Configuration...64
4.2.3 The Tracking Subsystem...66
4.2.4 Scene Graph File: scene.xml...67
Chapter 5 AR Pattern Generator..69
5.1 User Interface...69
5.2 Pattern Files 70
5.2.1 ARToolkit Marker Pattern File..71
5.2.2 StbES Marker Pattern File 71
5.3 Marker Preview and Printing 71

Chapter 6 Evaluation and Result 73

6.1 Experimental Task and Design 73
6.2 Experimental Measures 75
6.3 Results 77
6.4 Interviews...97
6.5 Discussion 98

Chapter 7 Performance Measurements 100

Chapter 8 Conclusion and Future Work 105

References 107

Appendix A: Python Script Examples..112

Appendix B: Experiment Questionnaire..113

List of Figures

Figure 1.1: Milgram's Mixed Reality Continuum 3
Figure 1.2: Sutherland's System 4
Figure 1.3: Examples of Augmented Reality in Broadcasting 5
Figure 1.4: First AR Application on PDA and Mobile 6
Figure 1.5: Screenshot of the Eye of Judgment...8
Figure 2.1: ARToolKit 15
Figure 2.2: ARTag Markers Detected in An Image 16
Figure 2.3: Examples for Using OsgART Library 17
Figure 2.4: An Example Work Session in DART 19
Figure 2.5: AMIRE Authoring Interface 20
Figure 2.6: Oil Refinery Application...21
Figure 2.7: The MARS Authoring Tool 22
Figure 2.8: WEM System 23
Figure 2.9: The ULTRA Interface 23
Figure 2.10: 3D Authoring Tool 24
Figure 2.11: Process Authoring Tool 24
Figure 2.12: Nike 3D Mobile Soccer Shoe 30
Figure 2.13: Fanta Virtual Tennis Display 31
Figure 2.14: Ford 'ka' 3D Mobile Car Display..32
Figure 2.15: Augmented Reality at Wellington Zoo 33
Figure 2.16: Mobile AR Market 33
Figure 3.1: The Structure of the AR Authoring System 34
Figure 3.2: ComposAR Interface Components 37
Figure 3.3: The ComposAR-Mobile Interface 39
Figure 3.4: The Python Code for Keypad Functionality...................................40
Figure 3.5: Different 3D Model on the Marker When the Button is Clicked ..41
Figure 3.6: Different Interface of ComposAR 42
Figure 4.1: Screen Snapshots of 3D Objects from Edgelib 45
Figure 4.2: Turtle 3D Model in MilkShape 3D. Model by psionic3d.co.uk 46
Figure 4.3: Four Frames of an Animated Turtle Model in MilkShape 3D 46
Figure 4.4: Rendering Methods in Edgelib 47
Figure 4.5: Sample Code for Edgelib Surface Setting 49
Figure 4.6: Camera Setting Pop-up Window 49
Figure 4.7: Camera View Setting for the Edgelib...50
Figure 4.8: An Example Code to Set Up the Virtual Camera...........................52
Figure 4.9: Prototype AR Viewer Using Edgelib 53
Figure 4.10: An Example of XML-RPC Structure 55
Figure 4.11: An Example Code for the XML-RPC Structure 56
Figure 4.12: A Truck Model in the AR Viewer 57
Figure 4.13: XML File for the Truck Model from ComposAR Tool...............58
Figure 4.14: Structure of Studierstube ES 60
Figure 4.15: Different Markers Types Supported by StbTracker 61

Figure 4.16: An Example of Windows Video Configuration File....................64
Figure 4.17: An Example of WindowsCE Device Video Configuration File..65
Figure 4.18: An Example for Configuring the Tracking Subsystem................66
Figure 4.19: An Example of Scene Graph File 67
Figure 4.20: Screenshot of the Example..68
Figure 5.1: AR Pattern Generator Interface 70
Figure 5.2: An Example of a Marker...72
Figure 6.1: A Target Authoring Scene 74
Figure 6.2: Subjective Survey Scores for Questions 1 - 9. 88
Figure 6.3: Subjective Survey Scores for Questions 10 - 13 89
Figure 6.4: Subjective Survey Scores for Questions 14 and 15........................91
Figure 6.5: Subjective Survey Scores for Questions 16 and 17........................92
Figure 6.6: Subjective Survey Scores for Questions 18 and 19........................93
Figure 7.1: Test Models Rendered for Benchmarking...................................103

List of Tables

Table 2.1: Types of Authoring Tools 14
Table 2.2: Authoring Tools for Mobile Phones 28
Table 4.1: Platforms Supported by Edgelib 44
Table 4.2: The Important Node for the Windows XP and Mobile Platform....63
Table 6.1: Average Time to Perform Task One...77
Table 6.2: Error Taken to Perform Task One..78
Table 6.3: Average Time to Perform Task Two 79
Table 6.4: Error Taken to Perform Task Two..80
Table 6.5: Average Time to Perform Task Three 81
Table 6.6: Error Taken to Perform Task Three...82
Table 6.7: Average Result for Ease of the Task..89
Table 6.8: Average Result for Felt in Control..91
Table 6.9: Average Result for People Liked Using the Tool............................92
Table 6.10: Results of the User Experience 96
Table 7.1: Tracking Speed on Different Size of the Markers.........................101
Table 7.2: Benchmarks Performed with Single and Multi Marker 101
Table 7.3: Result for the Test Model 103

Acknowledgments

I'd like to thank everybody without whom this dissertation would not have been possible:

First and foremost, I would like to thank my supervisor and co-supervisor Dr Tim Bell and Prof. Mark Billinghurst for having given me the opportunity to study in the area of Augmented Reality. During my time in the HIT Lab NZ which has a wonderful research environment, I learned a lot about the research and gained some experiences. I want to thank them especially for all invariable support and guidance to me.

Special acknowledgements go to Julian Looser and Harmut Seichter, who provided me with invaluable advice and assistance. I'd also like to thank all the other staff and students of this lab for their help.

I am grateful to my parents who always support and encourage me in my studies.

Abstract

This thesis describes the research on developing an authoring tool for mobile phone Augmented Reality (AR) applications. This work is based on earlier work at the HIT Lab NZ on ComposAR, a tool for authoring PC based AR applications. We describe modifications to ComposAR that allows end-users to prototype mobile AR applications on a PC, and player software that allows prototype AR applications to be delivered on a mobile phone. In this way, end-users with little programming experience can develop simple mobile AR applications. To prove the applicability of this authoring tool, a user evaluation was conducted with some users and performance compared between programmers and non-programmers and across different authoring tools.

1 Chapter 1 Introduction

Augmented Reality (AR) (Azuma et al., 2001) is a field of computer science research which studies systems that allow virtual images to be mixed with the real world. In recent years, the first AR applications have been deployed on PDAs (Pasman and Woodward, 2003) and mobile phones (Henrysson et al., 2005) . However, developing these applications requires a lot of low level coding and specialized skills. Unlike PC-based AR systems, there are no high-level authoring tools that allow developers to rapidly build mobile AR applications, especially for non-programmers. The focus of this thesis will be to develop an authoring tool for building mobile AR applications.

The HIT Lab NZ has developed a tool for the PC for authoring AR applications called ComposAR. ComposAR provides a Python-based scripting tool for specifying the virtual objects in an AR scene and the interactions between the objects. It creates an XML file as the output that specifies the AR scene content and interaction in the application. The main goal of this thesis is to modify the ComposAR tool to allow people to prototype mobile AR applications on a PC, and develop AR player software that allows AR experiences to be delivered on a mobile phone.

In this chapter, we will provide an introduction to the research topic, explain the fundamentals of Augmented Reality (AR) and authoring, and discuss why a mobile AR authoring tool is important.

1.1 An Introduction to Augmented Reality (AR)

What is Augmented Reality?

Ronald Azuma's definition of AR is one of the more focused descriptions**.** **Augmented reality (AR)** (Azuma et al., 2001) is an environment that includes both virtual reality and real-world elements, and has three key characteristics (Azuma, 1997):

- It combines real and virtual images.
- It is interactive in real time.
- The virtual imagery is registered in 3D.

Augmented Reality is one part of the broader interface taxonomy called ‘mixed reality’ (Milgram et al., 1994) (see Figure 1.1) that includes any display in which images of real and virtual objects are combined.

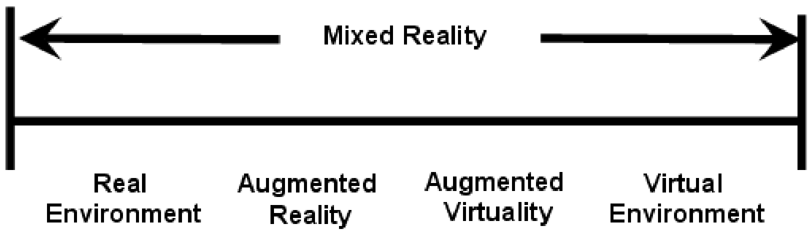


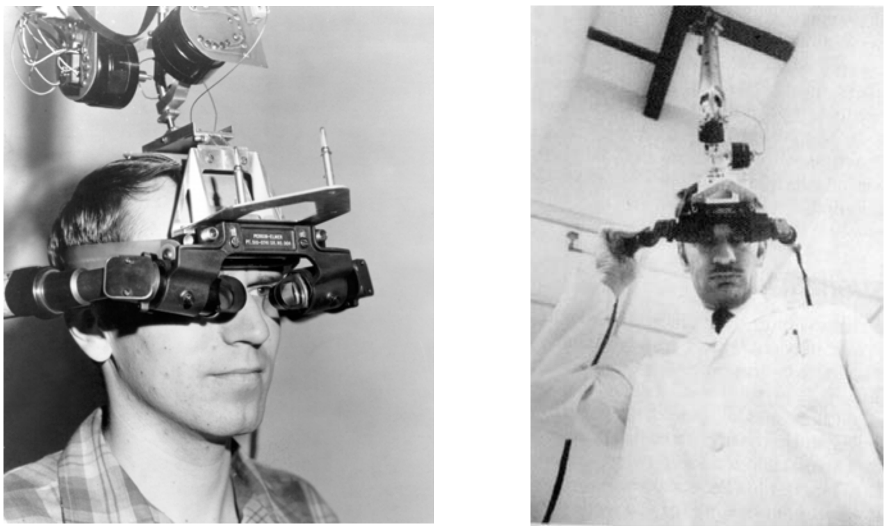
Figure 1.1: Milgram's Mixed Reality Continuum

Watching a TV or playing a football which does not need mediation by technology is called *a real experience*, while the opposite is called *a virtual experience* where reality is replaced by an immersive computer-generated world. Experiences lying between these two extremes are known as Mixed Reality, including Augmented Reality and Augmented Virtuality.

Augmented Reality presents the predominantly real environment augmented by the virtual objects, while Augmented Virtuality presents the predominantly real world objects merge into virtual environment.

A brief history of AR

The origins of AR can be traced to Ivan Sutherland's "Ultimate Display" (Sutherland, 1965) idea in 1965. Three years later, Sutherland implemented a real-time 3D HMD system. It is the first computer system that merged real and virtual images. It used a virtual reality headset with one CRT element for each eye, connected to a tracking rig (see Figure 1.2) (Sutherland, 1968).

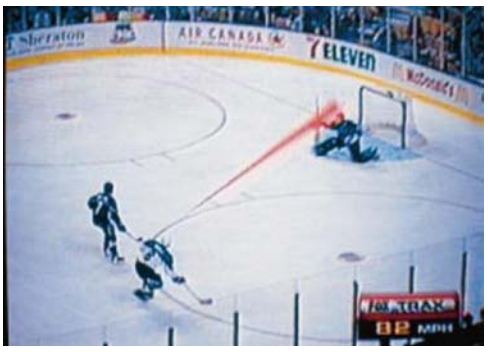


(a) Optical see-through HMD **(b) Head tracking**

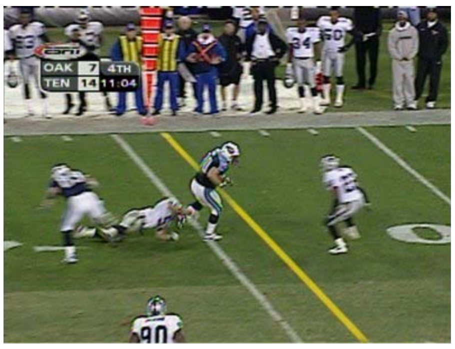
Figure 1.2: Sutherland's System (Image from Ivan Sutherland)

Around the same time, Furness developed the "Super Cockpit" as a flight simulator, which could generate the visual scene projected directly to the pilot headgear. The pilot could interact with this virtual scene and give verbal commands. This was used by the US Air Force (Furness, 1986).

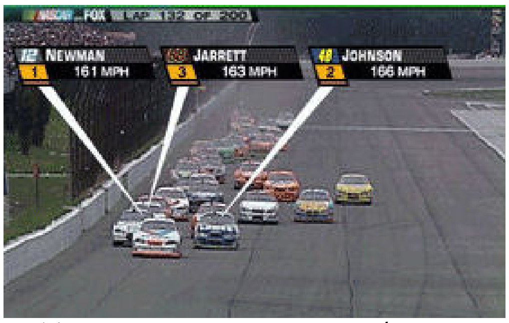
This technology began to be knows as “augmented reality” after Tom Caudell, a researcher at Boeing, developed a head-mounted display system which was used to help workers to install wire harnesses in aircraft in the early 1990s (Caudell and Mizell, 1992).



(a) Ice hockey puck with virtual comet trail (Image from Rick Cavallaro)



(b) The virtual first down line in American football (Image from Shel Brannan)

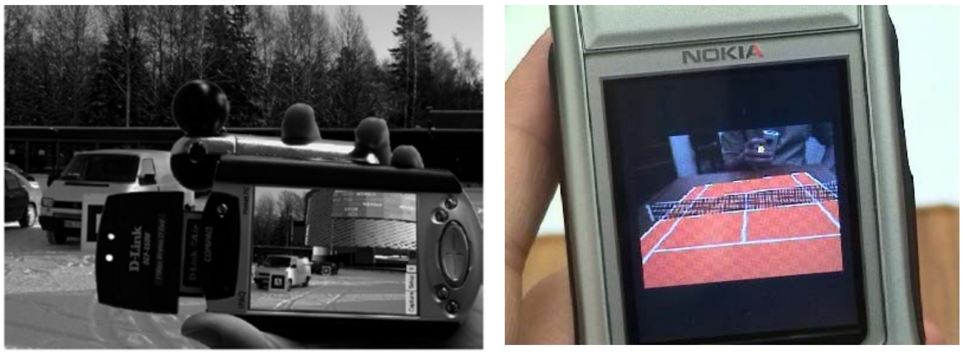


(c) The virtual car info in RACEf/x (Image from Sportvision)

Figure 1.3: Examples of Augmented Reality in Broadcasting

Augmented reality became widely used in sport broadcasting from 1997. One example is “FoxTrax” ice hockey puck, shown in Figure 1.3(a). The glowing puck with a virtual comet tail on the ice rink was tracked by the television cameras and indicated the path can be seen on TV (Cavallaro, 1997). Since 1998, the first and ten system has been used on football broadcasts. An example is the "first down" line in American Football broadcasts (Brannan, 2001), shown in Figure 1.3(b). A virtual yellow line is projected on the field to show how far the team has to go for a first down. Another example is RACEf/x shown in Figure 1.3(c), which is a motor sport. This system uses GPS track and display statistical information of the car's performance on the screen in real-time (Sportvision, 2006).

In 1998, the ARToolKit (Kato and Billinghurst, 1999) computer vision tracking library was released. It solved two of the main problems in Augmented Reality: one is the viewpoint tracking and another is virtual object interaction. ARToolKit has been widely used to build AR applications since 2004.



(a) Fully equipped iPAQ with the test model (Image from Pasman and Woodward) **(b) AR application on the mobile phone (Image from Henrysson et al)**

Figure 1.4: First AR Application on PDA and Mobile

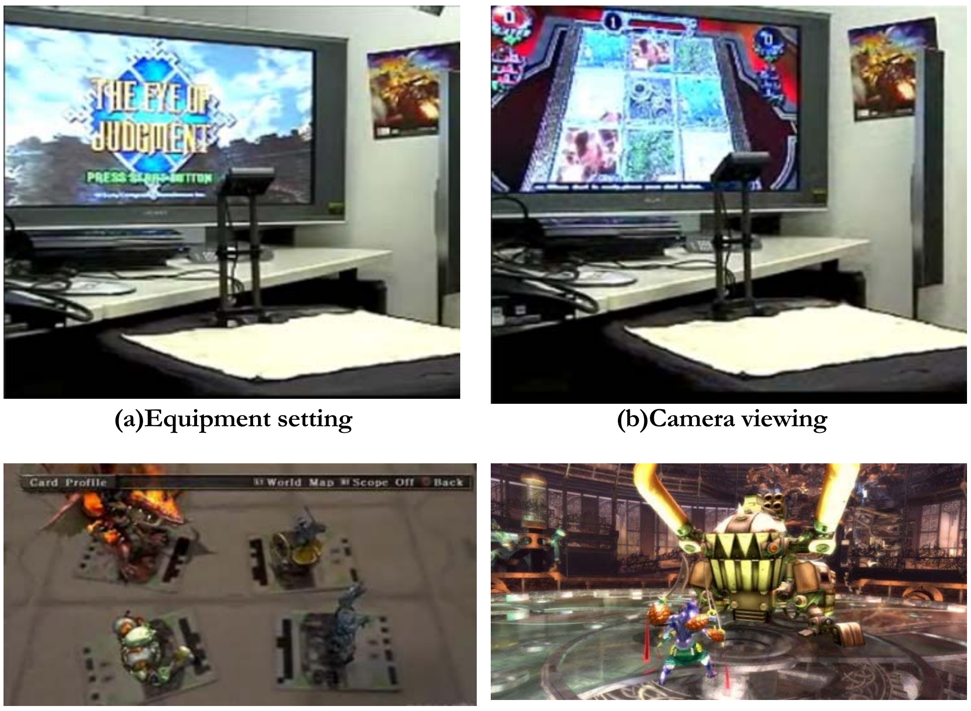
In the 2000s, the first AR application was deployed on PDAs (Pasman and Woodward, 2003) and mobile phone (Henrysson et al., 2005). Figure 1.4

(a) shows an outdoors AR demonstration on a PDA device which presents a client/server implementation. The camera in the client captures the image, and sends it to the server for processing. Once the virtual objects have been rendered on the server, the image is sent back to the client and overlaid on top of the original one to produce an AR view.

Figure 1.4 (b) shows an AR tennis application. It uses augmented reality running on Nokia mobile phones with a set of small fiducial markers for tracking. The phones can determine their own locations by tracking the markers, and can be used as ‘tennis rackets’ with a virtual ball. The direction and position vectors of the ball are sent over to the other phone using Bluetooth.

In 2007, Sony released a turn-based card battle video named “The Eye of Judgment” (see Figure 1.5), which was the first end user applications featuring Augmented Reality for Sony PlayStation 3 game console.

Just as shown in Figure 1.5(a) in this game, the following are needed: a playing mat which is a 3 × 3 grid of rectangles to place your cards upon, a camera which is called EyeToy and used to capture the image of the cards on the grid, a special stand which holds the camera to place the right angle and gets view of the playing mat, and 30 special cards used as game pieces each of which has AR fiducial marks on them to assist with tracking and AR overlay(shown in Figure 1.5 (a)).



(a)Equipment setting **(b)Camera viewing**

(C)Card use **(d)Battle mode in action**

Figure 1.5: Screenshot of the Eye of Judgment

When the cards are put down on a table, PlayStation 3 is able to recognize the card, via the EyeToy camera (see Figure 1.5 (b)). Each card is associated with a different monster. As shown in Figure 1.5(c), the virtual monsters erupt out of the cards on the television screen. As long as the cards can be moved around on the table, the monsters move around on the screen, allowing the player to interact with them just by moving them. Move them toward another card, and the monsters onscreen will interact and battle (see Figure 1.5(d))

The numbers of AR applications have since grown rapidly and widely with the first dedicated conferences (The international Symposium on Mixed and Augmented Reality - ISMAR[1]). For example, FLARToolkit (Koyama, 2008) — Flash-based Augmented Reality Toolkit, brings AR environment to the web browses.

1.2 Authoring Tool

An authoring tool[2] is a software package which developers use to create and package content deliverables to end users. Typically, authoring tools enable users to create a final application merely by linking together objects, allowing those who use the tool to produce attractive and useful application. Authoring tools require less technical knowledge to master and are often used for applications that present a mixture of textual, graphical, and audio data.

Since the term is rather general, authoring tools have been used widely. Some programs such as web editors, Flash, and PowerPoint are also considered as authoring tools. The most commonly used is to create e-learning modules. However, there is no group of programs specifically to support for mobile AR content.

[1] http://www.ismar-society.org/, online as of September 2009.
[2] Authoring tool, http://en.wikipedia.org/wiki/Authoring_systems, online as of September 2009.

1.3 Thesis Structure

This section provides a road-map of the chapters in this thesis.

Chapter 2 Background Research presents related work in the aresa of AR authoring tools, software libraire for mobile AR applications, and some advertising systems using mobile AR technology.

Chapter 3 AR Authoring System describes the overview of the whole system, and then introduces AR authoring tool in detail.

Chapter 4 AR Viewing Tools presents the AR viewing applications that we have developed for desktop and mobile AR applications.

Chapter 5 AR Pattern Generator describes why we need it and how it works for the AR application.

Chapter 6 Evaluation and Result describes the experiments to test the development of the authoring tool and performance compared between programmers and non-programmers and across different authoring tools.

Chapter 7 Performance Measurements presents the performance of mobile AR applications made by the authoring tools. The measurements compare tracking performance with different numbers of visible markers and different models.

Chapter 8 Conclusion and Future Work provides a concise summary of this contributions of this thesis and proposes directions for the furture work.

1.4 Research Questions

The main research questions of this thesis are:

- Are there any AR authoring tools for a mobile phone available today?
- How easy does the system make it for the non-programmers to develop applications?
- How fast does it perform on a mobile phone?
- What is the difference in quality of the image compared with that on a PC?
- What are the other issues to consider?

1.5 Research Contributions

The main contributions of this thesis are:

- Integrating one existing game engine library for viewing AR content
- Extending existing HIT Lab NZ PC based AR authoring tools to support mobile phones
- A formal evaluation of AR authoring tool for different end users

2 Chapter 2 Background Research

The research in this Masters thesis is mainly focused on developing and evaluating an authoring tool for mobile AR applications, particularly for use by developers with little programming experience. One target area for an application tools like this is for prototyping mobile AR advertising campaigns.

Although there is no existing work on mobile AR authoring tools for non-programmers, there are several previous AR authoring tools for PC applications that the research can be built on.

In this chapter, we first review AR authoring tools in general, and then software libraries for mobile AR applications. Finally, we present some advertising systems using mobile AR technology.

2.1 AR Authoring Tools

There are several existing authoring tools for building desktop AR applications. These can be organized into two types:

1) AR authoring tools for programmers

2) AR authoring tools for non programmers

Authoring tools for programmers are typically code libraries that require programming knowledge, while tools for non-programmers are those that require no programming knowledge, such as visual tools that include drag and drop interfaces for building applications without writing any lines of code. These categories can be further organized into low level tools which require coding/scripting skills, and higher level application builder tools which use high level libraries or visual authoring techniques. Example authoring tools are shown in Table 2.1 and described later in this section.

Table 2.1: Types of Authoring Tools

	Programmers	**Non-programmers**
Low level	ARToolKit arTag	DART AR-Blender ComposAR
High level	Studierstube osgART	AMIRE MARS ULTRA

2.1.1 AR Authoring Tools for Programmers

A number of programming libraries enable developers to author AR applications. For example, ARToolKit (Kato and Billinghurst, 1999) is a free and open-source C software library which can be used to develop AR interfaces by providing computer vision based tracking of black square markers.

Figure 2.1 shows ARToolKit being used to show a three-dimensional virtual character appearing standing on a real card. The user can see the AR scene by wearing the head set display. When the card is moved by the user, the virtual character moves with it and appears attached to the real card (ARToolKit, 2001).

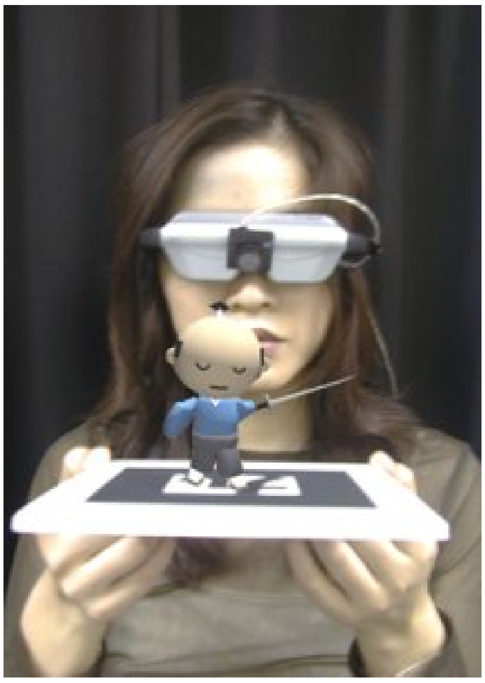


Figure 2.1: ARToolKit (Image from ARToolkit)

However, to develop an AR application with ARToolKit requires significant C programming skills. The additional code has to be developed for 3D model loading, interaction techniques, and other utility functions. This need for integration with additional libraries is typical of low level programming tools.

Another low level AR library is ARTag (Fiala, 2005), which is a computer vision based marker tracking system that uses digital coding theory to get a very low false positive and inter-marker confusion rate. It is a bi-tonal system which contains 2002 planar markers. Each marker consists of a square border and an interior region which are black or white cells filled with a 6 × 6 grid. Figure 2.2 shows some example ARTag markers. Like ARToolKit, to develop a complete AR application with ARTag requires considerable additional C/C++ programming experience.

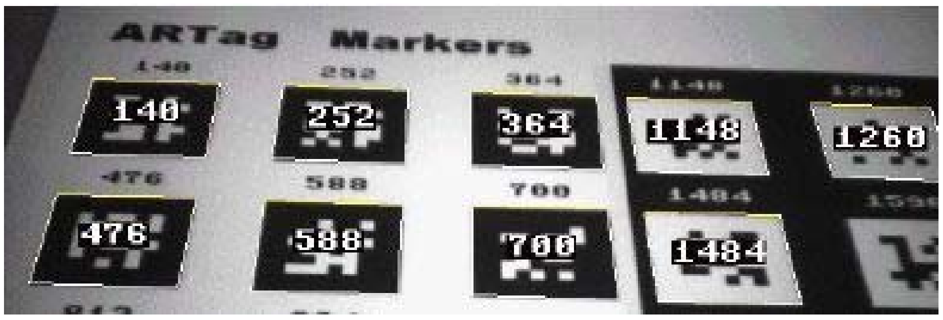


Figure 2.2: ARTag Markers Detected in An Image (Image from Mark Fiala)

The osgART library (Grasset et al., 2005) is a high level C++ Open Scene Graph library based on top of the ARToolKit tracking library. Unlike ARToolKit, osgART includes a code for loading 2D or 3D models and animation. A collection of classes is provided in it so that it is easy to make AR applications. Some of the main functionalities that the library supports are: high level integration of video input (which is how the video object deals with the mapping of the video texture on itself), spatial registration (which is the transformation mapping from the ARToolKit tracking to the OSG framework), and photometric registration (which is disparity between the real content and the virtual content).

There are some examples of AR applications (OSGART, 2006) built with osgART shown in Figure 2.3.

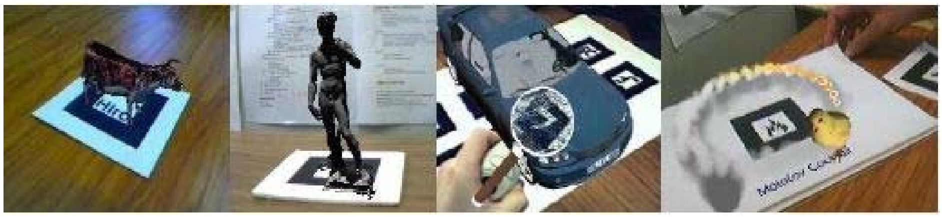


Figure 2.3: Examples for Using OsgART Library (Image from osgART)

Another high level library is the Studierstube library (Schmalstieg et al., 2002) which provides a complete distributed system for developing applications in virtual and augmented reality. The distributed nature of Studierstube makes it particularly good for developing collaborative augmented reality applications. Studierstube is a cross platform and is also a leading framework for the development of mobile, collaborative and ubiquitous AR applications.

A common feature of these libraries is that although they are of high level, they typically require C or C++ programming ability, users also require other content development tools to produce the AR content and it takes a relatively long time using them to produce an AR application.

2.1.2 AR Authoring Tools for Non-programmers

There is another set of AR authoring tools that have been developed for non-programmers such as artists or designers.

One of the first is DART (MacIntyre et al., 2005), the Designer's AR Toolkit (see Figure 2.4), which is a plug-in for the popular Macromedia Director software. The main aim of DART is to allow multimedia application designers to develop AR applicatioins. DART is designed to allow non-programmers to create AR experiences by using the low level AR services provided by the Director Xtras, and to integrate these with existing Director behaviours and concepts. DART supports both visual programming and a scripting interface. Unlike ARToolKit and osgART, DART is specifically developed for multimedia designers and non-programmers.

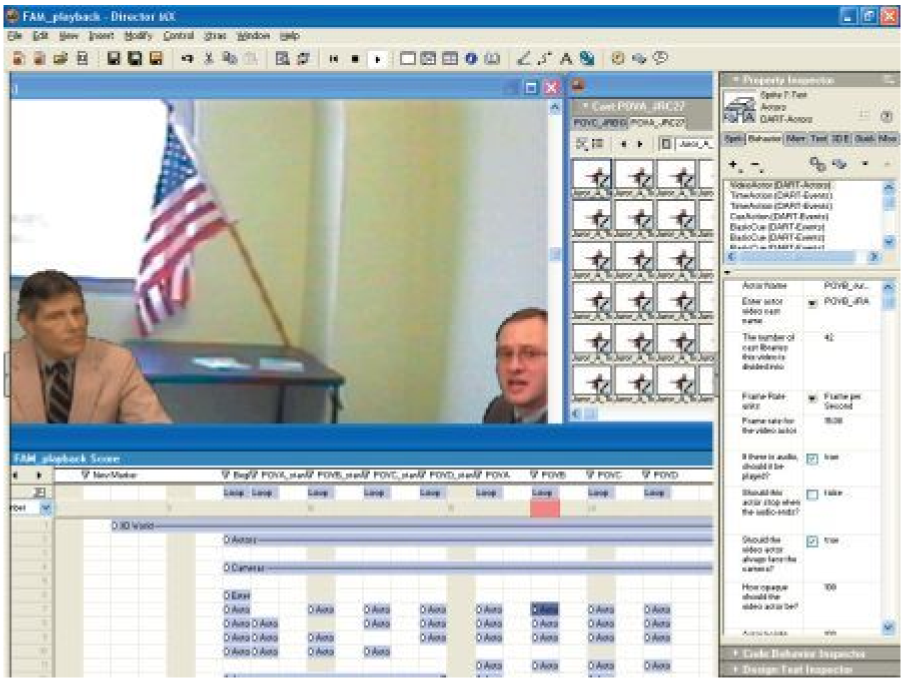


Figure 2.4: An Example Work Session in DART (while debugging the Four Angry Men (FAM) experience). The entire score for FAM is visible, including the nine scenes and most of the actors. The stage (containing the running experience) is visible, as is part of the content for one video actor, and some of Director's editing windows. (Image from MacIntyre et al.)

AMIRE (Grimm et al., 2002) is an authoring tool for the efficient creation and modification of augmented reality applications. The interface is shown in Figure 2.5 (AMIRE, 2002).

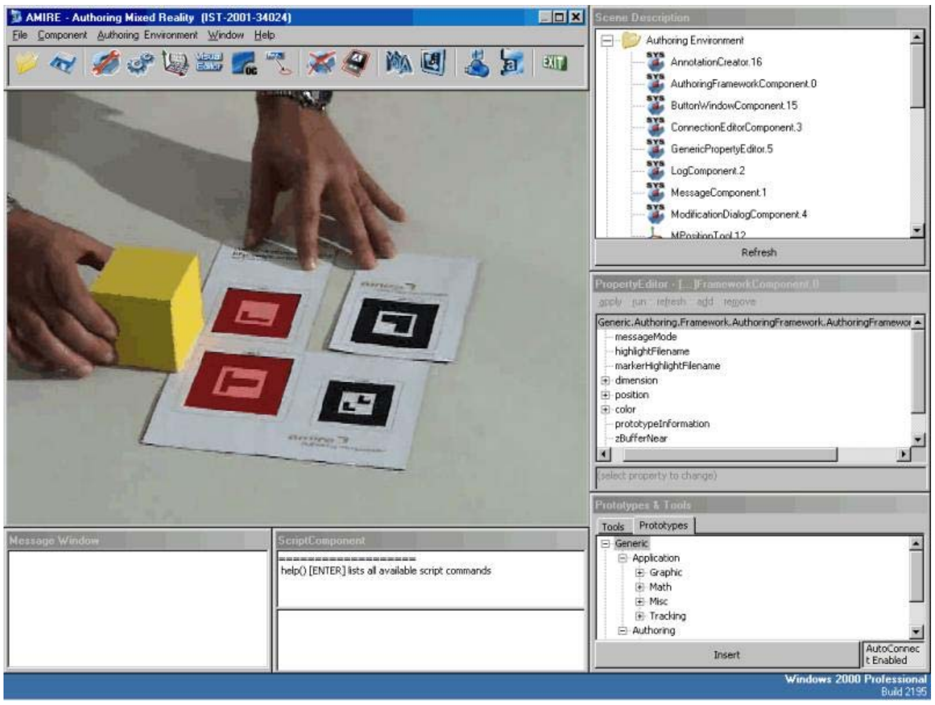
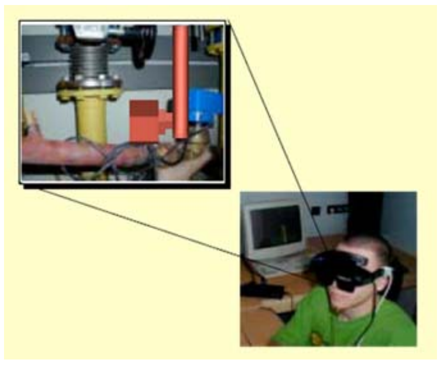
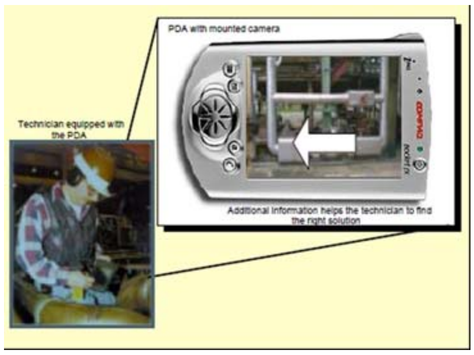


Figure 2.5: AMIRE Authoring Interface. (Image from Grimm et al.)

The AMIRE framework provides an interface to load and replace a library at runtime and uses visual programming techniques to interactively develop AR applications. AMIRE is designed to allow content experts to easily build applications without detailed knowledge about the underlying base technologies. Two completely different AR applications have been developed based on using AMIRE, an oil refinery (see Figure 2.6) and a museum, showing the flexibility and efficiency of the AMIRE approach.



(a) Indoor solution (Tracking system, iGlasses) (image from AMIRE website)


PDA with mounted camera
Technician equipped with the PDA
Additional information helps the technician to find the right solution

(b) Outdoor solution (Handheld/Tablet PC) (Image from R. DORNER et al.)

Figure 2.6: Oil Refinery Application

Some of these PC based authoring tools were also designed for building mobile AR applications. For example, the MARS (Mobile Augmented Reality Systems) authoring tool (Guven and Feiner, 2003) uses a 3D graphical user interface to allow users to create mobile outdoor AR applications. It is designed for non-programmers, and allows them to preview their results on a desktop workstation, as well as with an augmented or virtual reality system.

Using the MARS authoring tool, several situated documentaries were authored which told the stories of events that occurred on Columbia University campus (see Figure 2.7).

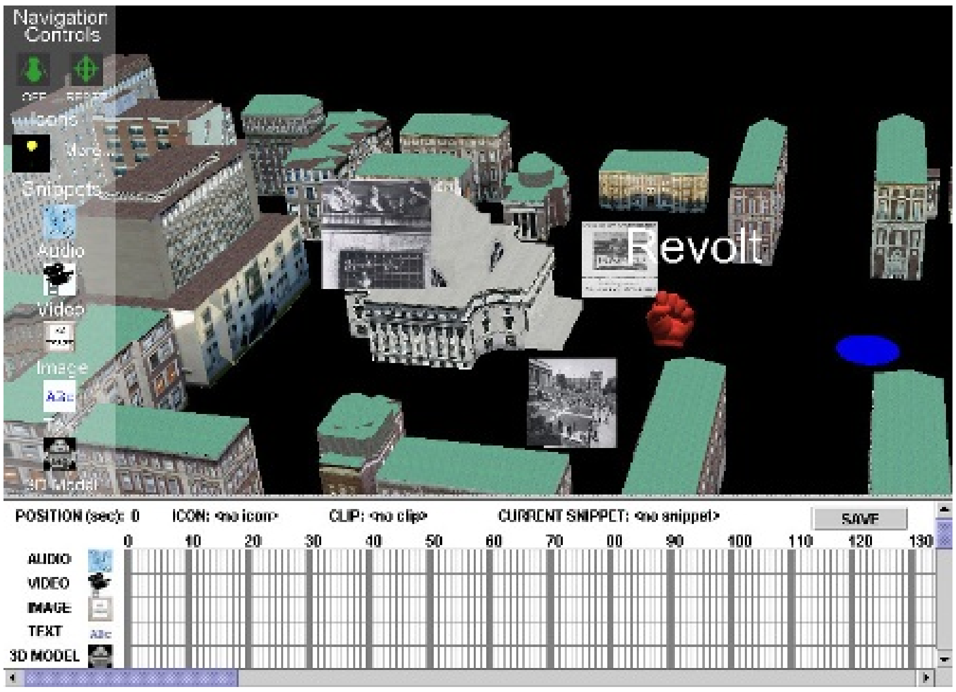


Figure 2.7: The MARS Authoring Tool (Image from Guven and Feiner)

Fisher has developed an authoring tool for creating outdoor AR experiences. In this case, authoring is done on a desktop computer with a web based 2D map or in the field with a mobile phone (Fisher, 2001) (Fisher, 2002). It can be used to author a variety of virtual tours through a specific location, depending on the viewpoint and expertise of the user. Figure 2.9 illustrates the Wearable Environmental Media (WEM) Project prototype mobile system (Fisher, 2001). The user wears a wireless backpack as shown in Figure 2.8, which is containing a number of different technologies for capturing the video imagery, transmitting the video and data, and determining the user location. The large disk is used to locate the user’s location. It is a GPS antenna, which is locating about 2-centimeter accuracy.

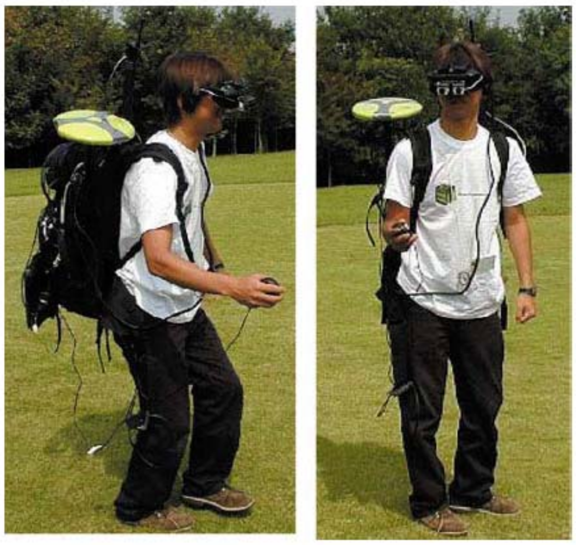


Figure 2.8: WEM System (Image from Fisher)

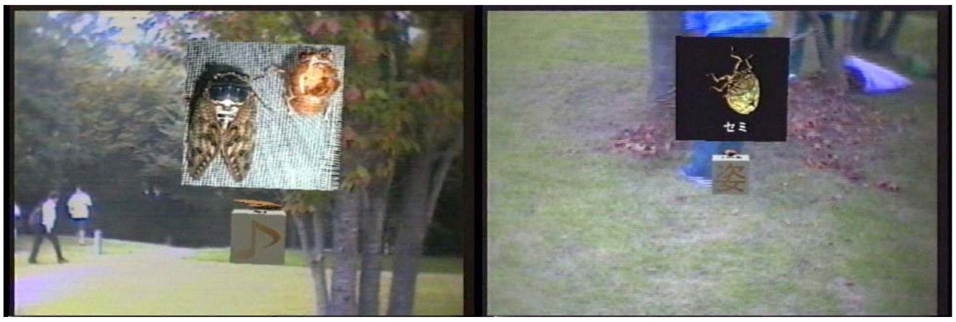


Figure 2.9: The ULTRA Interface (Image from Fisher)

Another example is the European project ULTRA (Alexandra Makri et al., 2005) – (“Ultra portable augmented reality for industrial maintenance applications”) which has the goal to implement a new mobile AR-system that works on minimal hardware. ULTRA features a set of content generation/authoring tools. The authoring tool has two parts: a 3D authoring tool (see Figure 2.10), and a visual process authoring tool (see Figure 2.11). The 3D authoring tool creates 3D animations, based on the concept of templates. The process authoring tool uses the visual programming to create an interactive application. This tool is designed for PDAs and handheld PCs and not for mobile phones.

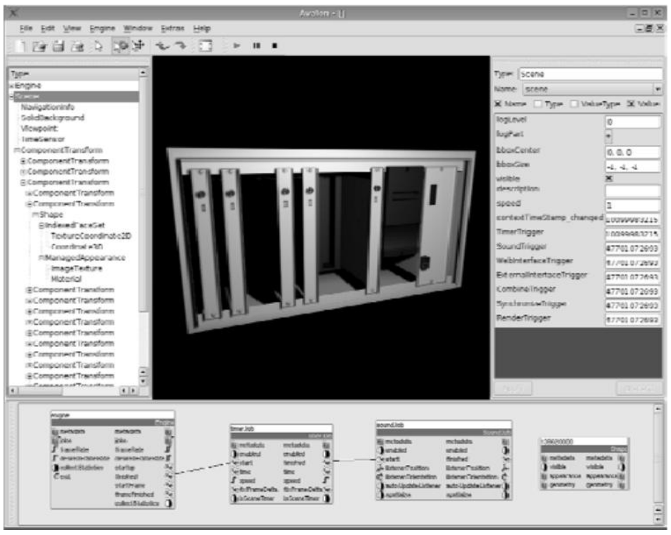


Figure 2.10: 3D Authoring Tool (Image from Alexandra Makri et al.)

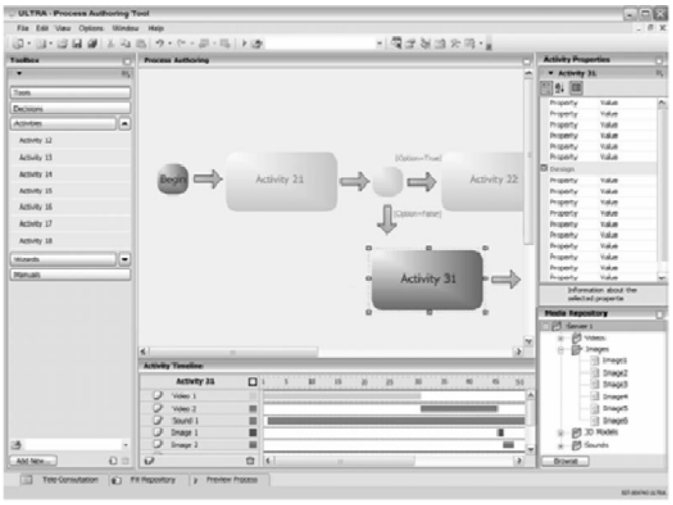


Figure 2.11: Process Authoring Tool (Image from Alexandra Makri et al.)

Some authoring tools are designed to extend other content development tools. For example, AR-Blender (Grimm, 2006) is an extended version of the Blender 3D modeling program that integrates ARToolKit into the Blender application. However, it is very complicated and time consuming to combine a virtual world with a real one, because similar problems remain in building 3D geometries and MR applications. Developers can use the Blender scripting interface to develop simple PC-based AR applications that include their 3D models.

A common feature of these tools is that they use visual programming techniques or simple scripting to support quick prototyping, they are interpretive rather than compiled allowing for fast redesign of ideas, and they are integrated into other design tools. However none of these tools can be used for authoring mobile phone AR applications.

2.2 AR Authoring Tools for Mobile Phones

Although there are several tools for building desktop AR applications, there is less support for the mobile phone AR. At the low level, the ARToolKit tracking library has been ported over to the Symbian operating system (Henrysson et al., 2005) but this requires the use of other code such as the OpenGL ES graphics library in order to complete a mobile AR application. The Studierstube Tracker library (Schmalstieg et al., 2002) is another low level AR tracking library that is available for multiple platforms such as Symbian, iPhone and Windows Mobile.

One of the only higher level programming libraries for mobile AR applications is the Studierstube ES (Dieter and Daniel, 2008) (StbES) library. This is a C++ based application framework for developing AR applications for mobile devices. It is a cross-platform, running on Windows, Windows Mobile, or the Symbian operating systems. Studierstube ES provides support for 2D and 3D graphics, video capture, tracking, multimedia output, persistent storage, multi-user synchronisation, and application authoring. It requires a high level of programming skill to use and so is not suitable for non-programmers.

Apart from Studierstube ES there are other tools for developing non-AR 2D and 3D graphics applications for mobile phones. One of the most powerful and low-level game engines for mobile devices is the Edgelib library (Edgelib, 2007), which is designed for developing quality applications and high-performance games. Its key features are: multi-platform development, high-performance graphics, Network connectivity and support for RGBA surfaces.

The M3GE (Mobile 3D Game Engine) library[3] is a Java game engine based on the Mobile 3D Graphics API for JME spec (M3G - JSR 184). It

[3] M3GE, http://m3ge.dev.java.net, online as of September 2008.

has a development library that allows graphical rendering to be handled by the application; image loading, input, output, and general functions like AI, collision detection and other rendering facilities are also managed. M3GE aims to perform all global functions in the application in a single core block, separating graphical routines and application logic. The engine was tested in a Siemens CX 65 phone and it operated at 8 to 16 frames per second.

For non-programmers, Python[4] is available for rapid development of mobile applications. The Symbian version of Python allows a user to develop python scripts on their desktop and then run them on their phone using a native interpreter. It has support for 2D and 3D graphics, camera input, file handling and networking, and many other functions for rapidly prototyping mobile applications. However it does not support a visual development tool and so requires the developer to learn scripting.

Other high level visual design tools are available to author mobile graphics applications. Among them, the most popular is FlashLite[5], a version of Adobe Flash that has been specifically designed for use on mobile phones. With FlashLite, a developer can use a combination of visual authoring and ActionScript scripting to easily build interactive phone applications such as games, information tools, screensavers, and e-learning applications.

[4] Python, http://www.python.org/, online as of September 2008.
[5] Adobe Systems Incorporated, http://www.adobe.com/products/flashlite/, online as of September 2008.

However, there is no support for 3D graphics or camera input. It has long been used on a pretty wide variety of devices like Sony Ericsson p900, Nokia 3650 and Nokia NGage to name a few.

Table 2.2 shows the tools available for developing mobile AR applications. Authoring tools are currently available for mobile AR applications (such as Studierstube ES) requiring C++ programming experience. There are some high level tools for making mobile graphics applications for non-programmers (such as FlashLite), but none of them has been adapted for mobile AR yet. So there is a need to develop a high level mobile AR authoring tool for non-programmers.

Table 2.2: Authoring Tools for Mobile Phones

	Programmers	**Non-programmers**
Low level	Studierstube Tracker M3GE ARToolkit for Symbian	
High level	Edgelib Studierstube ES	Python FlashLite

2.3 Advertising Using Mobile AR

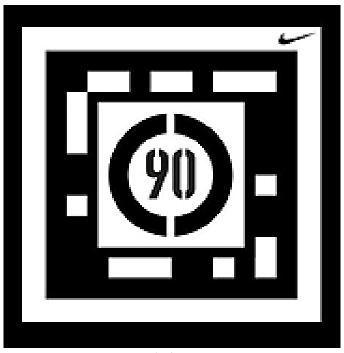
Mobile phones can do more and more in our lives, not only sending messages or making voices call, but also listening to music, watching videos or gaming. They also have other features such as GPS navigation, build-in cameras, WiFi connectivity, Bluetooth, Internet browsing and e-mail, and so on. One of the most interesting applications areas for mobile phones is advertising using Augmented Reality.

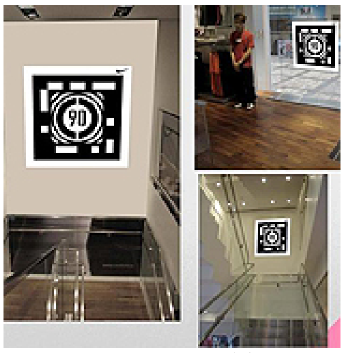
For example, early in 2009, Nike ran a mobile AR advertising campaign to target teens in Hong Kong to promote the launch of the T90 soccer shoe.

There were a series of hidden mobile codes all throughout Hong Kong in Nike flagship stores and at MTR subway stations (see Figure 2.12(a) and (b)). Once the consumers found the markers and pointed their camera phone at them, they received an image of a Nike soccer shoe and ball on their phone screen and revealed a special code unique to that location.

After getting the next location, consumers can find out the next secret destination by texting in these special codes. Texts, taken as a sweepstakes entry, could also win Nike merchandise. Users could use these codes to download a mobile application to view the T90 shoe from different angles in 3D through their mobile screens (see Figure 2.12(d)). Consumers had more chances to win Nike gear if they collected more codes.

Consumers could view the virtual product from a variety of different viewpoints because of the augmented reality technology, which enables the product to be revealed in a dynamic way.


(a) NikeT90 marker


(b) Nike flagship & MTR subway stations

(c) The real Nike shoe

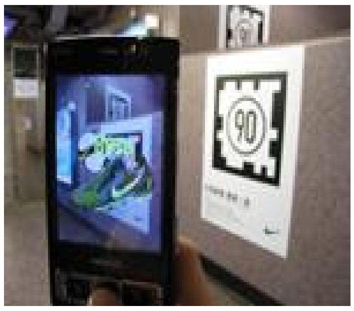
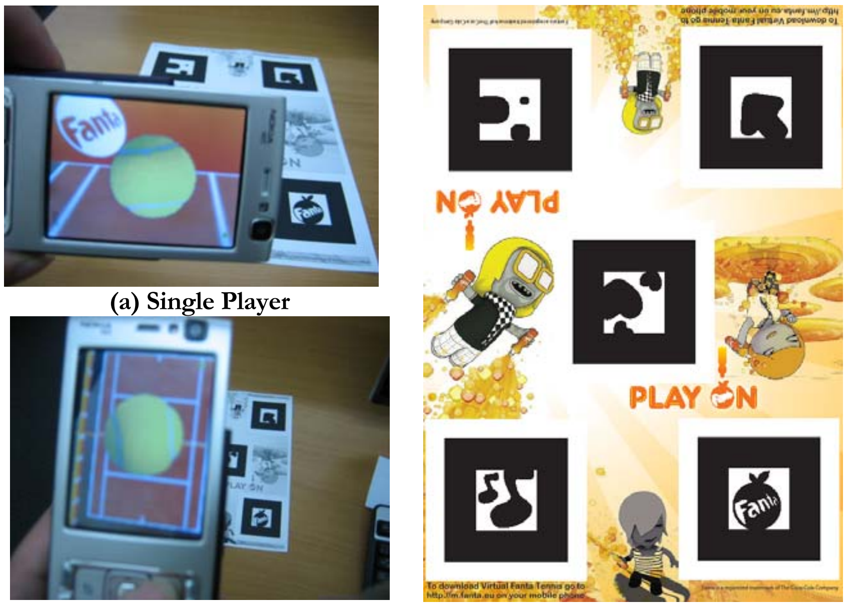

(d) Viewing through the phone

Figure 2.12: Nike 3D Mobile Soccer Shoe

In 2009, Coca-Cola Europe created a new mobile augmented reality advertising application in order to push its Fanta soft drink. This was the “Virtual Tennis” mobile game, the world's first 3D augmented reality tennis game. It offers two modes: a two-player mode which connects two phones so players can compete via Bluetooth; and a single-player practice mode where the player hits the ball off a wall.



(a) Single Player

(b) Two Players

(c) Court Image

Figure 2.13: Fanta Virtual Tennis Display

Players take their positions on either side of a printed “court” (see Figure 2.13). Once in position, players can see a virtual tennis court through their camera phones and hit the virtual tennis ball by using the phones as rackets. This ball movement is determined by the angle and position of the players’ phone.

A mobile AR campaign was also developed for the Ford ‘Ka’ aimed at targeting youth. In this case stickers were placed on the streets and the sides of buildings. Whenever the consumer used a camera phone to look at the sticker, a 3D virtual model of a Ka would appear on the phone screen (see Figure 2.14), appearing to float on top of the background video visuals in real time. A URL - GoFindIt.net - is displayed with the movement of the phone at particular angle.

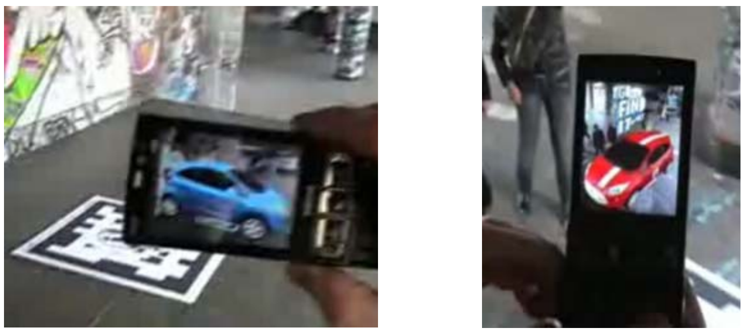


Figure 2.14: Ford ‘ka’ 3D Mobile Car Display

The first commercial AR application for advertising was developed in 2007. Mobile phone users were invited to download the software, and then pointed their phone to the marker image on the newspaper. A 3D model of a bear, cheetah, and giraffe will appear on the screen (see Figure 2.15). The benefit of advertisment, placed in a major newspaper, reached 750,000 people, leading to a 32% growth in visitors at Wellington Zoo[6].

[6] http://theinspirationroom.com/daily/2007/augmented-reality-at-wellington-zoo/

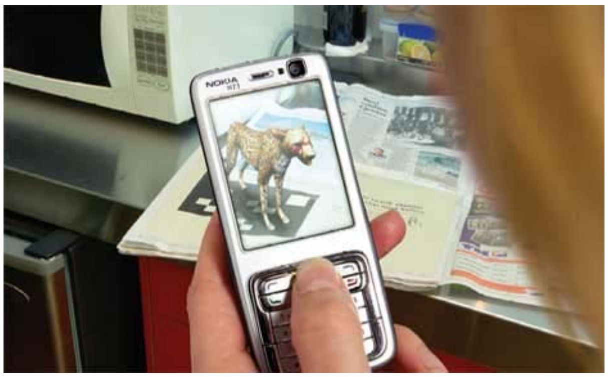


Figure 2.15: Augmented Reality at Wellington Zoo (Image from HITLab)

Each of these advertising examples required a significant amount of programming effort to implement. The goal of our work is to make a tool that non-programmers, such as advertising content people, could use to rapidly prototype mobile AR applications.

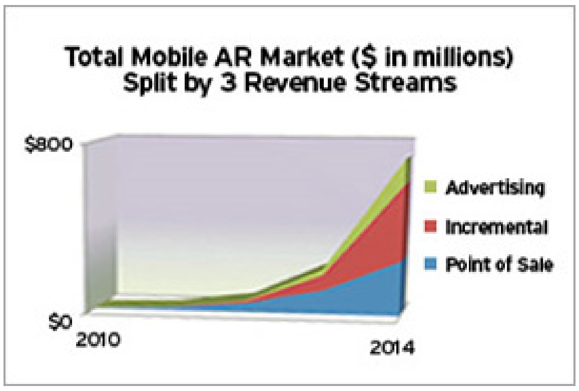


Figure 2.16: Mobile AR Market (Image from Mark Walsh)

As shown in Figure 2.16, "AR is still a long way from being a widespread reality on mobile devices" (Walsh, 2009). However, the authoring tool will be needed in the mobile AR market.

3 Chapter 3 AR Authoring System

The AR authoring tool presented in this thesis is designed to help people who have no experience in programming to create their own AR scene. For example, this tool may help people in the marketing or adverting industries to make simple demonstrations to show their clients.

There are two main components of the AR authoring system: the AR authoring tool and the AR viewing tool. In this chapter, we first overview the whole system, and then introduce the AR authoring tool. Chapter 4 will describe the AR viewing tool in more detail.[7]

3.1 Overview of the System

Figure 3.1 shows the components of the system that we have developed. In the rest of the chapter we will describe these system components in more detail.

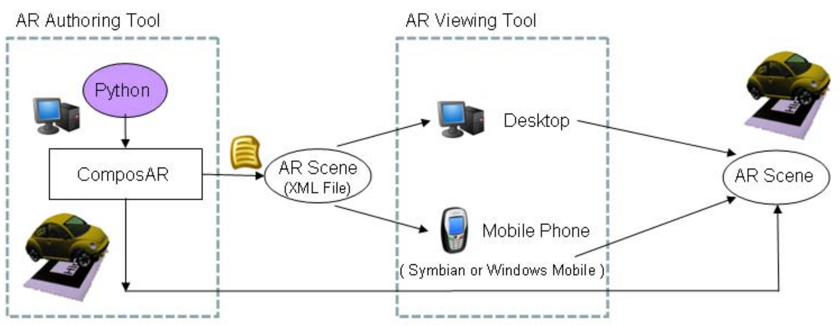


Figure 3.1: The Structure of the AR Authoring System

[7] An early version work has been published. WANG, Y., LANGLOTZ, T., BELL, T. & BILLINGHURST, M. (2009) An Authoring Tool for Mobile Phone AR Environments. *Proc NZCSRSC 09, New Zealand Computer Science Research Student Conference.* Auckland, New Zealand.

The authoring system we have developed is based on a modified version of the ComposAR tool. ComposAR is a PC application that allows users to easily create AR scenes. It is based on the osgART, ARToolKit and wxWidgets libraries.

The Python language is used to develop the overall ComposAR system. The user interface and runtime behavior are not only easy to customize, but also easy to test the 3D modules in the authoring environment. This is because ComposAR provides a Python based scripting tool for specifying the virtual objects in an AR scene. Python can also be used to modify the ComposAR interface to create a tool for mobile AR scene authoring.

In addition to creating a PC based authoring tool, there will also need to be a mobile AR viewing tool so that the applications developed can be run on a mobile phone. For this we will use a multi-platform game engine that can run on both PCs and mobile phones. The users will quickly prototype AR applications on both type devices.

ComposAR creates an XML file output which specifies the AR scene content and interaction in the application. In our work, we create an AR viewing application based on the viewing tool library that will read the XML file and render the AR scene on a mobile platform. Thus the user will be able to author the application on a PC and run it on a mobile phone.

3.2 AR Authoring Tool

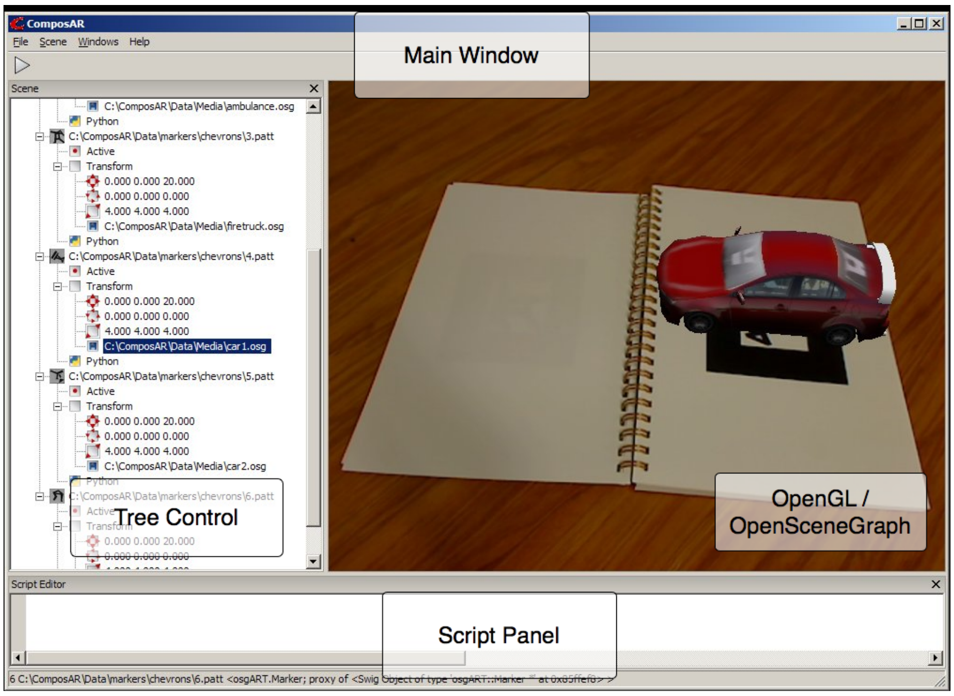
The AR authoring tool is an authoring tool that allows the user to create an AR scene which associates virtual content with real objects and defines interactions for those objects. The HIT Lab NZ has developed a tool for the PC for authoring AR applications called ComposAR. Firstly, we describe the ComposAR tool in general, and then discuss how we customized ComposAR for mobile AR authoring.

3.2.1 ComposAR

ComposAR (Seichter et al., 2008) is written in Python by using various extension libraries. The overall goal of the design is to keep ComposAR a pragmatic tool revealing its advanced features only on demand. In order to hide technical aspects such as the projection matrix or the scene-graph branch for the video background, the AR component of the application is emphasized. It focuses the user's authoring attempts on the marker content and their transformations.

These are accessed through a tree layout. A node in the tree structure can be activated with a single click, highlighting the respective 3D scene object and showing manipulation handles. Activating a node facilitates editing it, which includes virtual object file loading, or manual entry of transformation data, such as the translation, rotation and scale.

ComposAR provides a graphical user interface (GUI) divided into three panels (see Figure 3.2). This GUI was implemented in wxPython which is a wrapper for the cross-platform GUI and system development toolkit wxWidgets. In a similar way to wxPython, osgPython[8], a comprehensive wrapper for the OpenSceneGraph, was developed. The plugins for ARToolKit, the GPL version of osgART (Looser et al., 2006) with the respective bindings, and various video input sources, is within this package. However, the wide variety of plugins is not only available for OpenSceneGraph, but also for database loading and writing.



```
C:\ComposAR\Data\Media\ambulance.osg
Python
C:\ComposAR\Data\markers\chevrons\3.patt
  Active
  Transform
    0.000 0.000 20.000
    0.000 0.000 0.000
    4.000 4.000 4.000
    C:\ComposAR\Data\Media\firetruck.osg
  Python
C:\ComposAR\Data\markers\chevrons\4.patt
  Active
  Transform
    0.000 0.000 20.000
    0.000 0.000 0.000
    4.000 4.000 4.000
    C:\ComposAR\Data\Media\car1.osg
  Python
C:\ComposAR\Data\markers\chevrons\5.patt
  Active
  Transform
    0.000 0.000 20.000
    0.000 0.000 0.000
    4.000 4.000 4.000
    C:\ComposAR\Data\Media\car2.osg
  Python
C:\ComposAR\Data\markers\chevrons\6.patt
  Active
  Transform
    0.000 0.000 20.000
    0.000 0.000 0.000
```

```
6 C:\ComposAR\Data\markers\chevrons\6.patt <osgART.Marker; proxy of <Swig Object of type 'osgART::Marker *' at 0x05ffefd> >
```

Figure 3.2: ComposAR Interface Components (Image from Seichter et al.)

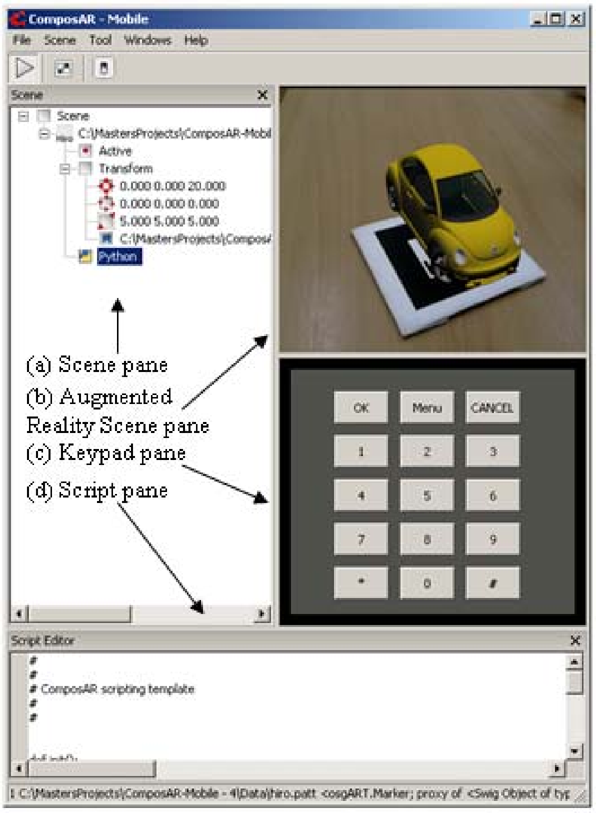
[8] OsgPython, http://code.google.com/p/osgswig, online as of September 2008

Seichter et al. described ComposAR as providing “some basic interaction approaches based on a standard repertoire common in AR applications, including interaction based on fiducially proximity, occlusion, tilting and shaking.” (Seichter et al., 2008)

3.2.2 ComposAR Customization

In order for ComposAR to be used for developing mobile AR applications, it needed to be modified to emulate the small screen size and limited input options of mobile phones. The Python language is used to develop the overall ComposAR system, so the ComposAR interface can be changed using Python code.

A simplified graphical user interface (GUI) for ComposAR was developed to match the form factor of the target mobile phone (see Figure 3.3). It is composed of a live video view of the scene with the same resolution as the typical mobile phone camera (320 × 240 pixel or 640 × 480 pixels), and a virtual keypad that emulates a mobile phone keypad. Camera input is taken from a webcam on the PC. With this GUI, the end-users can associate 3D virtual models with real AR tracking markers. In addition, it allows users to add simple keypad based interactions to the virtual scene.



```
#
#
# ComposAR scripting template
#
#
```

1 C:\MastersProjects\ComposAR-Mobile - 4\Data\hiro.patt <osgART.Marker; proxy of <Swig Object of typ

Figure 3.3: The ComposAR-Mobile Interface

As shown in Figure 3.3, there are four panes in the new ComposAR interface:

(a) Scene pane

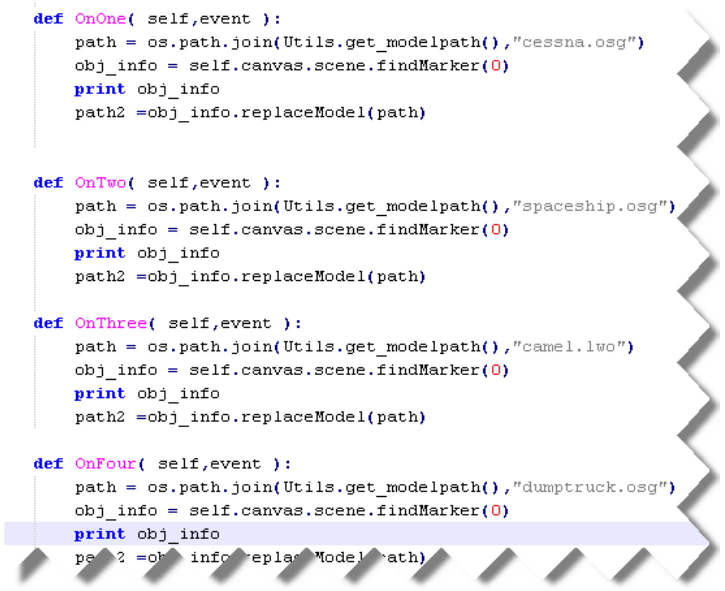
(b) Augmented Reality Scene pane

(c) Keypad pane

(d) Script pane

The Scene pane enables the designer to select markers and 3D models stored on the local system and to create links between markers and 3D models. Once the marker and corresponding 3D models are linked it is possible to change the position, rotation and scale of the assigned 3D models in the AR scene. An interaction script in Python can be written in the Script pane for specifying the virtual object interactions in the AR scene. Keypad based interaction within the AR scene can be simulated using the virtual keypad in the Keypad pane.

The keypad panel has several functions, such as scaling and rotating the 3D models, and browsing all 3D models within the file folder. An example code for integrating the keypad pane in the python code is shown in Figure 3.4.

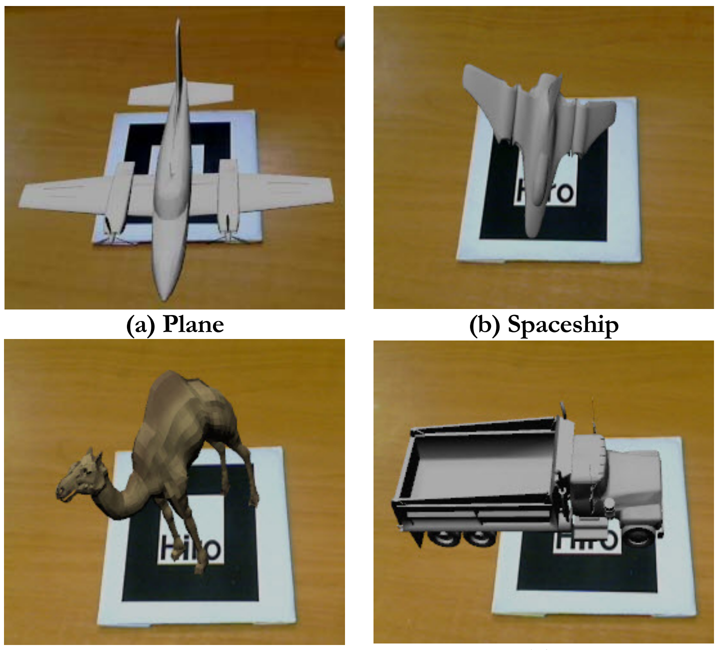


```
def OnOne( self,event ):
    path = os.path.join(Utils.get_modelpath(),"cessna.osg")
    obj_info = self.canvas.scene.findMarker(0)
    print obj_info
    path2 =obj_info.replaceModel(path)


def OnTwo( self,event ):
    path = os.path.join(Utils.get_modelpath(),"spaceship.osg")
    obj_info = self.canvas.scene.findMarker(0)
    print obj_info
    path2 =obj_info.replaceModel(path)

def OnThree( self,event ):
    path = os.path.join(Utils.get_modelpath(),"camel.lwo")
    obj_info = self.canvas.scene.findMarker(0)
    print obj_info
    path2 =obj_info.replaceModel(path)

def OnFour( self,event ):
    path = os.path.join(Utils.get_modelpath(),"dumptruck.osg")
    obj_info = self.canvas.scene.findMarker(0)
    print obj_info
    pa  2 =ob  info  eplac  Model  ath)
```

Figure 3.4: The Python Code for Keypad Functionality

In example 3.4, when Buttons “1”, “2”, “3”, “4” are clicked in turn, the corresponding 3D model will appear on the marker (as seen in Figure 3.5):

- Button One: A plane (Figure 3.5a)
- Button Two: A spaceship (Figure 3.5b)
- Button Three: A camel (Figure 3.5c)
- Button Four: A truck (Figure 3.5d)

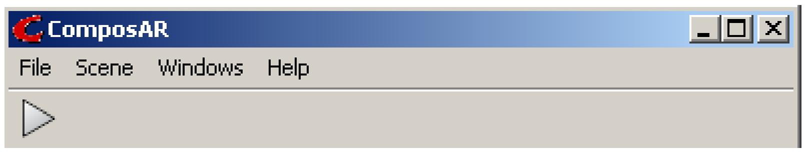

(a) Plane **(b) Spaceship**

(c) Camel **(d) Truck**

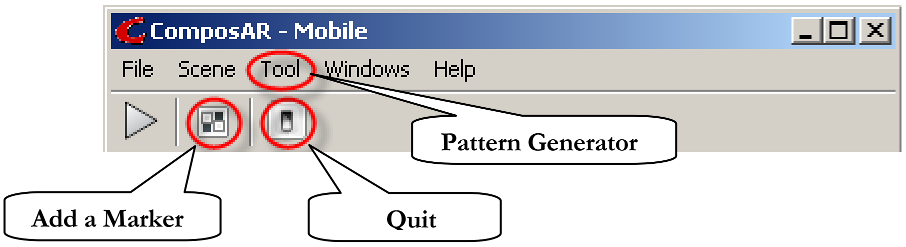
Figure 3.5: Different 3D Model on the Marker When the Button is Clicked

Another scripts example (see Appendix A) is for the rotation function. When Buttons “*” or “#” is clicked in turn, the corresponding 3D model will rotate on the marker.

In order to make this tool easy to use, the interface has been changed a little. Two icons and one tool have been added in a toolbar below the menu panel. As shown in Figure 3.6, users can add a marker more easily than before by clicking on a button in the toolbar. The pattern generator allows the users to create their own markers. This will be described in detail in Chapter 5.



(a) ComposAR



(b) Modified ComposAR

Figure 3.6: Different Interface of ComposAR

One of the advantages of using ComposAR is that scripts are interpreted, so that immediate feedback on the fly can be seen from the Augmented Reality Scene panel if any of the content is updated.

4 Chapter 4 AR Viewing Tools

In addition to replaying the created AR scene from the PC based authoring tool, a mobile AR viewing tool was needed. The loaded XML file was produced by the ComposAR tool and rendered it onto a live video stream to create AR viewing on the mobile phone. This will provide a mobile authoring tool that a person can use to author an AR application on a PC and run on a mobile phone.

In this chapter, we will describe the AR viewing applications that we have developed for desktop and mobile AR applications.

There are two different desktop AR viewing tools that we have developed during this research. These are used to load the AR scenes created in the ComposAR application and test them before running them on a mobile phone. The first prototype is based on the Edgelib library, and a second later prototype is based on the Studierstube ES library. Each of these software libraries has its own advantages and disadvantages. In the following sections we will describe the Edgelib and Studierstube applications respectively in more detail.

4.1 Edgelib

Edgelib (Edgelib, 2008) is a powerful C++ multi-platform game engine for mobile devices. It enables users to develop high-quality applications and high-performance games through different platforms, such as Windows

Mobile phone, Symbian phone, Linux/Windows desktop, Apple iPhone or iPod Touch, and so on.

Edgelib has two key features: One is a device independent API for Multi-platform development, and the second is a device independent API for high-performance graphics.

4.1.1 Multi-platform Deve1opment

Edgelib features a true multi-platform independent API. It operates as a generic interface and makes use of all of its key features for all supported platforms. Edgelib currently supports the platforms shown in Table 4.1.

Table 4.1: Platforms Supported by Edgelib

Mobile Phone						
Windows Mobile		Symbian Series			Apple	
Pocket PC	Smartphone	Series 60	Series 80	Series 90	iPhone	iPod Touch
		N-Gage™ 6680 E60 N95	92xx 9300 9500	7710		

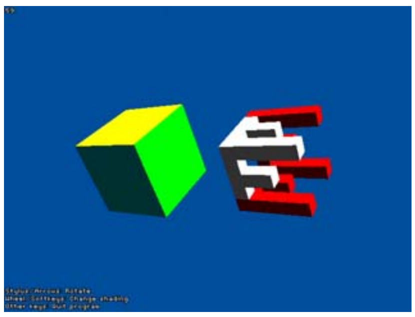
Desktop		Game Console	
Windows	Linux	GPH[9]	Gizmondo
2000/XP/Vista	X11	F-100/F-200	Windows CE

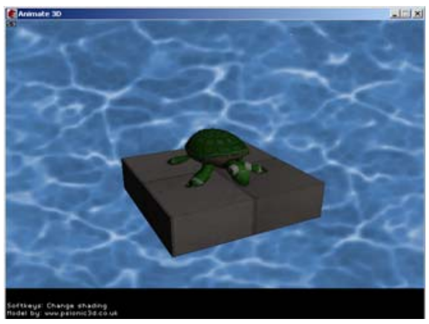
[9] GPH: GamePark Holding

For Windows Mobile all kinds of screen resolutions are supported, including 176×220, VGA, WVGA, QVGA, and QWVGA. The new screen resolutions for Symbian (such as 320×240 and 352×416) are also fully supported.

4.1.2 High-performance Graphics

The Edgelib library can draw 2D and 3D graphics on each device in full screen mode. 3D models contain a vertex list that is linked into polygons. Models can be created manually or loaded from 3D Studio Max (.3ds), MilkShape 3D (.ms3d) or Edgelib 3D (.e3d) files. Even animated 3D models can be loaded. Models can be drawn by using either OpenGL ES or Edgelib's fast internal 3D renderer. The Edgelib animation functions support translation (movement) and rotation animations. This internal platform uses an independent 3D engine if OpenGL ES is not available. Figure 4.1 shows screen shots of sample Edgelib applications.


(a) Simple 3D objects



(b) 3D objects with texture mapping

Figure 4.1: Screen Snapshots of 3D Objects from Edgelib

In Figure 4.1(a), the objects are drawn with no shading, while an animated turtle MilkShape 3D model is shown in Figure 4.1(b). The turtle can move and rotate on the texture mapping block. The different views (front, side, top and whole model) of this turtle 3D model are shown in Figure 4.2.

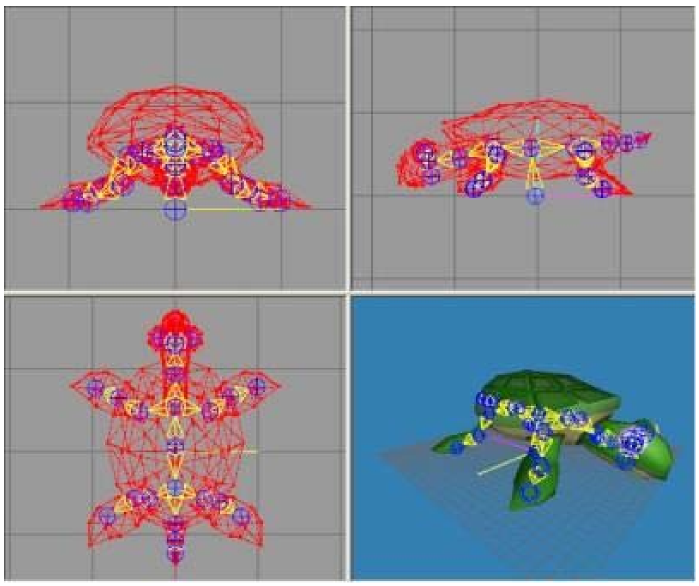


Figure 4.2: Turtle 3D Model in MilkShape 3D. Model by psionic3d.co.uk

There are total 150 frames of this animated turtle 3D model. Figure 4.3 displays four frames of it. This turtle 3D model can be rendered very fast in the Edgelib application. The frame rate on the screen is 59-60 fps.

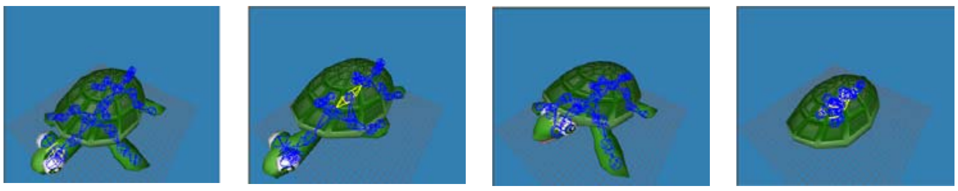


Figure 4.3: Four Frames of an Animated Turtle Model in MilkShape 3D

The pictures (Figure 4.4) below demonstrate each rendering method: wireframe, no shading, flat shading, gouraud shading, and texture mapping. They are the key features in the internal 3D engine.

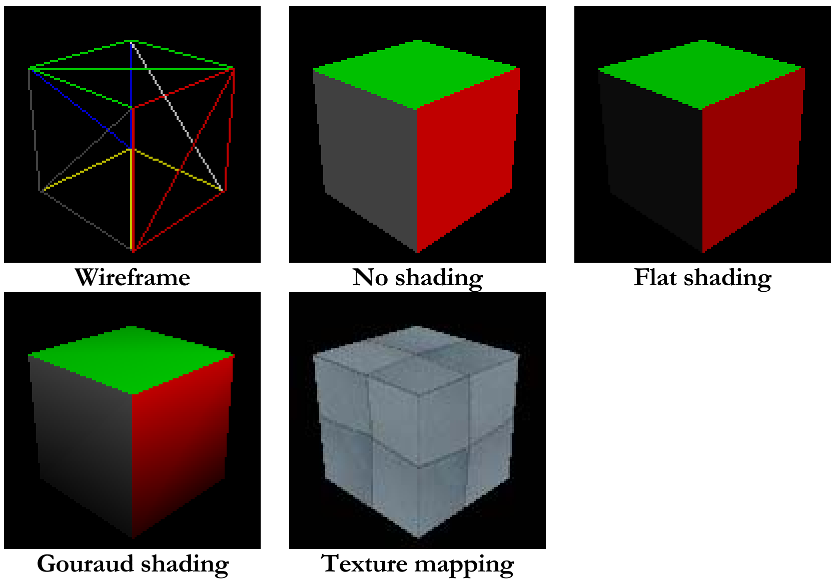


Figure 4.4: Rendering Methods in Edgelib (Image from Edgelib website)

4.1.3 Edgelib Integration with ARToolKit on a Desktop PC

Since both the ARToolKit and the Edgelib libraries run on Symbian and Windows Mobile phones, and on the desktop, any desktop AR applications built by using these libraries should be able to be ported to a mobile phone. However, video capture from a camera is currently not supported by the Edgelib libary, so in order to develop an AR viewer application we need to integrate the Edgelib library with the ARToolKit tracking library for AR tracking.

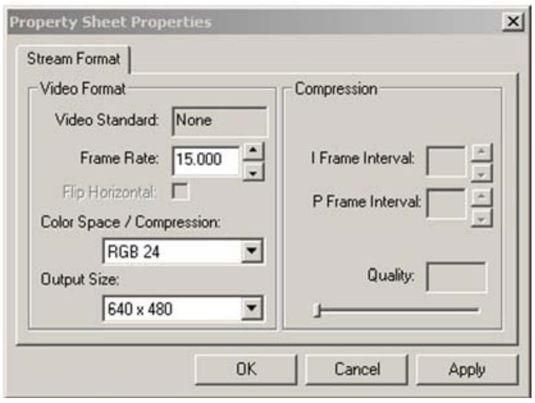
The image frames of the video source captured from an attached camera need to be made available to the tracking component and scene graph. The image frames are needed not only by the tracker in order to locate markers and calculate transformations, but also by the scene graph to display the real world behind the virtual objects. Once the tracker locates a marker and calculates its transformation, that information is transferred within the scene graph.

The tracker provides a projection matrix which determines a perspective projection used to display the 3D graphics. The intrinsic camera parameters which the tracker uses are necessary to accurately track markers, and the correct projection matrix is needed for the resulting transformations to align the virtual objects with the live video. The video frames captured from the camera can be made to be continuously uploaded into a texture as the background. Figure 4.5 shows the sample code to copy background data to the Edgelib surface.

```
/* A pointer to the memory data of the locked surface.*/
unsigned char *memptr = background.Lock(&info);
        /* If this is not NULL, it will be filled with detailed surface information. */
        if (memptr) {
                unsigned long yctr;
                /*The cparam variables need to be replaced by your own variables to
        determine the properties of the captured image. */
                for (yctr = 0; yctr < (unsigned long)cparam.ysize; yctr++) {
                        ClassEMemory::Copy( &memptr[yctr * info.realpitch],
                        &dataPtr[yctr * cparam.xsize * info.bitwidth / 8],
                        cparam.xsize * 32 / 8 );
                }
                /* This unlocks a previously locked surface */
                background.Unlock();
        }
/* Draw the background picture*/
DrawBackground(display);
```

Figure 4.5: Sample Code for Edgelib Surface Setting

The settings for the attached camera settings can be seen in Figure 4.6. In this case the frame rate is automatically set to 15 FPS (frames per second) and the output screen size is 640 × 480 pixels with the setup.


Property Sheet Properties

Stream Format

Video Format

Video Standard: None

Frame Rate: 15.000

Flip Horizontal:

Color Space / Compression: RGB 24

Output Size: 640 x 480

Compression

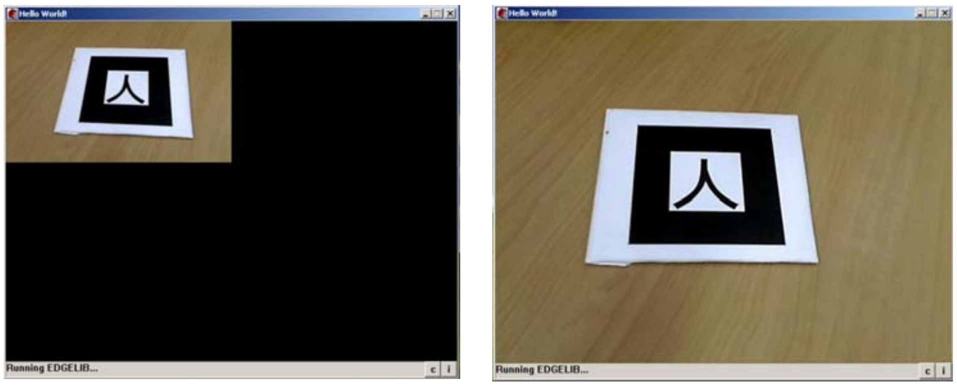
I Frame Interval:

P Frame Interval:

Quality:

Figure 4.6: Camera Setting Pop-up Window

Once the camera captures the images, we can lock the back buffer and manually copy the pixel data to the surface. It is very important to set the camera field of view in Edgelib to the same as the real camera connected to the desktop, otherwise we will get an incorrect display as shown in Figure 4.7(a). This resulted from a setting of 640 × 480 pixels in Edgelib whereas the real camera view was 320 × 240 pixels. Since they are not the same view size, the display is incorrect. However, as shown in Figure 4.7(b), the correct results are produced when both Edgelib and the real camera view are set to 640 × 480 pixels.



(a) Wrong setting **(b) Correct setting**

Figure 4.7: Camera View Setting for the Edgelib

Edgelib used the following command:

```
ERESULT OnDisplayConfig(EDISPLAYCONFIG *config)
```

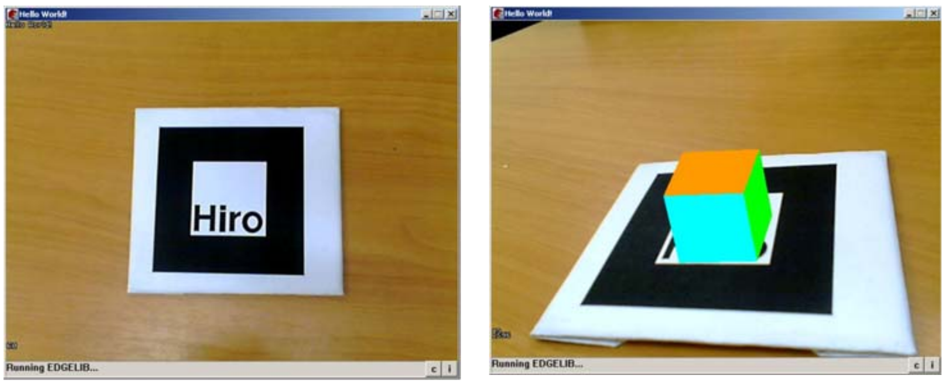
to configure the display properties. The default width and height setting for the desired resolution or size of the window for Windows desktop applications is 240 × 320 pixels.

There are many view sizes for the camera, for example: 160 × 120 pixels, 320 × 180 pixels, 320 × 240 pixels, 640 × 480 pixels. In order to make the application flexible, we want to use the real camera to set up Edgelib view rather than set up the default value which is 640 × 480 pixels. Figure 4.8 shows the code we wrote for setting up the virtual camera in Edglib.

```
//Configure display
ERESULT ClassMain::OnDisplayConfig(EDISPLAYCONFIG *config)
{
	ARParam  wparam;
	ClassEStd::StrCpy(config->caption, "Hello World!");
	config->icon = IDI_MAIN;

	  /* open the video path */
	if( arVideoOpen( vconf ) < 0 ) exit(0);
	/* find the size of the window */
	if( arVideoInqSize(&xsize, &ysize) < 0 ) exit(0);
	printf("Image size (x,y) = (%d,%d)\n", xsize, ysize);

	/* set the initial camera parameters */
	if( arParamLoad(cparam_name, 1, &wparam) < 0 ) {
	       printf("Camera parameter load error !!\n");
	       exit(0);
	}
	arParamChangeSize( &wparam, xsize, ysize, &cparam );
	arInitCparam( &cparam );
	printf("*** Camera Parameter ***\n");
	arParamDisp( &cparam );

	if( (patt_id=arLoadPatt(patt_name)) < 0 ) {
	      printf("pattern load error !!\n");
	      exit(0);
	}

	/* open the graphics window */
	argInit( &cparam, 1.0, 0, 0, 0, 0 );
	arVideoCapStart();

	/*Desktop resolution (we an change the background size to mobile
phone 320*240 or 260*220)*/
	config->width =cparam.xsize;
	config->height =cparam.ysize;

	/*Other options*/
	config->orientation = DOR_AUTO;
	config->engineconsole = false;

	return(E_OK);
}
```

Figure 4.8: An Example Code to Set Up the Virtual Camera

Figure 4.9(b) shows a virtual cube with six different colours, which has been programmed to appear on a single marker. It is a snapshot of the AR viewer using Edgelib which integrates with ARToolKit.



(a)Video test screenshot **(b)Simple AR application created with Edgelib**

Figure 4.9: Prototype AR Viewer Using Edgelib

4.1.4 Input XML File

The AR viewer should load the XML file produced by the ComposAR tool and render it onto a live video stream to create an AR view. The XML file contains all the information about the AR scene including the virtual objects and their transformation. Unfortunately, Edgelib only uses an XML-RPC node[10] which is a single element in the data tree to store data. The node can be used to access the parent, children and siblings easily, but it has only limited flexibility because of the simplicity of its architecture,

[10] Edgelib SDK 3.60 released on March 25, 2008

leading to potential complexity as the number of different requests increases.

To represent the values, XML-RPC defines several basic data types, such as integers, floating-point numbers, Boolean values, strings, date-times and Binary, and for compound data structures, such as arrays and structs.

In this research, we use the data type XML-RPC arrays, which are best thought of as untyped lists because the data items within an array may be of any type, simple or compound. However, the data items we use are not the same types; they can be represented as multidimensional arrays by embedding an array within an array.

In general, elements in a normal XML document may contain mixed contents which can be both text and other elements. For example, `<animal name="dog" legs="4"/>`

However, XML-RPC does not use this feature. To keep things simple, it uses only elements. Elements in XML-RPC contain either text-only text or other elements only. The example above should be: `<animal><name>dog</name><legs>4</legs></animal>`

XML-RPC also defines the XML payload, which contains information about the method to invoke. The very important part is the parameter list enclosed by the `<params>` element which may contain zero or more

<param> elements. Even if the method requires no parameters, the <params> element must still be present. The following example in Figure 4.11 shows how the different platforms can be used in Edgelib. Its structure is shown in Figure 4.10.

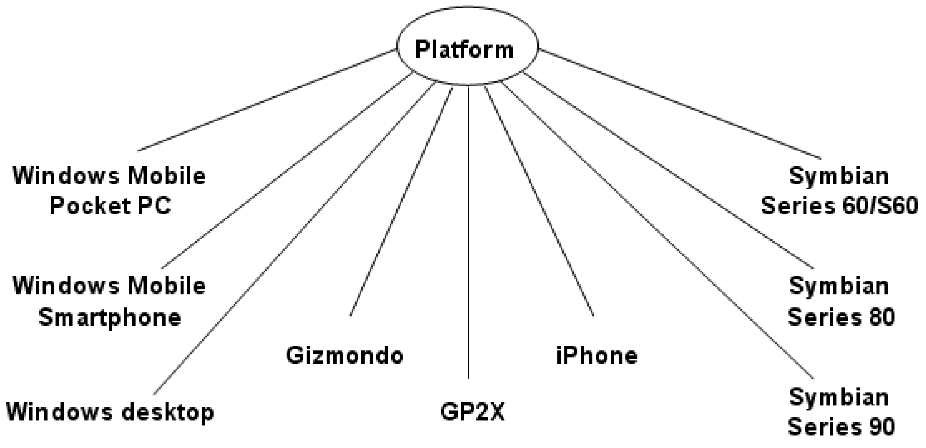


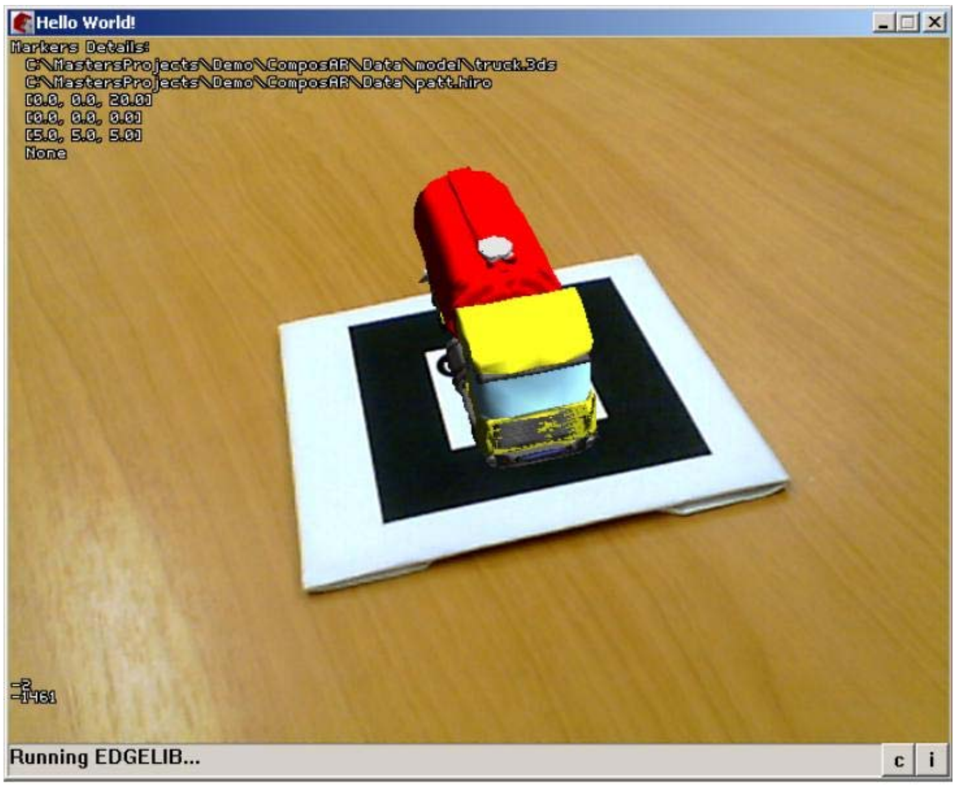
Figure 4.10: An Example of XML-RPC Structure

As we known, Windows Mobile has Pocket PC and smartphone. But in the example, we have to use two child nodes instead of one category by using XML-PRC structure. The same situation is for the Symbian phone. We use three nodes for the Symbian phone category.

```
   <?xml version="1.0" encoding="iso-8859-1" ?>
- <methodResponse>
 - <params>
  - <param>
   - <value>
    - <struct>
     - <member>
        <name>Platform</name>
      - <value>
       - <array>
        - <data>
         - <value>
              <string>Windows Mobile Pocket PC</string>
           </value>
         - <value>
              <string>Windows Mobile Smartphone</string>
           </value>
         - <value>
              <string>iPhone</string>
           </value>
         - <value>
              <string>Symbian Series 60 / S60</string>
           </value>
         - <value>
              <string>Symbian Series 80</string>
           </value>
         - <value>
              <string>Symbian Series 90</string>
           </value>
         - <value>
              <string>GP2X</string>
           </value>
         - <value>
              <string>Gizmondo</string>
           </value>
         - <value>
              <string>Windows desktop</string>
           </value>
         </data>
        </array>
       </value>
      </member>
     </struct>
    </value>
   </param>
  </params>
</methodResponse>
```

Figure 4.11: An Example Code for the XML-RPC Structure

In order to use Edgelib, we have to let ComposAR export the XML-RPC format. Figure 4.13 shows an example for an output XML file which has many tags in it created by the ComposAR tool. It contains the truck model (see Figure 4.12) information and the marker information, which specifies the file of the virtual object and its location, which marker associate with, and the virtual object translation, rotation, and scale.



```
Markers Details:
  C:\MastersProjects\Demo\ComposAR\Data\model\truck.3ds
  C:\MastersProjects\Demo\ComposAR\Data\patt.hiro
  [0.0, 0.0, 20.0]
  [0.0, 0.0, 0.0]
  [5.0, 5.0, 5.0]
  None
```

```
-2
-1461
```

```
Running EDGELIB...
```

Figure 4.12: A Truck Model in the AR Viewer

```
<?xml version="1.0" ?>
- <composar os="nt" utc="Thu, 21 May 2009 16:30:34 +0000" version="0.1">
  - <scene>
      <videos />
      <trackers />
    - <markers>
      - <params>
        - <param>
          - <member>
              <name> marker</name>
            - <marker>
              - <value>
                - <array>
                  - <model>
                    - <value>
                        <string>C:\MastersProjects\Demo\ComposAR\Data\model\truck.
 3ds</string>
                      </value>
                    </model>
                  - <name>
                    - <value>
                        <string>C:\MastersProjects\Demo\ComposAR\Data\patt.hiro</st
 ring>
                      </value>
                    </name>
                  - <position>
                    - <value>
                        <string>[0.0, 0.0, 20.0]</string>
                      </value>
                    </position>
                  - <rotation>
                    - <value>
                        <string>[0.0, 0.0, 0.0]</string>
                      </value>
                    </rotation>
                  - <scale>
                    - <value>
                        <string>[5.0, 5.0, 5.0]</string>
                      </value>
                    </scale>
                  - <script>
                    - <value>
                        <string>None</string>
                      </value>
                    </script>
                  </array>
                </value>
              </marker>
            </member>
          </param>
        </params>
      </markers>
    </scene>
  </composar>
```

Figure 4.13: XML File for the Truck Model from ComposAR Tool

4.2 Studierstube ES

Studierstube ES (StudierstubeES, 2008) is a general handheld AR platform. It enables handheld devices to run AR applications. It has cross-platform support for a variety of platforms, such as Windows XP/Vista, Windows CE/Mobile, Gizmondo, Mac OS/iphone, Linux and Symbian series 60.

Studierstube ES has components for addressing graphics, handling video, tracking, rendering, and so on. In order to make an application that can be run independently of any infrastructure and scale to an arbitrary number of simultaneous users, processing is done natively on the handheld device. Typical frame rates on smartphones are in the order of 5-30 fps, depending on the content and device (Dieter and Daniel, 2008).

The structure of Studierstube ES is shown in Figure 4.14. There are two general levels in the software stack of the Studierstube handheld AR framework. The lower levels such as Core, Math, IO, Tracker, and Muddleware provide the basic functionality that an AR system requires. Studierstube ES (Embedded System) and SG (Scene-Graph) combine these services in a high-level layer for the applications running on top of it.

The core components essential for AR are the Studierstube Tracker which is a real-time fiducial tracking component, and Studierstube Scene Graph which is a rendering engine running on top of OpenGL ES or Direct3D Mobile.

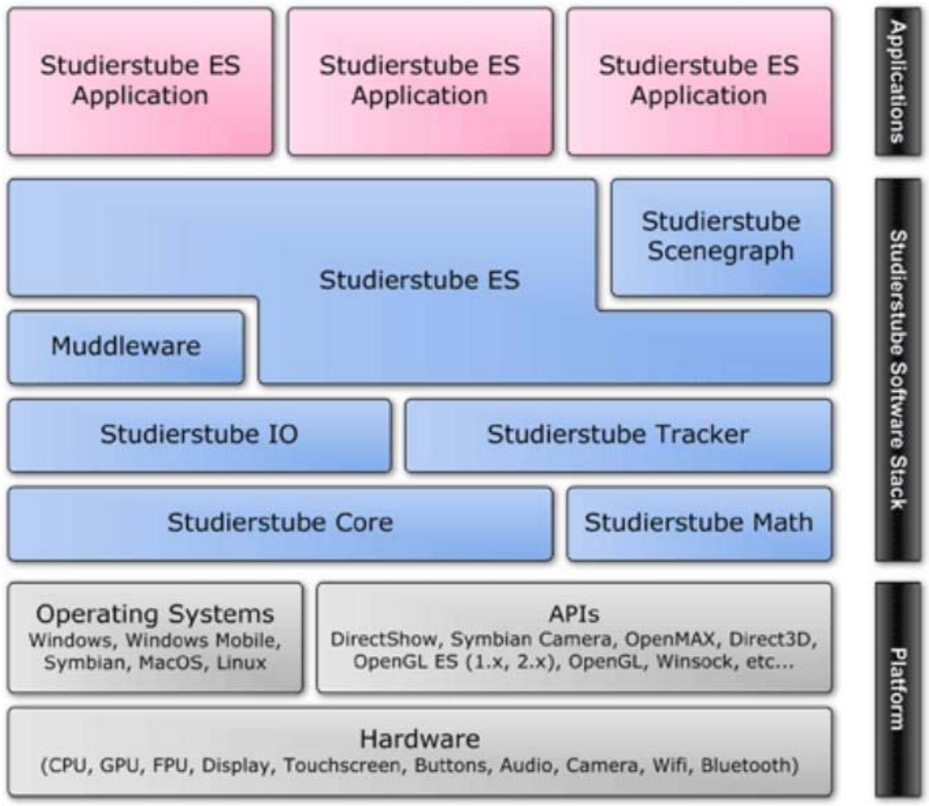

Studierstube ES Application
Studierstube ES Application
Studierstube ES Application
Applications
Studierstube ES
Studierstube Scenegraph
Muddleware
Studierstube IO
Studierstube Tracker
Studierstube Core
Studierstube Math
Studierstube Software Stack
Operating Systems
Windows, Windows Mobile, Symbian, MacOS, Linux
APIs
DirectShow, Symbian Camera, OpenMAX, Direct3D, OpenGL ES (1.x, 2.x), OpenGL, Winsock, etc...
Hardware
(CPU, GPU, FPU, Display, Touchscreen, Buttons, Audio, Camera, Wifi, Bluetooth)
Platform

Figure 4.14: Structure of Studierstube ES (from Studierstube ES Website)

Studierstube Tracker supports a wide variety of markers (see Figure 4.15): Template marker, ID marker, DataMatrix marker, Frame marker, Split marker, and Grid marker. In our research, we use the Template marker which recognizes an image either colour or not placed inside a black rectangle.

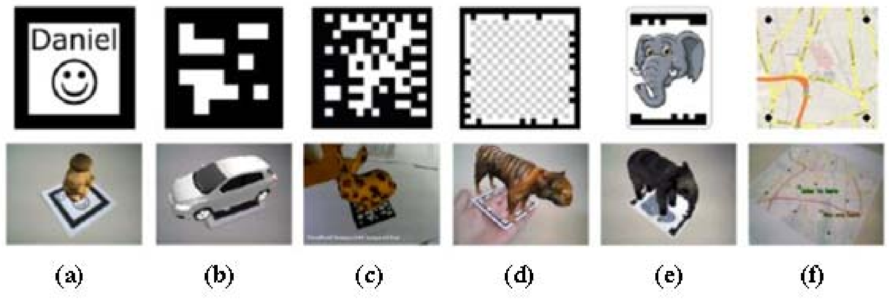


Figure 4.15: Different Markers Types Supported by StbTracker (a) Template Marker (b) ID Marker (c) DataMatrix Marker (d) Frame Marker (e) Split Marker (f) Grid Marker

One of the components of StbES can work with XML files. It uses a modified version of the TinyXML[11] library for parsing, loading and saving XML files (Yeoh, 2005).

The four main tasks that this research is focused on are:

- how to make the configuration,
- how to set up the video,
- how to configure the tracking subsystem,
- how to set up scene graph.

[11] TinyXML is a simple, small, C++ XML parser that can be easily integrated into other programs.

4.2.1 Configuring StbES

StbES can be configured by using XML configuration files. However, the main configuration of StbES for Windows XP and Windows Mobile is almost the same. Both of them must set StbES as the root node, and they can have other nodes such as Tracker, Logging, Window, RenderTarget, Background, Video, Audio, Scene, WidgetManager, Application, and GUI. The screen rendering size is different between Windows XP and Windows Mobile, so the main differences between Windows XP and Windows Mobile are the RenderTarget and Video as in Table 4.2.

The most important node parameters are listed as follows:

- **Logging** is a tool for debugging the crashes. It lets the system create a log file which saves a lot of debug time since it provides information about errors and relevant events in the system at runtime.

- **RenderTarget** can create both software and hardware renderings. It not only makes off-screen render targets allowing direct video memory access, but also makes on-screen render targets for acceleration. “PixMap” is used when the target device does not feature dedicated video memory.

Table 4.2: The Important Node for the Windows XP and Mobile Platform

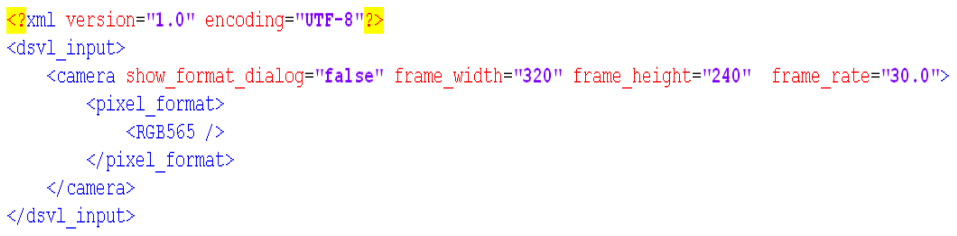
Platform / Important Node	Windows XP	Windows Mobile
StbES	must be the root node	
Logging	level = [OFF | ERROR | WARNING | INFO]	
	draw-fps = [true | false]	
	file = FILENAME	
Window	width	
	height	
	rotation	
	fullscreen= [true | false]	
RenderTarget	type = [PIXMAP | WINDOW]	type = [PIXMAP | VIDMEM]
Background	pixels = [OFF | IMAGE | COLOR]	
	colorValue = “1 0 0 1” (RGBA)	
Video	type = [IMAGE | DSVL]	type=[IMAGE | DSVideoCE]
	file =FILENAME	
Audio	enabled = [true | false]	
Scene	file = FILENAME	
WidgetManager	font, char-width, char-height	
Tracker	displayInfos= [true | false]	
Application	name =NAME	
GUI	exitkey = CODE_OF_KEY	

- **Video** is a major importance node. It contains many types of the video source such as Gizmondo device. Alternatively developers can manually select a specific video mode and configure cropping, format conversion, and zooming.

4.2.2 Video Configuration

The video input can come from either a camera video or an .avi file. The video configuration file is different from Windows and Windows CE devices. However, they both use the "xml" tag in the first line which is just an information header.

The file **dsvl.xml** is used to configure the DirectShow Video Library for Windows. Figure 4.16 shows a sample video configuration file for Windows.



```
<?xml version="1.0" encoding="UTF-8"?>
<dsvl_input>
    <camera show_format_dialog="false" frame_width="320" frame_height="240"  frame_rate="30.0">
        <pixel_format>
            <RGB565 />
        </pixel_format>
    </camera>
</dsvl_input>
```

Figure 4.16: An Example of Windows Video Configuration File

In the sample code as shown in Figure 4.16, the camera setting is:

- The video frame size is 320 × 240
- The video frame rate is 30
- There is no dialog box shown there
- The color format of the camera is RGB565[12].

The file **dsvideoce.xml** is used to configure the DirectShow Video for the Windows CE library. It is only needed if a user wants to configure the video settings. Figure 4.17 shows a sample video configuration file for a Windows CE device.

```
<?xml version="1.0" encoding="UTF-8"?>
<DSVideoCE register-dll="DSVideoCE_Filter.dll">
    <Camera width="320" height="240" fps="15" format="yuv12" />
    <Output width="240" height="240" swizzle34="true" rotate="false" />
</DSVideoCE>
```

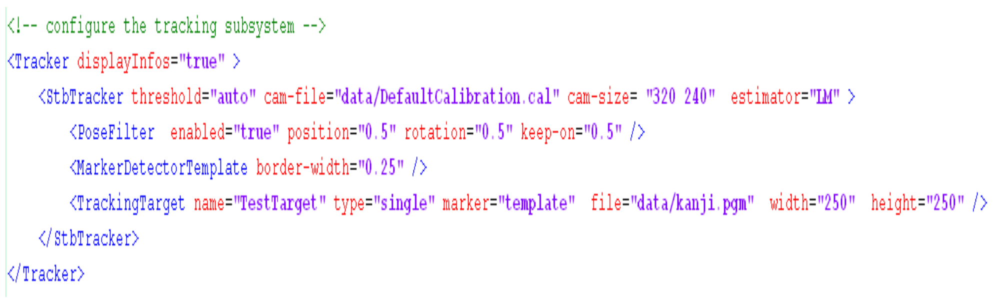
Figure 4.17: An Example of WindowsCE Device Video Configuration File

In this example code as shown in Figure 4.17, we use DirectShow to filter the standard DSVideoCE library. The “width” and “height” fields take the size of the camera image, which is 320 × 240 in our test configuration. The “width” and “height” fields define the output image size which is 240 × 240.

[12] The mode is used in many devices with color screens.

4.2.3 The Tracking Subsystem

StbTracker tracking is based on finding key features. The tracker automatically connects them in a pipeline: Once a tracker is created, then the features can register one by one, it passes the frame to the tracker, finally, the tracking results will gain. Figure 4.18 shows a sample configuration for the tracking subsystem.



```
<!-- configure the tracking subsystem -->
<Tracker displayInfos="true" >
    <StbTracker threshold="auto" cam-file="data/DefaultCalibration.cal" cam-size= "320 240"  estimator="LM" >
        <PoseFilter  enabled="true" position="0.5" rotation="0.5" keep-on="0.5" />
        <MarkerDetectorTemplate border-width="0.25" />
        <TrackingTarget name="TestTarget" type="single" marker="template"  file="data/kanji.pgm"  width="250"  height="250" />
    </StbTracker>
</Tracker>
```

Figure 4.18: An Example for Configuring the Tracking Subsystem

The “StbTracker” tag states the StbTracker tracking system. The system calculates the position of the camera relative to the position of a special marker. For the tracking, a calibration file which compensates for the lens distortion of the particular camera should be loaded. In the example as shown in Figure 4.18, we use the default calibration file on desktop for this research. The size of the video frames captured by the camera is 320 × 240.

“TrackingTargets” are here the fiducial markers, the ID of which has been registered with the tracking system, and which hold relevant application functionality. These targets are needed to calculate the relative camera pose to the marker. In this example, a single template marker is used.

As we mentioned in Section 4.2, the template marker allows placing an image inside the rectangle (see Figure 4.15). This marker layout image is the same as the marker we use in ComposAR authoring tool, although the file formats of the markers are quite different. In order to convince the end user, the pattern generator also can create different marker formats for each tool. This will be described in detail in Chapter 5.

4.2.4 Scene Graph File: scene.xml

The scene file contains the field connections to the tracking system and traverses the scene graph for rendering. Figure 4.19 shows a sample scene graph file.

```
<?xml version="1.0" encoding="UTF-8"?>
<Scene name="MyScene">
        <MatrixCamera projMatrix="REF StbTracker.projMatrix" />
        <TransformSeparator active="REF TestTarget.visible">
                <MatrixTransform matrix="REF TestTarget.matrix" />
                <LightSeparator>
                        <DirectionalLight direction="-1 -0.5 0.3" />
                        <Transform              name="PlayerTransform"
                        translation="0 0 40" />
                        <Cube width="80" height="80" depth="80" />
                </LightSeparator>
        </TransformSeparator>
</Scene>
```

Figure 4.19: An Example of Scene Graph File

In the example code, the projection matrix of a real camera was used, and loaded the projection matrix which is needed to project the 3D objects into 2D image space. The value of the “projMatrix” field is part of the “Tracker” node. The node and its fields are defined in the “config.xml” file. The transformation separator is currently in use in this example. It is only affects the objects inside the enclosure. “MatrixTransform” loaded the transformation matrix of the marker called “TrackingTarget”, so that the position and orientation of 3D objects defined below depend on the marker’s transformation.

In the example, a simple cube with an edge length of 80 is displayed on the marker, see Figure 4.20.

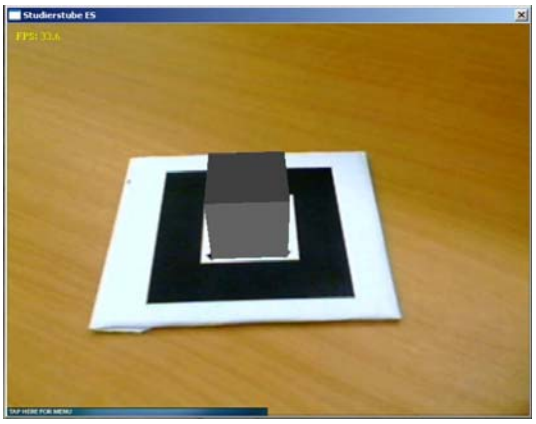


Figure 4.20: Screenshot of the Example

5 Chapter 5 AR Pattern Generator

The AR Pattern Generator is a tool for producing the marker for this AR application. This is a modified vision from BuildAR (Looser, 2008). It not only can output the pattern file, but also can print out the image file as a marker. This is very convenient for the user who wants to create their own markers for the AR application.

In this research, there are two types of markers used: ComposAR uses the ARToolKit marker with the file format is *.patt* file, and the StbES uses the template marker with the file format is *.pgm* file. However, both of them have one common feature: the image is inside the black square. The important thing is that the markers' pattern file must use the same name, but different file type for the two applications.

5.1 User Interface

The background is an 800 width by 800 height black rectangle with 400 width by 400 height white rectangle centered in the middle, as shown below in Figure 5.1.

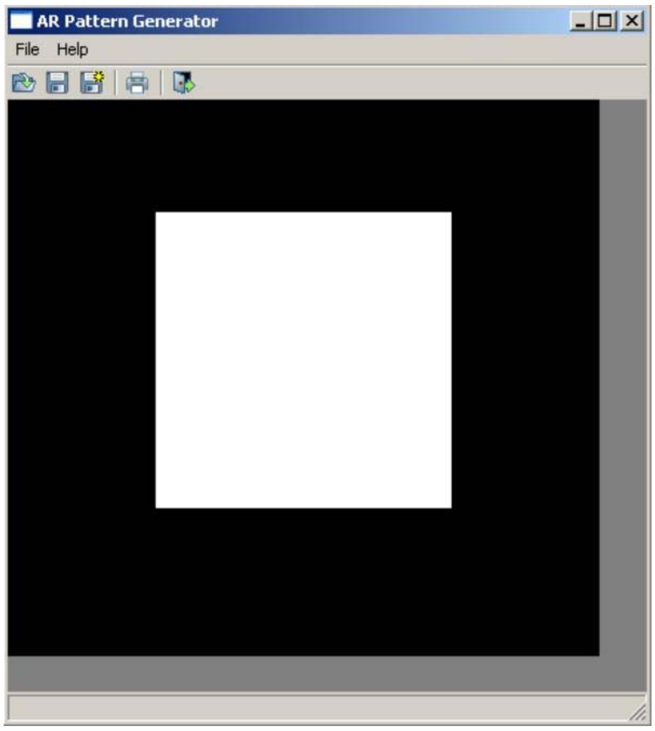


Figure 5.1: AR Pattern Generator Interface

The user can insert a colorful or non color image inside the white place, and the program can scale the input image to 16 × 16 pixels. The format of the insert image can be *BMP*, *PNG* or *GIF*. There are two save icons for saving the two types of pattern files. This generator also can print preview and print out the marker.

5.2 Pattern Files

Pattern files are files that contain data that represents the image in the center of a marker. Although one or more pattern files can be loaded at the same time, the program still knows what markers we are looking for in the video stream. Pattern files also can be tracked from other squares objects in the scene, and enable to distinguish one marker from another.

5.2.1 ARToolkit Marker Pattern File

The format of the pattern file for the ARToolKit marker is the *.patt* file, which is a low resolution data file used by the tracking library to identify the marker. It uses red, green and blue which are 8 bit unsigned integers in the range of [0,255] to present the inserted image values.

5.2.2 StbES Marker Pattern File

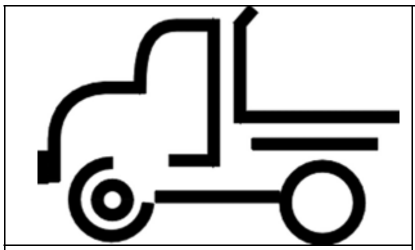
The output pattern file for the StbES marker is a *.pgm* file, which is the lowest common denominator grayscale file format. PGM means Portable Gray Map. It uses the luminance component of the inserted image to convert to greyscale image. The luma value is calculated by using

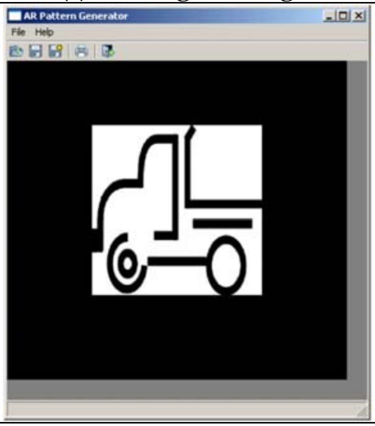
$$\text{Luma} = 30\%\ \text{Red} + 59\%\ \text{Green} + 11\%\ \text{Blue}$$

In the formal, the weightings 30%, 59%, and 11% are chosen to closely match the sensitivity of the eye to red, green, and blue.

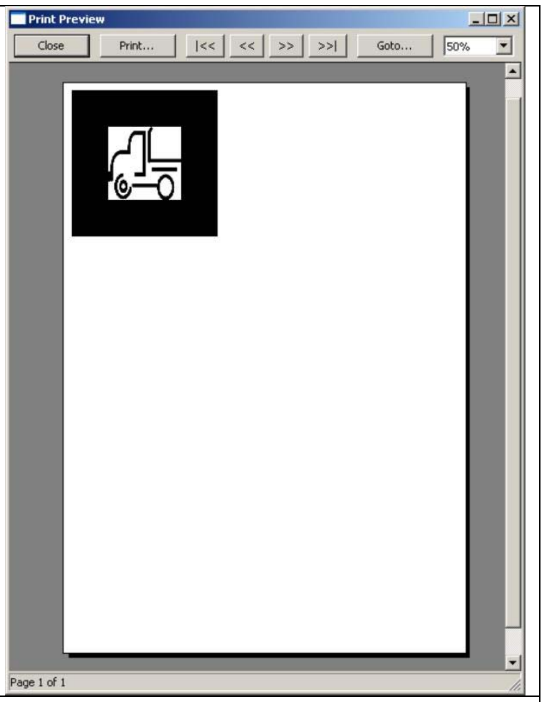
5.3 Marker Preview and Printing

Users also can print out the marker. This is the preview for the created marker. Figure 5.2 shows the example that is created by this tool. (a) is the original image, (b) is the marker view and (c)is the print preview.


(a) The original image



(b) The marker view



(c) The print preview

Figure 5.2: An Example of a Marker

6 Chapter 6 Evaluation and Result

This chapter describes an experiment to test the ComposAR authoring tool when used by the programmers and the non-programmers and its performance compared to different authoring tools.

6.1 Experimental Task and Design

In order to evaluate our AR authoring tool, we compared it to other two applications:

(1) BuildAR which is a PC based AR authoring tool that enables users to create a simple augmented reality scene on the desktop.

(2) Notepad ++ which is an XML editor and not an AR authoring tool called. It is a free source code editor. One of its main features is XML syntax highlighting and folding.

The experiment follows a 3 × 3 × 2 repeated measures design. The two user groups were *programmers* and *non-programmers*. The programmer means someone who can write or debug any computer programs, while the non-programmer means someone who seldom or never writes any codes for programming. There were three tasks, and the three authoring tools used were ComposAR, BuildAR, and an XML editor.

In the experiment participants had to finish a set of three standard authoring tasks using each tool. The task descriptions are as follows:

Task One: Model Loading

- Add the first marker to the AR scene
- Load the first model on the marker
- Add the second marker to the AR scene
- Load the second model on the marker

Task Two: Model Manipulation

Subjects were given a sample AR scene (see Figure 6.1(a)) that already had one model on an AR marker. Using an empty marker, subjects had to load the same model on it and then make it the same size and position of the first model (see Figure 6.1(b)) by using the *translate*, *rotate* and *scale* functions.

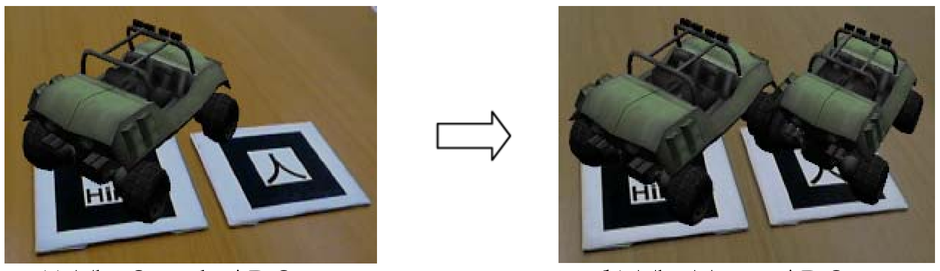

(a) The Sample AR Scene (b) The Target AR Scene

Figure 6.1: A Target Authoring Scene

Task Three: Model Viewing

There were several model files within one folder on the computer. Subjects were told to load and view them one by one using the AR authoring tool.

For the ComposAR authoring tool subjects also completed an additional task where they loaded a “jeep” model, created an XML file of the scene, and then loaded that file on a mobile phone AR viewer to see the AR view of the jeep on the phone.

The participants completed the tasks using all three authoring tools in a counterbalanced order to reduce learning effects. The first tool was randomly chosen when a participant started the experiment.

6.2 Experimental Measures

After the participants finished each task, they filled out a questionnaire about the tool they just used and the tasks they just finished, and how they felt about the tool interface. Then the participants moved on to another authoring tool. After the participants used all the three different tools, they filled out another questionnaire asking them to rank the conditions in several categories.

In addition, the Task Completion Time and Number of Errors made were also captured. The Task Completion Time was measured as the time it took each person to complete the task. Errors were measured as how many errors were present after the user thought they had completed the task correctly. There could be two types of errors: the wrong result, and clicking in the wrong place but eventually figuring out correct place.

The introductory instructions to the participants emphasized the focus on introducing the tool interface and the need to learn how to use the tool. Users were asked to complete the tasks with as fewer errors as possible.

There were 30 participants (12 female and 18 male), aged from 21 to 35 years old. 12 people stated that they had never or seldom written any computer programs before, these were the non-programmers group, the remaining participants were programmers. The experiment lasted about one hour for each user using the three tools, including the introduction and a short concluding discussion. Data analysis was performed by using SPSS version 17 and the main effect was tested using a repeated ANOVA analysis. If a main effect was found, pair-wise post-hoc comparisons were performed by using the Bonferroni adjustment for multiple comparisons.

The questions the participants had to answer after each tool can be grouped into four main categories: performance of the task, ease of completing the task, feeling of control and liking using. Subjects were asked to mark on a scale of 1 to 7 how much they agreed or disagreed with the statements (1 = Strongly Disagree and 7 = Strongly Agree). Appendix B includes the original questionnaires that were handed out to the participants.

6.3 Results

In this section the results from the experiment are presented. Two groups (programmers and non-programmers) performed the same three tasks by using the three different tools; Table 6.1 shows the average amount of time it took to complete task one (Model Loading) using each tool.

Table 6.1: Average Time (Std. Error) to Perform Task One

Tool	**Non-Programmer**	**Programmer**
ComposAR	95.31s (12.17)	102.38s (10.97)
BuildAR	108.54s (12.86)	99.19s (11.59)
XML	319.62s (25.45)	218.94s (22.94)

An ANOVA analysis found a significant difference (F (1, 27) = 4.972, P < 0.05) in the time the non-programmers and programmers took when they performed this task. Mauchly’s test indicates that the assumption of sphericity had been violated; therefore the degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity. Doing this we found a significant difference between the three different tools (F (1.58, 42.669) = 72.958, P < 0.05) in the time it took to perform the task, and also a significant interaction between two groups and the different tools (F (1.58, 42.669) = 6.535, P < 0.05).

A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the XML editor, and the difference between the BuildAR and the XML editor, but no difference between the

ComposAR and the BuildAR. So using the ComposAR and the BuildAR tools was both faster than using the XML editor.

The numbers of times users make a mistake were counted while they performed task one (Model Loading). Table 6.2 shows the average number of user errors in completing task one. The errors were including: users clicked in the wrong place, users forgot to save file, and users loaded the wrong marker files or AR scene files.

Table 6.2: Error Taken (Std. Error) to Perform Task One

Tool	**Non-Programmer**	**Programmer**
ComposAR	0.54 (0.29)	0.88 (0.26)
BuildAR	1.15 (0.28)	0.63 (0.25)
XML	2.77 (0.52)	0.88 (0.47)

An ANOVA analysis found a significant (F (2, 54) = 4.796, P < 0.05) interaction between the two groups and the different tools, a significant difference (F (1, 27) = 5.815, P < 0.05) in the errors non-programmers and programmers made, and a significant difference (F (2, 54) = 5.428, P < 0.05) in errors made with the three different tools.

A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the XML editor, and a difference between the BuildAR and the XML editor, but no difference in the errors made with the ComposAR and the BuildAR tools. Using the ComposAR and the BuildAR, users made fewer errors than using the XML editor, and programmers made fewer errors than non-programmers.

Table 6.3 shows the average time it took to complete task two (Model Manipulation) using each tool.

Table 6.3: Average Time (Std. Error) to Perform Task Two

Tool	Non-Programmer	Programmer
ComposAR	98.92s (27.76)	75.07s (25.70)
BuildAR	236.17s (22.14)	132.43s (20.50)
XML	63.75s (6.93)	48.07s (6.42)

An ANOVA analysis found no significant (F (1.282, 30.771) = 2.821) interaction between the two groups and the different tools. However, a significant difference was found (F (1, 24) = 9.101, $P < 0.05$) between the non-programmers and the programmes in the time to do task two. Mauchly's test indicated that the assumption of sphericity had been violated; therefore the degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity. When this was done, a significant difference was found (F (1.282, 30.771) = 21.381, $P < 0.05$) in the time to do the task between the three different tools.

A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the BuildAR tools, and a difference between the BuildAR and the XML editor, but no difference between ComposAR and the XML editor. It took longer time using the

BuildAR application to complete task two than using ComposAR, and it also took longer time with the BuildAR than the XML editor.

The numbers of times users made a mistake were counted while they performed task two (Model Manipulation). Table 6.4 shows the average number of user errors in completing task two. The errors were including: users clicked in the wrong place, users forgot to save file, users loaded the wrong marker files or AR scene files, and users gained the wrong results.

Table 6.4: Error Taken (Std. Error) to Perform Task Two

Tool	**Non-Programmer**	**Programmer**
ComposAR	1.80 (0.33)	0.54 (0.32)
BuildAR	9.17 (1.33)	4.38 (1.28)
XML	0.50 (0.21)	0.46 (0.20)

An ANOVA analysis found a significant difference in the number of errors between the programmers and the non-programmers (F (1, 23) = 8.656, P < 0.05). Mauchly's test indicated that the assumption of sphericity had been violated; therefore the degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity. Doing this we found a significant difference between the three different tools (F (1.075, 24.721) = 38.740, P < 0.05), and a significant interaction between the two groups using the three different tools (F (1.075, 24.721) = 5.235, P < 0.05).

A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the BuildAR tools, and a difference between the BuildAR and the XML editor, but no difference between the ComposAR and the XML editor. Users using the BuildAR made more errors than using ComposAR, and the BuildAR users also made more errors than the XML editor users in performing task two.

Table 6.5 shows the average performance time for each tool for task three (Model Viewing).

Table 6.5: Average Time (Std. Error) to Perform Task Three

Tool	**Non-Programmer**	**Programmer**
ComposAR	33.67s (8.69)	38.93s (7.77)
BuildAR	56.08s (9.73)	63.13s (8.71)
XML	173.67s (21.60)	150.53s (19.32)

An ANOVA analysis found no significant difference (F (1, 25) = 0.073) in the task time between the non-programmers and the programmers. But, Mauchly's test indicated that the assumption of sphericity had been violated; therefore the degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity. Doing this we found a significant difference (F (1.548, 38.699) = 58.851, $P < 0.05$) between the three authoring tools, and a significant interaction (F (1.548, 38.699) = 0.942, $P < 0.05$) between the two groups of users and the three tools.

A post-hoc analysis with Bonferroni correction showed pair-wise differences between all the three tools. It took people more time to complete the task with the XML editor than with the BuildAR, the

BuildAR user also took more time than with the ComposAR, and XML editor required more time than with the ComposAR. Therefore, the ComposAR application is the fastest tool for completing task three.

The numbers of times users made a mistake were counted while they performed task three (Model Viewing). Table 6.6 shows the average number of user errors in completing task three. The errors were including: users clicked in the wrong place, users forgot to save file, users loaded the wrong marker files or AR scene files, and users gained the wrong results.

Table 6.6: Error Taken (Std. Error) to Perform Task Three

Tool	Non-Programmer	Programmer
ComposAR	0.00 (0.12)	0.20 (0.11)
BuildAR	0.33 (0.38)	0.53 (0.28)
XML	0.83 (0.27)	1.07 (0.25)

An ANOVA analysis found no significant difference (F (1, 25) = 1.151) between the non-programmers and the programmers in the number of errors. But, Mauchly’s test indicated that the assumption of sphericity had been violated; therefore the degrees of freedom were corrected using Greenhouse-Geisser estimates of sphericity. Doing this we found a significant difference (F (1.576, 39.391) = 6.785, $P < 0.05$) between the three different tools, and no significant interaction (F (1.576, 39.391) = 0.003) between the two groups of users and the three tools.

A post-hoc analysis with Bonferroni correction showed a pair-wise difference in the errors made with the ComposAR and the XML editor, but no difference between the BuildAR and the XML editor, and no difference between the BuildAR and the ComposAR. Users of ComposAR produced fewer errors than users of the XML editor when doing task three.

To evaluate the users' subjective feelings about the user interface, we asked questions in a number of different categories. In the task performance category we asked the following nine questions:

- Q1: I can easily add a marker
- Q2: I can easily load a 3D model
- Q3: I can easily translate the 3D model
- Q4: I can easily rotate the 3D model
- Q5: I can easily scale the 3D model
- Q6: I can easily browse the entire set of 3D models by using this tool
- Q7: I can easily control this tool
- Q8: I can easily tell what was going on
- Q9: I feel very comfortable while using this tool

Figure 6.2 and Table 6.7 show the subjective survey scores for the questions one to nine.

Table 6.7: Average Result (Std. Error) for Performance of the Task

	Non Programmer			Programmer		
	ComposAR	**BuildAR**	**XML**	**ComposAR**	**BuildAR**	**XML**
Q1	5.69 (0.38)	5.92 (0.38)	4.08 (0.57)	6.53 (0.33)	6.47 (0.33)	5.12 (0.50)
Q2	5.62 (0.42)	6.00 (0.42)	4.23 (0.56)	6.18 (0.37)	6.18 (0.37)	5.18 (0.49)
Q3	5.46 (0.42)	5.69 (0.50)	4.62 (0.51)	6.29 (0.36)	5.00 (0.44)	5.29 (0.45)
Q4	5.15 (0.37)	4.00 (0.59)	4.85 (0.53)	6.29 (0.32)	4.88 (0.52)	5.06 (0.46)
Q5	5.39 (0.35)	4.92 (0.61)	5.00 (0.55)	6.53 (0.30)	4.94 (0.53)	5.24 (0.48)
Q6	5.85 (0.31)	5.01 (0.54)	4.39 (0.56)	6.29 (0.27)	5.71 (0.47)	4.18 (0.49)
Q7	5.77 (0.40)	5.77 (0.35)	3.92 (0.51)	5.77 (0.35)	5.41 (0.36)	4.88 (0.45)
Q8	5.15 (0.52)	5.01 (0.38)	3.54 (0.45)	5.77 (0.45)	5.77 (0.33)	5.71 (0.39)
Q9	5.54 (0.37)	5.00 (0.46)	4.15 (0.52)	5.88 (0.32)	5.35 (0.40)	4.82 (0.45)

Performing an ANOVA analysis on Q3 did not find any significant difference.

On Q1: *I can easily add a marker*, an ANOVA analysis found a significant difference (F (2, 56) = 9.679, $P < 0.05$) between the three tools, a significant difference (F (1, 28) = 4.882, $P < 0.05$) between the programmers and the non-programmers. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the XML editor, and a difference between BuildAR and the XML editor, and no difference between ComposAR and the BuildAR applications. Participants can easily add a marker using ComposAR and the BuildAR tools than using the XML editor.

On Q2: *I can easily load a 3D model*, an ANOVA analysis found a significant difference (F (1.666, 46.643) = 7.770, P < 0.05) between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the BuildAR and the XML editor, and a difference between ComposAR and the XML editor, and no difference between ComposAR and the BuildAR applications. Participants can easily load a 3D model using the ComposAR and the BuildAR tools than using the XML editor.

On Q4: *I can easily rotate the 3D model*, an ANOVA analysis found a significant difference (F (2, 56) = 4.215, P < 0.05) between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the BuildAR, there were no difference between ComposAR and the XML editor, and no difference between the XML editor and the BuildAR applications. Participants can easily rotate the 3D model using the ComposAR than using the BuildAR.

On Q5: *I can easily scale the 3D model*, an ANOVA analysis found a significant difference (F (1.923, 53.846) = 3.322, P < 0.05) between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the BuildAR, there were no difference between ComposAR and the XML editor, and no difference between the XML editor and the BuildAR applications. Participants can easily scale a 3D model using the ComposAR than using the BuildAR.

On Q6: *I can easily browse the entire set of 3D models by using this tool*, an ANOVA analysis found a significant difference (F (1.923, 54.687) = 8.892, $P < 0.05$) between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the BuildAR and the XML editor, and a difference between ComposAR and the XML editor, and no difference between ComposAR and the BuildAR applications. Participants can easily browse the entire set of 3D models by using the ComposAR and the BuildAR tools than using the XML editor.

On Q7: *I can easily control this tool*, an ANOVA analysis found a significant difference (F (1.952, 54.655) = 5.659, $P < 0.05$) between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the XML editor, and no difference between BuildAR and the XML editor, and no difference between ComposAR and the BuildAR applications. Participants can easily control the ComposAR than control the XML editor.

On Q8: *I can easily tell what was going on*, an ANOVA analysis found a significant difference (F (1, 28) = 9.554, $P < 0.05$) between the programmers and the non-programmers.

On Q9: *I feel very comfortable while using this tool*, an ANOVA analysis found a significant difference (F (2, 56) = 4.061, $P < 0.05$) between the three tools. between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the ComposAR and the XML editor, and no difference between BuildAR and the XML editor, and no difference between ComposAR and the BuildAR applications. Participants felt more comfortable using the ComposAR than using the XML editor.

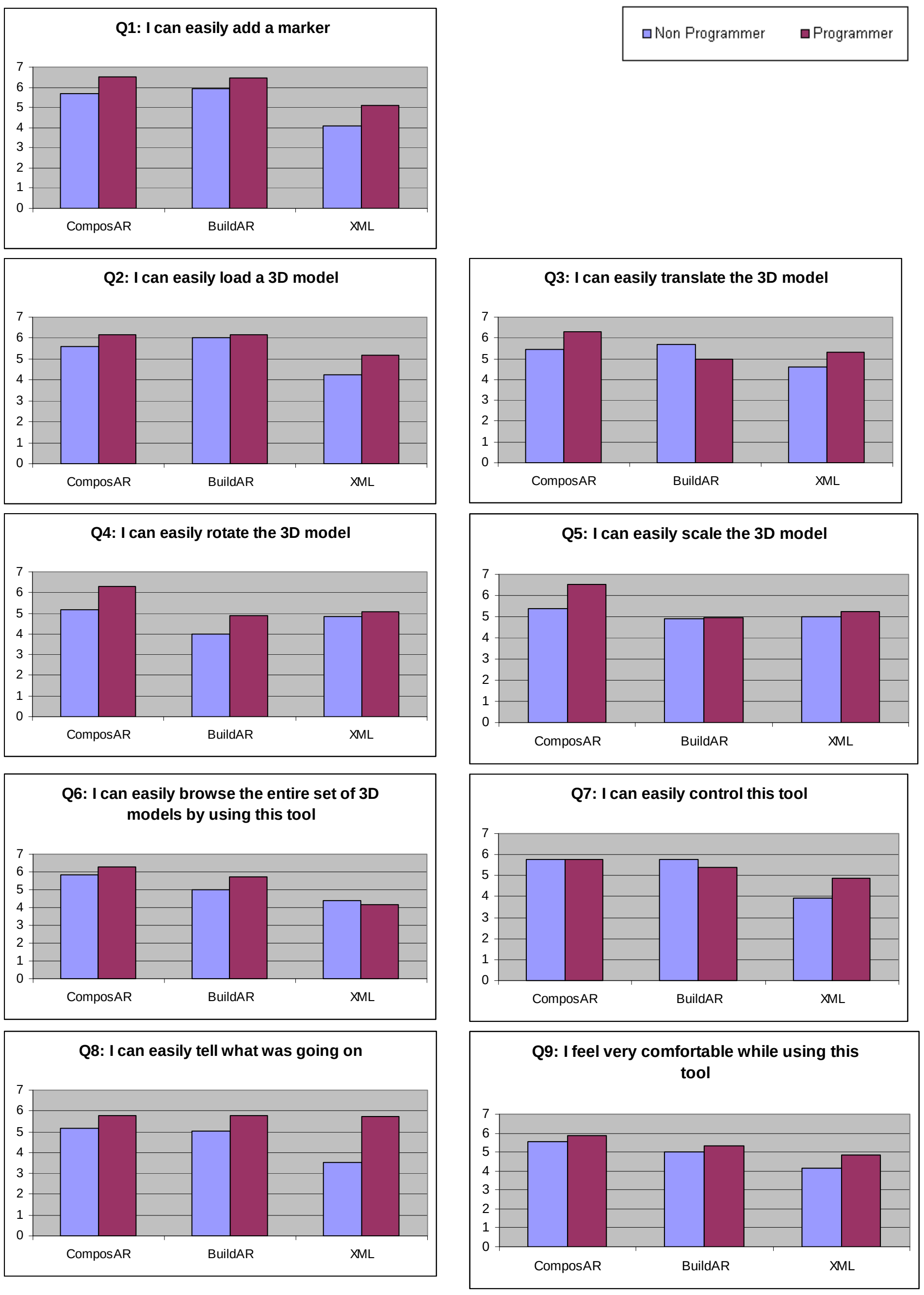


Figure 6.2: Subjective Survey Scores for Questions 1 - 9.

A second set of questions related to the ease of the task:

- Q10: I always understand clearly what I was supposed to do
- Q11: I had sometimes problems with the user interface
- Q12: The tool was sometimes confusing
- Q13: The tasks were easy to solve

Table 6.8: Average Result (Std. Error) for Ease of the Task

	Non Programmer			**Programmer**		
	ComposAR	**BuildAR**	**XML**	**ComposAR**	**BuildAR**	**XML**
Q10	4.77 (0.43)	5.46 (0.45)	4.54 (0.45)	5.77 (0.38)	5.47 (0.39)	5.59 (0.42)
Q11	4.23 (0.53)	4.62 (0.53)	3.85 (0.55)	3.65 (0.46)	4.53 (0.47)	2.88 (0.48)
Q12	3.62 (0.58)	4.62 (0.47)	3.86 (0.45)	3.24 (0.51)	4.06 (0.41)	3.12 (0.39)
Q13	5.92 (0.34)	5.39 (0.43)	4.23 (0.41)	6.18 (0.30)	5.71 (0.38)	5.71 (0.36)

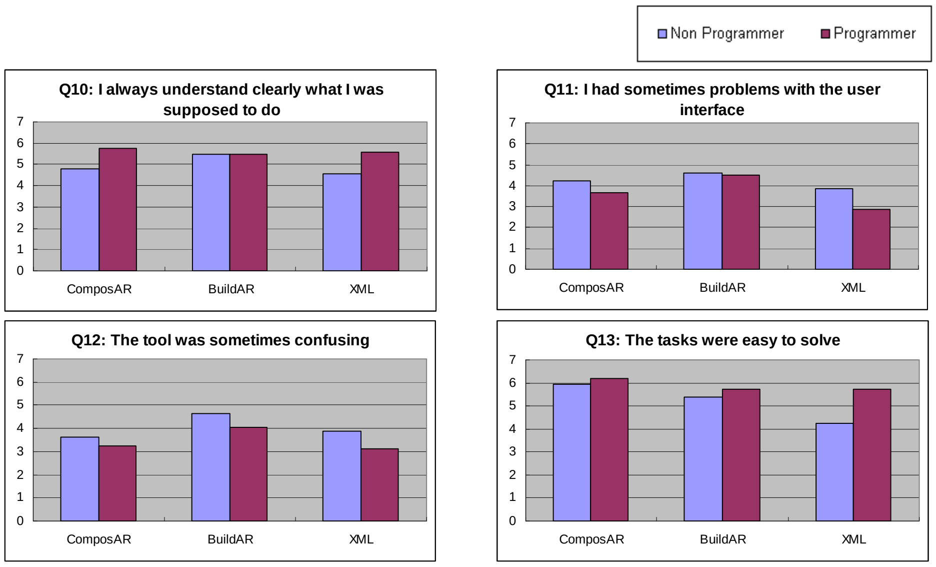


Figure 6.3: Subjective Survey Scores for Questions 10 - 13

Table 6.8 and Figure 6.3 show the average user scores from these questions. Performing an ANOVA analysis on Q10 and Q12 did not find any significant difference.

On Q11: *I had sometimes problems with the user interface*, an ANOVA analysis found a significant difference (F (2, 56) = 4.061, P < 0.05) between the three tools. A post-hoc analysis with Bonferroni correction showed a pair-wise difference between the BuildAR and the XML editor, but a difference between ComposAR and the XML editor, and no difference between ComposAR and the BuildAR applications. Participants had sometimes more problems with the user interface using the XML editor than using the ComposAR and the BuildAR tools.

For Q13: *The tasks were easy to solve*, an ANOVA analysis found a significant difference (F (1, 28) = 4.436, P < 0.05) between the programmers and the non-programmers. There is also a significant difference (F (2, 56) = 4.614, P < 0.05) between the three tools. Post-hoc analysis with Bonferroni correction showed a pair wise difference between the ComposAR and the XML editor, but no difference between the BuildAR and the XML editor, and no difference between the ComposAR and the BuildAR applications. Participants felt that the task was easier to solve using the ComposAR tool than using the XML editor.

To measure how people felt in control was the questions we asked:

- Q14: The user interface made me feel in control
- Q15: The user interface was easy to use

Table 6.9: Average Result (Std. Error) for Felt in Control

	Non Programmer			Programmer		
	ComposAR	**BuildAR**	**XML**	**ComposAR**	**BuildAR**	**XML**
Q14	5.15 (0.46)	5.31 (0.44)	3.92 (0.48)	5.29 (0.40)	5.41 (0.39)	5.77 (0.42)
Q15	5.08 (0.41)	5.08 (0.45)	3.77 (0.50)	5.53 (0.36)	5.35 (0.40)	4.35 (0.44)

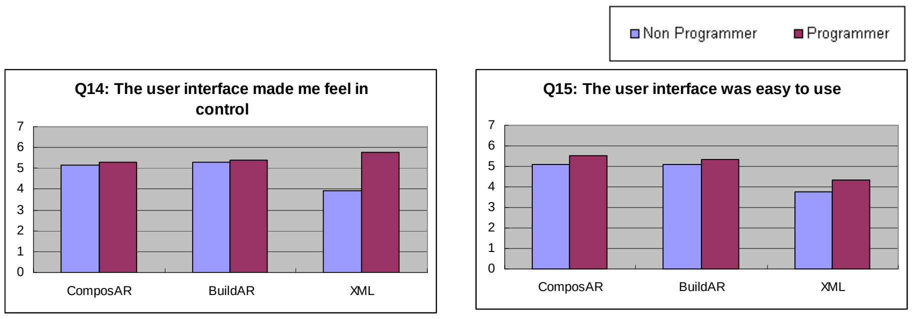


Figure 6.4: Subjective Survey Scores for Questions 14 and 15

Table 6.9 and Figure 6.3 show the average result. Performing an ANOVA analysis on Q14 did not find any significant difference.

However, an ANOVA analysis found significant difference for Q15: *The user interface was easy to use* (F (2, 56) = 6.453, P < 0.05). A post-hoc comparison showed that users felt that using ComposAR and

BuildAR was easier than using the XML editor, and there was no difference between the ComposAR and the BuildAR for ease.

The last group of questions we asked were about how much people liked using the tool:

- Q16: I enjoyed using the tool
- Q17: Using the tool was a great experience

Table 6.10: Average Result (Std. Error) for People Liked Using the Tool

	Non Programmer			Programmer		
	ComposAR	**BuildAR**	**XML**	**ComposAR**	**BuildAR**	**XML**
Q16	6.00 (0.34)	5.01 (0.40)	3.85 (0.54)	5.71 (0.30)	5.12 (0.35)	4.41 (0.48)
Q17	5.62 (0.38)	4.77 (0.53)	4.39 (0.54)	5.47 (0.38)	4.88 (0.46)	4.18 (0.48)

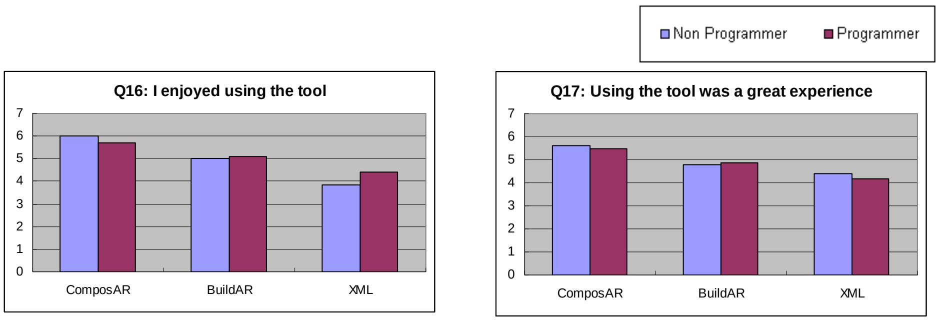


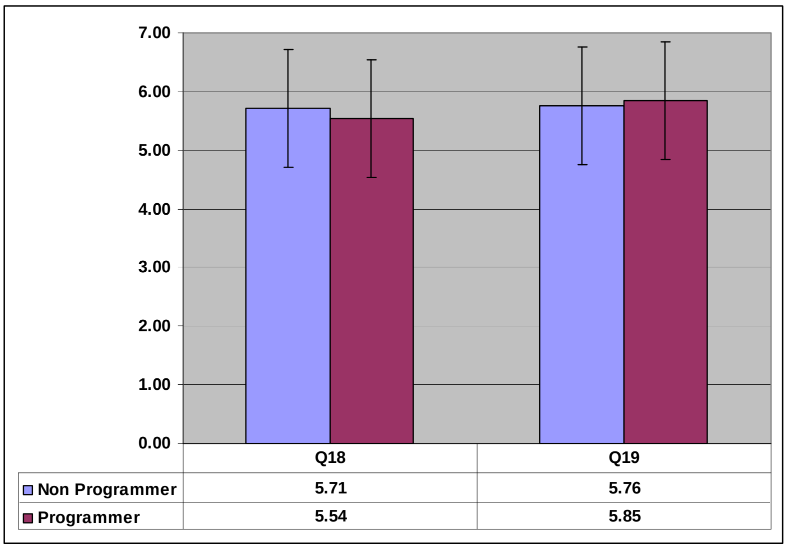
Figure 6.5: Subjective Survey Scores for Questions 16 and 17

Table 6.10 and Figure 6.4 show the average results. An ANOVA analysis found a significant difference in responses to Q16: *I enjoyed using the tool* (F (2, 56) = 13.294, P< 0.05). A post-hoc analysis found that the participants enjoyed using the ComposAR and the BuildAR more than using the XML editor, but there was no difference as for how much they liked ComposAR and BuildAR.

An ANOVA analysis was found a significant difference in response to Q17: *Using the tool was a great experience* F (2, 56) = 11.813, P<0.05. A post-hoc analysis showed that participants preferred using the ComposAR than the XML editor, but there was no difference between the BuildAR and the XML editor, and no difference between ComposAR and BuildAR.

For the ComposAR tool, two additional questions were asked:

- Q18: This tool would fit well into a mobile phone AR application
- Q19: I would like to use this tool for mobile phone AR applications



	Q18	Q19
🟦 Non Programmer	5.71	5.76
🟥 Programmer	5.54	5.85

Figure 6.6: Subjective Survey Scores for the Questions 18 and 19

Figure 6.6 shows the average result for the two questions. Performing an ANOVA analysis on Q18 and Q19 did not find any significant difference between the programmers and the non-programmers. They had the same opinions for these two questions.

In addition, subjects were also asked to rate each of the conditions on a scale of one to seven according to a set of criteria shown in Table 6.11. For each criteria the score 1 = lowest, 7 = highest. Table 6.11 shows the average results.

Users felt that ComposAR is the easiest to use (Friedman test $\chi_r^2 = 18.712$, $df = 2, N = 30, p < 0.001$); ComposAR is the most interesting. (Friedman test $\chi_r^2 = 22.962$, $df = 2, N = 30, p < 0.001$); ComposAR is not boring (Friedman test $\chi_r^2 = 22.169$, $df = 2, N = 30, p < 0.001$); The XML editor is the most precise (Friedman test $\chi_r^2 = 11.553$, $df = 2, N = 30, p < 0.05$); ComposAR is efficient (Friedman test $\chi_r^2 = 14.857$, $df = 2, N = 30, p < 0.05$); ComposAR is funny (Friedman test $\chi_r^2 = 27.244$, $df = 2, N = 30, p < 0.001$); The XML editor is very skilled (Friedman test $\chi_r^2 = 13.473$, $df = 2, N = 30, p < 0.05$); ComposAR is more satisfying. (Friedman test $\chi_r^2 = 20.265$, $df = 2, N = 30, p < 0.001$); ComposAR is more engaging. (Friedman test $\chi_r^2 = 23.146$,

$df = 2, N = 30, p < 0.001$); Overall, ComposAR is rated better than the others. (Friedman test $\chi_r^2 = 9.529$, $df = 2, N = 30, p < 0.05$).

Table 6.11: Results (Std. Error) of the User Experience

	Measure	**ComposAR**				**BuildAR**				**XML**			
		Non Programmer		**Programmer**		**Non Programmer**		**Programmer**		**Non Programmer**		**Programmer**	
		Mean	Std. Error	Mean	Std. Error	Mean	Std. Error	Mean	Std. Error	Mean	Std. Error	Mean	Std. Error
1	**Easy**	6.00	0.267	6.00	0.23	4.15	0.48	5.35	0.42	2.77	0.54	4.29	0.48
2	**Interesting**	5.62	0.35	5.82	0.31	4.46	0.47	5.29	0.41	3.08	0.57	3.41	0.50
3	**Boring**	2.69	0.48	2.35	0.42	3.15	0.49	3.00	0.42	4.46	0.55	4.29	0.48
4	**Precise**	5.85	0.31	5.35	0.27	4.00	0.47	5.00	0.41	5.39	0.34	6.00	0.29
5	**Obvious**	5.00	0.48	5.53	0.42	4.08	0.55	4.94	0.48	3.15	0.52	4.94	0.48
6	**Efficient**	5.46	0.42	5.82	0.37	4.15	0.43	5.18	0.37	2.85	0.50	4.18	0.44
7	**Funny**	5.39	0.38	5.53	0.33	5.00	0.42	5.06	0.37	3.15	0.55	3.35	0.48
8	**Skilled**	3.54	0.53	3.65	0.47	3.15	0.40	3.29	0.35	4.54	0.48	4.88	0.42
9	**Satisfying**	5.54	0.40	5.82	0.35	4.62	0.39	5.24	0.34	3.23	0.51	4.59	0.45
10	**Engaging**	5.39	0.31	6.06	0.27	4.69	0.34	5.24	0.30	3.23	0.51	4.47	0.45
11	**Overall**	5.39	0.40	5.65	0.35	4.54	0.45	5.06	0.40	4.54	0.45	5.06	0.40

6.4 Interviews

The participants were interviewed after they finished the experimental tasks. After using the XML editor, most non-programmers felt it was hard to finish the tasks, although there were some comments on the editor. Two participants only finished task one, and gave up the rest of two tasks, because they did not understand and felt very boring. But some programmers found it easy to play with this XML editor. The same code was quickly copied and pasted to manipulate the model.

Most participants pointed out that it was a bit difficult to figure out how to save the *translation*, *rotation* and *scale* factors by using the BuildAR tool to complete task two. Three participants gave up finishing task two because the controls in BuildAR are not obvious. They felt confused and did not know how to confirm the entered data.

Most participants felt the ComposAR tool was much easier to use than the other two tools. However, some of them complained that the icons needed to be clarified and tips for the buttons should appear on the screen. Otherwise, they would complete the tasks faster and with fewer errors.

Using the ComposAR tool, some participants said that it was so interesting that they were able to view the 3D object using their mobile phone.

6.5 Discussion

There were significantly different user subjective results between the three different tools, and the difference between the programmers and the non-programmers using the different tools.

One of the most obvious differences was seen in the model viewing task. When using the XML editor, the users opened the file folder, browsed all the files and remembered one of the file names, then typed into the editor and saved it, and finally, they ran the application to view the model. The BuildAR users entered the files folder and opened the model file to view the model. What the ComposAR users did was only click the button, which only took several seconds. Thus it was the fastest tool for model viewing.

Another key difference was between the programmers and the non-programmers using the three different tools to do the model loading. The main difference occurred between the programmers and the non-programmers using XML editor, while there was no difference between them using ComposAR and BuildAR. Most programmers understood the pre-existing code, so they could easily and quickly load the model. In contrast, most non-programmers could not understand what was written on the editor, and did not know what the code meant, and did not even know how to control them. So it took the non-programmers several minutes to figure out what was going on. Both the ComposAR users and the BuildAR users loaded the model in a similarly way, which was to open the file folder and choose one model.

Users rated the ComposAR as the tool they most liked using. This was because of how easy it was to manipulate the model, and support for direct viewing the model with any changes. In contrast, after modifying the code by using the XML editor, users must restart the application to view the model. Although the BuildAR tool could view changes of the model directly, users felt it was hard to manipulate the model by using the *translation*, *rotation*, and *scale* values.

The programmers made more mistakes and required more time in completing task one, because some programmers presumably more experiments, while the non-programmers felt more curious though the test.

In the performance of the task category, most non-programmers felt it is very easy to add a marker, loaded a 3D model even felt easily control the ComposAR tool than control the XML editor.

7 Chapter 7 Performance Measurements

To test the performance of mobile AR applications made by the authoring tool, benchmarks were performed on a mobile phone device (HP iPAQ 612c). These tests compare tracking performance with different numbers of visible markers and different models. All tests were done with the mobile AR viewer application developed in this Masters thesis.

The tests were run on an HP mobile phone device which is currently available on the market, running Windows CE. Additionally the benchmarks were run on a PC as a comparison of the processing power on the mobile phone to a typical PC-based set up.

The specific devices used were:

• **HP iPAQ 612c,** an enterprise-level PDA phone with a Marvell 520MHz processor with 128MB of RAM.

• **Intel 2.40 GHz Core Duo,** a standard PC-based setup.

Three different scenarios were evaluated: using single marker tracking, separate multi-marker tracking, and combined multi-marker tracking.

An initial experiment was run to check on whether or not the marker size should be a factor. Table 7.1 shows the tracking speed result on the different size of the markers. Contrary to expectation, the size of the marker does not influence the tracking speed. The reason for this is that the edge following step generally adds only very little to the overall calculation time

Table 7.1: Tracking Speed on Different Size of the Markers

Size (cm)	20 × 20	15 × 15	10 ×10	8 × 8	4 × 4	2 ×2	1 ×1
FPS	26.7	26.8	26.5	26.7	26.3	26.3	26.5

The systems were then run with 1, 4, 6 and 10 markers being tracked simultaneously, on both the phone and desktop system. Separate/Combined. The separate marker means every single marker; whereas the combined marker means two or more marker appear on the same paper. The speeds (measured in frames per second, fps) are shown in Table 7.2. A higher fps value is better; rates below 10 fps will look jerky to the viewer, and commercial video uses rates of around 24 fps and higher.

Table 7.2: Benchmarks Performed with Single and Multi Marker, Speed Shown in Frames Per Second

Device	**Single Marker**	**Multi Marker (4 markers)**		**Multi Marker (6 markers)**		**Multi Marker (10 markers)**	
		Separate	Combined	Separate	Combined	Separate	Combined
HP iPAQ 612c	18.3	16.3	15.9	15.7	13.4	14.4	10.7
Desktop	26.9	25.7	24.8	21.8	20.9	13.1	12.2

The single marker tracking represents the fastest fame rate. Tracking a multi-marker set with N visible markers is slower than tracking N independent markers. However, tracking a multi-marker set with six visible markers still performs faster on this mobile phone device, while tracking a multi-marker set with ten visible markers will cause a bottleneck.

The portable device is not significantly slower than the desktop machine, and both produce rendering speeds that are acceptable for video viewing.

Benchmarks

In addition to tracking, the AR system needs to render images in the view, and the rendering speed will depend on the image being displayed. To compare the mobile phone and the desktop applications, several tests images were used as benchmarks:

• **Cube:** This test renders a cube on top of the marker (see Figure 7.1(a)).

• **Jeep:** This test renders a detailed and textured model of a jeep[13] on top of a marker (see Figure 7.1(b)). The 3D model consists of 2032 polygons with different pixel texture: 512 × 512, 256 × 256, and 128 × 128.

• **Truck:** This test renders a highly detailed and lit model of a truck[14] on top of a marker (see Figure 7.1(c)). The model consists of one mesh and 12 materials. In total, the model contains 16387 vertices and 31716 polygons.

[13] Jeep model was free downloaded from website: http://www.psionic3d.co.uk/.
[14] Truck model was free downloaded from website: http://www.turbosquid.com/.

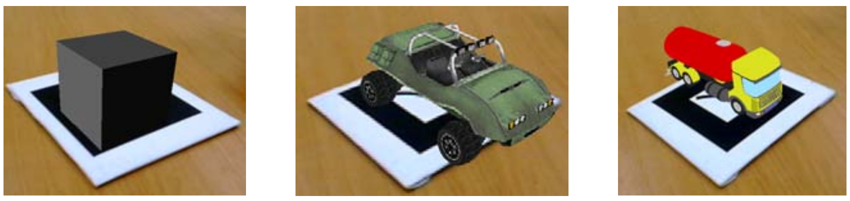

(a) Cube (b) Jeep (c) Truck

Figure 7.1: Test Models Rendered for Benchmarking

Five tests were performed on the mobile phone and desktop. Table 7.3 shows the results of the tests. The table lists only frames per second (fps) values. The higher values are better.

Table 7.3: Result for the Test Model (Frames Per Second)

Device	Cube	Jeep (128×128)	Jeep (256×256)	Jeep (512×512)	Truck
HP iPAQ 612c	20.6	14.5	12.0	10.1	5.4
Desktop	31.2	29.4	28.5	27.6	19.5

The performance on the desktop machine is higher than on the mobile phone, although all results are acceptable for live viewing. We note the following effects:

- A smaller image size (number of pixels) produces better performance on both platforms
- A small number of polygons in the 3D model produce better performance
- Speed deteriorating by no room on display on this phone at once

However, if we concern the affects list above, we can create mobile AR applications by the authoring tool.

8 Chapter 8 Conclusion and Future Work

In this research, we have focused on the design, development and evaluation of an authoring tool for building mobile phone AR applications. We did some background researches that found there is no such authoring tool for mobile AR applications. The AR authoring system has been described in detail in the thesis. Finally, a user evaluation was conducted between the programmers and non-programmers using different authoring tools, and some tests performance measurements were taken on a mobile phone.

The experiment evaluations showed that the participants prefer the ComposAR tool to the other two tools, and there was no significant difference between the programmers and the non programmers using this tool.

The performance measurements evaluate the speed performance of the system on a mobile phone and desktop PC. Even quite complex images (with less visiable markers and detailed 3D models) performed at video rates that are acceptable to users. Not surprisingly, better performance on the mobile phone was obtained with smaller images size, a small number of polygons in the 3D model and fewer markers produce.

In the future, we would like to adapt the authoring tool for different operating systems and mobile platforms. It can be used widely. Another possible direction for further research is to improve the interface design, such as:

- Supporting a touch screen with the labeled buttons; and
- Providing an easy way to drag the marker and content files rather than go through the folders or subfolders; and
- Adding animation and sound.

References

ALEXANDRA MAKRI, JENS WEIDENHAUSEN, PETER ESCHLER, DIDIER STRICKER, OLIVER MACHUI, CARLOS FERNANDES, SERGIO MARIA, GERRIT VOSS & IOANNIDIS, N. (2005) ULTRA Light Augmented Reality Mobile System. *Proceedings of the ISMAR.*

AMIRE (2002) AMIRE: authoring mixed reality Main Page: http://amire.sourceforge.net/.

ARTOOLKIT (2001) ARToolKit website: http://www.hitl.washington.edu/artoolkit/.

AZUMA, R., BAILLOT, Y., BEHRINGER, R., FEINER, S., JULIER, S. & MACINTYRE, B. (2001) Recent advances in augmented reality. *Computer Graphics and Applications, IEEE,* 21**,** 34-47.

AZUMA, R. T. (1997) A Survey of Augmented Reality. *In Presence: Teleoperators and Virtual Environments,* 6**,** 355-385.

BRANNAN, S. (2001) How the First-Down Line Works. HowStuffWorks.com.

CAUDELL, T. P. & MIZELL, D. W. (1992) Augmented reality: an application of heads-up display technology to manual manufacturing processes. *System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on.*

CAVALLARO, R. (1997) The FoxTrax Hockey Puck Tracking System. *IEEE Comput. Graph. Appl.,* 17, 6-12.

DIETER, S. & DANIEL, W. (2008) Mobile Phones as a Platform for Augmented Reality. *Proceedings of the IEEE VR 2008 Workshop on Software Engineering and Architectures for Realtime Interactive Systems.* Reno, NV, USA, Shaker Publishing.

EDGELIB (2008) Edgelib website:http://www.edgelib.com/.

FIALA, M. (2005) ARTag, a fiducial marker system using digital techniques. *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on.*

FISHER, S. (2001) Environmental Media: Accessing Virtual Representations of Real-Time Sensor Data and Site-specific Annotations Embedded in Physical Environments. *Proc. WSMM '01 (Seventh Int. Conf. on Virtual Systems and Multimedia).* Berkeley, CA.

FISHER, S. (2002) An Authoring Toolkit for Mixed Reality Experiences. *Proc. IWEC '02 (Int. Workshop on Entertainment Computing).* Makuhari, Japan.

FURNESS, T. A. (1986) The Super Cockpit and Human Factors Challenges. *Proceedings of Human Factors Society 30th Annual Meeting.*

GRASSET, R., LOOSER, J. & BILLINGHURST, M. (2005) OSGARToolKit: tangible + transitional 3D collaborative mixed reality framework. *Proceedings of the 2005 international conference on Augmented tele-existence.* Christchurch, New Zealand, ACM.

GRIMM, P. (2006) AR-Blender

GRIMM, P., HALLER, M., PAELKE, V., REINHOLD, S., REIMANN, C. & ZAUNER, R. (2002) AMIRE - authoring mixed reality. *Augmented Reality Toolkit, The First IEEE International Workshop.*

GUVEN, S. & FEINER, S. (2003) Authoring 3D hypermedia for wearable augmented and virtual reality. IN FEINER, S. (Ed.) *Wearable Computers, 2003. Proceedings. Seventh IEEE International Symposium on.*

HENRYSSON, A., BILLINGHURST, M. & OLLILA, M. (2005) Face to face collaborative AR on mobile phones. *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on.*

KATO, H. & BILLINGHURST, M. (1999) Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System. *Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality.* IEEE Computer Society.

KOYAMA, T. (2008) FLARTookit http://saqoosha.net/en/flartoolkit/start-up-guide/.

LOOSER, J. (2008) BuildAR http://www.hitlabnz.org/wiki/BuildAR.

LOOSER, J., GRASSET, R., SEICHTER, H. & BILLINGHURST, M. (2006) OSGART - A Pragmatic Approach to MR. *Proceedings of International Symposium of Mixed and Augmented Reality.*

MACINTYRE, B., GANDY, M., DOW, S. & BOLTER, J. D. (2005) DART: a toolkit for rapid design exploration of augmented reality experiences. *ACM SIGGRAPH 2005 Papers.* Los Angeles, California, ACM.

MILGRAM, P., KISHINO, F., TAKEMURA, H. & UTSUMI, A. (1994) Augmented Reality: A Class of Displays on the Reality-Virtuality Continuum. *Proceedings of the SPIE Telemanipulator and Telepresence Technologies,* 2351**,** 282-292.

OSGART (2006) OSGART website: http://www.artoolworks.com/community/osgart/.

PASMAN, W. & WOODWARD, C. (2003) Implementation of an augmented reality system on a PDA. *Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality.*

SCHMALSTIEG, D., FUHRMANN, A., HESINA, G., SZALAVÁRI, Z., ENCARNAÇÃO, L. M., GERVAUTZ, M. & PURGATHOFER, W. (2002) The Studierstube Augmented Reality Project. *Presence: Teleoperators & Virtual Environments,* 11**,** 33-54.

SEICHTER, H., LOOSER, J. & BILLINGHURST, M. (2008) ComposAR: An intuitive tool for authoring AR applications. *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality.* IEEE Computer Society.

SPORTVISION (2006) Sportvision website http://www.sportvision.com/.

STUDIERSTUBEES (2008) StudierstubeES website http://studierstube.icg.tu-graz.ac.at/handheld_ar/stbes.php.

SUTHERLAND, I. E. (1965) The Ultimate Display. *Proceedings of IFIPS Congress* New York.

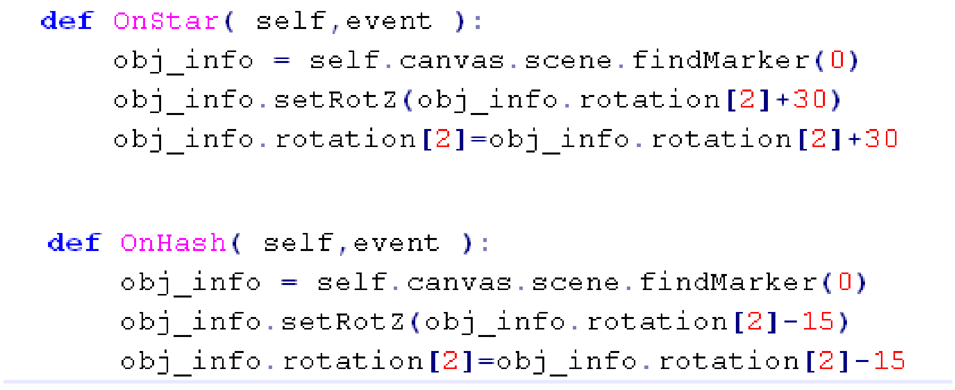
SUTHERLAND, I. E. (1968) A head-mounted three dimensional display. *Proceedings of the December 9-11, 1968, fall joint computer conference, part I.* San Francisco, California, ACM.

WALSH, M. (2009) Augmented Reality To Ramp on Mobile http://www.mediapost.com/publications/?fa=Articles.showArticle&art_aid=118173.

WANG, Y., LANGLOTZ, T., BELL, T. & BILLINGHURST, M. (2009) An Authoring Tool for Mobile Phone AR Environments. *Proc NZCSRSC 09, New Zealand Computer Science Research Student Conference.* Auckland, New Zealand.

YEOH, H. (2005) *Using TinyXML with Visual C++*.

Appendix A: Python Script Examples



```
def OnStar( self,event ):
    obj_info = self.canvas.scene.findMarker(0)
    obj_info.setRotZ(obj_info.rotation[2]+30)
    obj_info.rotation[2]=obj_info.rotation[2]+30


def OnHash( self,event ):
    obj_info = self.canvas.scene.findMarker(0)
    obj_info.setRotZ(obj_info.rotation[2]-15)
    obj_info.rotation[2]=obj_info.rotation[2]-15
```

Appendix B: Experiment Questionnaire

Survey Questions

Date:

User ID: __________ Gender: __________Age: __________

How familiar are you with programming (circle one)?

1	2	3	4	5	6	7
Not very Familiar						Very Familiar

How familiar are you with Augmented Reality (circle one)?

1	2	3	4	5	6	7
Not very Familiar						Very Familiar

You are going to do some tasks by using the three different tools. In these tasks, you will see virtual items shown on the desktop PC screen and the mobile phone screen overlaid on markers in the real world.

You will finish several tasks by using each tool. After each section, you will have to answer some questions about the control and how you felt about the tool application.

ComposAR

Thank you for trying AR authoring tool! Please answer a few questions about your experience. See blow for how your answers will be used.

On a scale of 1 to 7, please circle the number according to how much you feel strongly agree or strongly disagree with the following statements:

I can easily add a marker

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily load a 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily translate the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily rotate the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily scale the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily browse the entire set of 3D models by using this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily control this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily tell what was going on

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I feel very comfortable while using this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

Please rate your agreement with the following statements about your experience using this authoring tool.

I always understand clearly what I was supposed to do
Disagree 1 2 3 4 5 6 7 Agree
I had sometimes problems with the user interface
Disagree 1 2 3 4 5 6 7 Agree
The tool was sometimes confusing
Disagree 1 2 3 4 5 6 7 Agree
The tasks were easy to solve
Disagree 1 2 3 4 5 6 7 Agree

User Interface

The user interface made me feel in control
Disagree 1 2 3 4 5 6 7 Agree
The user interface was easy to use
Disagree 1 2 3 4 5 6 7 Agree

Overall

I enjoyed using the tool
Disagree 1 2 3 4 5 6 7 Agree
Using the tool was a great experience
Disagree 1 2 3 4 5 6 7 Agree
This tool would fit well into a mobile phone AR application
Disagree 1 2 3 4 5 6 7 Agree
I would like to use this tool for mobile phone AR application
Disagree 1 2 3 4 5 6 7 Agree

How much did you like using this authoring tool?

Not very much 1 2 3 4 5 6 7 Very much

Further comments:

BuildAR

Thank you for trying AR authoring tool! Please answer a few questions about your experience. See blow for how your answers will be used.

On a scale of 1 to 7, please circle the number according to how much you feel strongly agree or strongly disagree with the following statements:

I can easily add a marker

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily load a 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily translate the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily rotate the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily scale the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily browse the entire set of 3D models by using this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily control this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily tell what was going on

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I feel very comfortable while using this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

Please rate your agreement with the following statements about your experience using this authoring tool.

I always understand clearly what I was supposed to do

Disagree 1 2 3 4 5 6 7 Agree

I had sometimes problems with the user interface

Disagree 1 2 3 4 5 6 7 Agree

The tool was sometimes confusing

Disagree 1 2 3 4 5 6 7 Agree

The tasks were easy to solve

Disagree 1 2 3 4 5 6 7 Agree

User Interface

The user interface made me feel in control

Disagree 1 2 3 4 5 6 7 Agree

The user interface was easy to use

Disagree 1 2 3 4 5 6 7 Agree

Overall

I enjoyed using the tool

Disagree 1 2 3 4 5 6 7 Agree

Using the tool was a great experience

Disagree 1 2 3 4 5 6 7 Agree

How much did you like using this authoring tool?

Not very much 1 2 3 4 5 6 7 Very much

Further comments:

XML Editor

Thank you for trying XML Editor! Please answer a few questions about your experience. See blow for how your answers will be used.

On a scale of 1 to 7, please circle the number according to how much you feel strongly agree or strongly disagree with the following statements:

I can easily add a marker

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily load a 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily translate the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily rotate the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily scale the 3D model

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily browse the entire set of 3D models by using this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily control this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I can easily tell what was going on

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

I feel very comfortable while using this tool

Strongly Disagree	1	2	3	4	5	6	7	Strongly Agree

Please rate your agreement with the following statements about your experience using this authoring tool.

I always understand clearly what I was supposed to do

Disagree 1 2 3 4 5 6 7 Agree

I had sometimes problems with the user interface

Disagree 1 2 3 4 5 6 7 Agree

The tool was sometimes confusing

Disagree 1 2 3 4 5 6 7 Agree

The tasks were easy to solve

Disagree 1 2 3 4 5 6 7 Agree

User Interface

The user interface made me feel in control

Disagree 1 2 3 4 5 6 7 Agree

The user interface was easy to use

Disagree 1 2 3 4 5 6 7 Agree

Overall

I enjoyed using the tool

Disagree 1 2 3 4 5 6 7 Agree

Using the tool was a great experience

Disagree 1 2 3 4 5 6 7 Agree

How much did you like using this authoring tool?

Not very much 1 2 3 4 5 6 7 Very much

Further comments:

Ranking

You have just experienced three applications which you just use to complete the task. Please rank the tools in order for the following questions.

1	2	3	4	5	6	7
		XML Editor	BuildAR	ComposAR		
Not very easily					Very easily	
Not very interesting					Very interesting	
Not very boring					Very boring	
Not very precise					Very precise	
Not very obvious					Very obvious	
Not very efficient					Very efficient	
Not very fun					Very fun	
Not very skilled					Very skilled	
Not very satisfying					Very satisfying	
Not very engaging					Very engaging	

Further comments: