How Mobile Phones Perform in Augmented Reality Marker Tracking?

Víctor Fernández, Juan M. Orduña, Pedro Morillo [1]

Abstract— **Collaborative Augmented Reality (CAR) systems allow multiple users to share a real work environment including computer-generated images in real time. Currently, the hardware features of most mobile phones not only provide excellent multimedia services, but they also include wireless network capabilities to support local and remote communication and thus, they provide a natural platform for CAR systems. However, the wide variety of these hardware features can have important effects on the performance of the mobile CAR applications.**

This paper presents the experimental characterization of mobile phones for Augmented Reality marker tracking, a core task that any CAR application must include. The characterization is performed in regard to well-known performance metrics in distributed systems, and the results show that the most time consuming stage is the marker detection stage, followed by the image acquisition stage. Therefore, the improvement of these applications should be addressed to improve these stages. Moreover, the rendering stage is decoupled on some devices, depending on the operative system used. This decoupling process allows avoiding low refresh rates, facilitating the collaborative work. These results can be used as the basis for an efficient design of CAR systems and applications.

Key words— **Collaborative Augmented Reality; Marker tracking; Mobile Phones; Performance improvement**

I. Introduction

AU gmented Reality (AR) systems are a combination of visualization, tracking and interaction devices systems aimed to interactively blend real images and virtual objects in a real environment. The sense provided to the end user is that the real world is supplemented with virtual (computer-generated) objects that appear to coexist in the same space as the real world. Since the beginning of AR systems, the potential of collaborative AR (CAR) systems was exploited for different activities like Teleconferencing or Collaborative Computing [1]. Wearable devices were used to provide CAR systems where a wearable AR user could collaborate with a remote user at a desktop computer [2].

On other hand, the continue improvement in silicon technology, together with the evolution of design methodologies, allowed to integrate complex computing Systems-on-Chip (SoCs). As a result, a lot of devices comprising a computing embedded system pervade our daily life, and they have been used for CAR systems. One of these devices are mobile phones. Effectively, current mobile phones have full color displays, integrated cameras, fast processors and even dedicated 3D graphics chips, and they have become an ideal platform for CAR systems [3]. However, the wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems, can have significant effects on the performance of the CAR application, in terms of system latency, frames per second or number of supported clients with certain latency levels. These effects are particularly important in the process of the AR marker tracking, an essential process that takes place in any CAR application. Taking into account that CAR applications should be interactive, the design of an efficient marker tracking process must take into account these effects in order to fulfill the required specifications.

In this paper, we propose an in-depth performance characterization of different mobile phones for Augmented Reality marker tracking, starting from some preliminary results [4]. We have implemented a simple Augmented Reality marker tracking application on a real system, and we have measured the performance achieved with different mobile phones. In order to ensure a representative study of the mobile phone market, we have considered different mobile phones based on two different operating systems (OS): Android OS [5], and iOS [6]. The results show that the most time consuming stage in an AR application is the marker detection stage, followed by the image acquisition stage, the rendering stage and finally, the transmission stage. Therefore, any improvement of the CAR applications (categorized from the obtained results as CPU-intensive but not memory-intensive) should be addressed to improve the image acquisition process. Regarding the operating systems, the results show that the rendering stage is decoupled on devices using the Android OS, in such a way that it is executed with the other stages concurrently. However, this stage can be programmed to work as in iOS operating system in ad-hoc implementations. The results also show that some recent mobile phones like iPhone 4 [6] only works with high resolution images. As a result, these mobile devices need a lot of time for detecting the markers in the camera image.

The rest of the paper is organized as follows: Section II shows some related work about CAR applications on mobile phones. Section III presents some details on how AR marker tracking is implemented on mobile phones. Next, Section IV describes the characterization setup, and Section V shows the characterization results. Finally, Section VI presents some conclusions and future work to be done.

[1]Departamento de Informática, Universidad de Valencia, e-mail: {Victor.Fernandez-Bauset, Juan.Orduna, Pedro.Morillo}@uv.es

II. Related Work

Augmented Reality superimposes multimedia content - 3D object, text, sound, etc - on real world through a display or screen. In order to locate digital contents on a specific image of the real world point, some references within the image are needed. These references are known as markers, and two methods are usually used to track them: natural feature tracking and fiducial marker tracking. The former method uses interest point detectors and matching schemes to associate 2D locations on the video with 3D locations [7]. This process can be grouped in three large phases: interest point detection, creation of descriptor vectors for these interest points, and comparison of vectors with the database [8]. The latter method uses fiducial markers to find a specific position of real world. This process can be divided in three phases: edge detection, rejection of quadrangles that are too large or too small, and checking against the set of known patterns [7].

There are few solutions based on fiducial marker tracking over mobile phones. In 2003, ArToolKit [9], one of the most well-known software libraries for developing Augmented Reality (AR) applications, was released for Windows CE, and the first self-contained application was developed for mobile phones. This software evolved later as the ArToolKitPlus tracking library [7]. Moreover, a tracking solution for mobile phones that works with 3D color-coded marks was developed [10], and a version of ArToolKit for Symbian OS was developed, partially based on the ArToolKitPlus source code [3]. The research teams behind these works have worked on fiducial marker tracking, but not from the collaborative point of view. Also, there are many other works that focus on natural feature tracking [7], [11], [12], [13].

Although real-time natural feature tracking over mobile devices has been currently achieved [7], fiducial marker tracking is more widely used, because it allows simultaneous computational robustness and efficiency. A large number of locations and objects can be efficiently labeled by encoding unique identifiers on the markers. Additionally, the markers can be detected with angles near to 90 degrees [7].

III. CAR Applications on Mobile Phones

Any CAR application needs a device equipped with an on-board camera, CPU and display. The most common devices used for CAR applications are Tablet PCs or mobile phones. We will focus on mobile phones, because they are more wearable devices than tablet PCs, and therefore they are more suitable for many CAR applications designed for daily life common situations [14].

There are different kinds of mobile phones, with different operative systems (OS) and capabilities. The most extended OSs for mobile phones are Nokia Symbian, Google Android OS (commonly referred as Android), RIM/Blackberry, Apple iOS, Microsoft Windows Mobile/Phone 7 and Samsung Bada [15]. In this work, we are focusing on two of them, Android and iOS, because they share the vast majority of the current market [16].

The Augmented Reality marker tracking process in CAR applications can be split into four stages, as depicted in Figure 1: The first stage is denoted as image acquisition stage, and it consists of obtaining an image from the camera's flow. In the second stage, markers are detected from the image obtained before. Using the position of this markers, the third stage consists of drawing a 3D object on the image. Finally, in the fourth phase this information (for example, the position(s) of the mark(s)) is sent to the other application nodes through some kind of broadcast communication.

The first three phases are similar on any AR application [7], but the last one can be performed by using different technologies like WiFi, 3G or Bluetooth [17]. Although there are some classic CAR applications that uses Bluetooth, usually WiFi or 3G technologies are used, since the use of Bluetooth severely limits the spatial range of transmission.

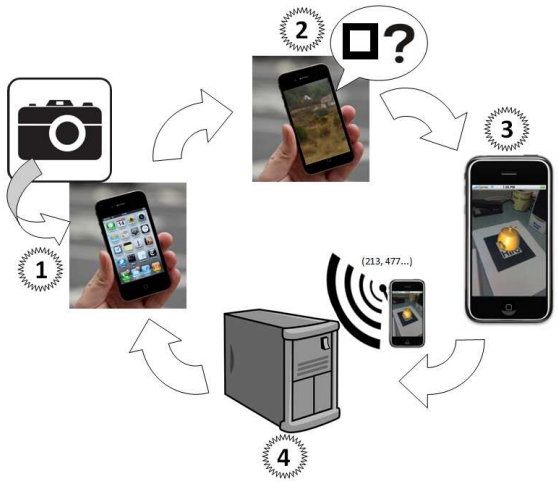

Fig. 1. A description of the most common stages in the AR marker tracking process.

IV. Characterization Setup

In this work, we propose the characterization of each of the stages of an AR marker tracking process in different mobile phones. For characterization purposes, the application performed the sending of the positions of the marks. Different types of markers are available, such as ARToolKit, ARToolKitPlus, ARTag [18], ARSTudio, QR-Code, ShotCode, etc.. However, the most widely used are the first two ones, due to their source code availability [3]. For that reason, we have selected the ARToolKitPlus library. ARToolKitPlus is the ARToolKit version for mobile devices that, among other adjustments, eliminates the use of floating point arithmetic, since most mobile devices are not optimized to deal with it[7]. Finally, we focus on two different operating systems that are widely used in mobile phones: Android and iOS.

A. Test description

We have tested two different mobile phones using Android and another two mobile phones using iOS operating system. Table I shows the main features of these mobile phones. For the Android operative system, we have considered the Motorola Milestone, with 550 MHz of CPU frequency, and Nexus One, with almost double CPU frequency (998 MHz). For the iOS operating system, we have considered the iPhone 3G, with 412 MHz of CPU frequency, and iPhone 4, with double CPU frequency (800 MHz). All of them include a 5 megapixels (5MP) resolution camera, except iPhone 3G, which equips a 2 megapixels (2MP) resolution camera.

TABLE I
HARDWARE FEATURES OF THE CONSIDERED MOBILE PHONES

OS	Android	
Model	Milest.	Nex. One
CPU (MHz)	550	998
RAM (MB)	256	512
Camera (MP)	5,02	4,92
OS	iOS	
Model	iPh 3G	iPh 4
CPU (MHz)	412	800
RAM (MB)	128	512
Camera (MP)	1,92	4,92

On the Internet, there are some implementations for Android and iOS that are open source. We have used them as a starting point to obtain an AR marker tracking implementation. As we mentioned earlier, CAR applications are closely related with AR applications. Previous studies on AR applications show that the mark size does not affect performance, that the tracking computing is primarily CPU bound and not influenced much by the operating system, and that the tracking performance increases linearly with the CPU clock [19]. The problem of changing lighting conditions is solved on ARToolKitPlus with an Automatic thresholding, so there is no need to worry about this issue. The increased resolution on the camera provides only minimal improvement in the tracking quality [7]. Our purpose is to analyze the amount of time that each stage needs to run, the CPU consumption, the amount of memory that it requires, and the round-trip delay of the data transmission. We have performed all the tests with a single mark, since multi-marker tracking provides highly stable tracking [7].

B. Android implementation

NyARToolKit [20] is a complete version of ARToolkit [9] that was exclusively written in Java. This makes it slower in execution than the original, but it also makes it completely independent of the processor architecture. As the original, NyARToolKit is a library of functions oriented to visual interpretation and integration of Virtual Reality (VR) data into physical environments, including real-time camera vision functionality, 3D rendering of virtual objects, and integrating both into the output stream. Despite being native to Java, the toolkit works with C# and the Android operating system, and it uses OpenGL ES for rendering. After obtaining the source code, we analyzed it to delimit each of stages of the AR marker tracking process by adding timestamps. Then, we added the sending stage, creating a TCP socket that sends the information to a Server or other devices. Among the different camera resolutions that offers Android, we have chosen a small one, in order to provide a fast way to find the mark from the image obtained. Concretely, we have used a resolution of 320x240 pixels for both mobile phones.

Figure 2 shows a snapshots of the AR marker tracking implementation performed during a test with real industrial elements, for illustrative purposes. Concretely, this image shows the results of a CAR system developed for remotely repairing an engine in a industrial environment. In this process, the on-site worker (which is repairing on-site the machine at the factory) is guided by the qualified technician (at the laboratory) until the maintenance/repair task is completed. The snapshot shows that the AR marker tracking allows superimposing green arrows pointing to the three connectors that the on-site worker must disconnect in this step. In this case, the CAR application also adds some instructions in text (written in Spanish).

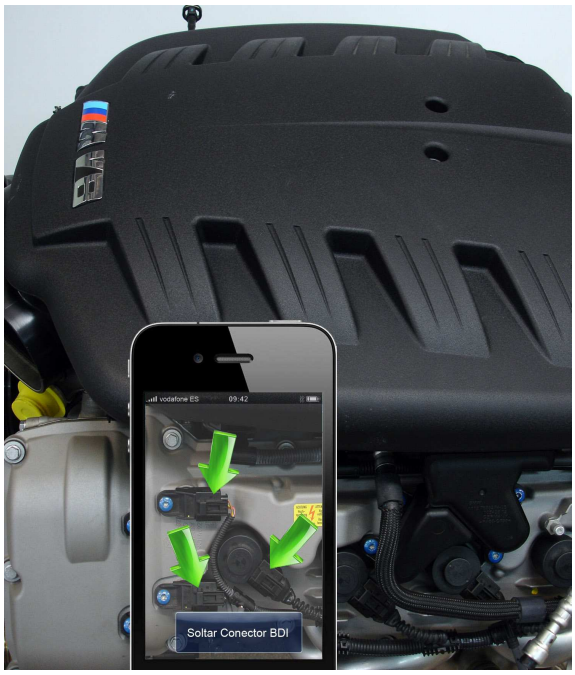

Fig. 2. A snapshot of the marker tracking implementation.

C. iOS implementation

The iOS implementation has been developed by Benjamin Loulier [21] and is based on ARToolKitPlus [7]. Among the features provided by this application, we can found single marker detection (the marker detection is done using an objective-c wrapper developed over ARToolKitPlus), loading of 3D

objects using custom XML and ".h" files (or ".obj", but the parser is very slow for now), and only one texture file is supported. The association between a markerID and an object is done by using a XML exchange file, which in turn gives access to a GUI to modify the display parameters associated to an object[22]. It also uses OpenGL ES for rendering.

After getting the application, we did the same procedure as in the Android version: analyzing its stages, putting time marks and adding the sending stage, also with TCP sockets.

In contrast to Android, iOS only provides two camera resolutions: full or half. In half resolution it obtains the same resolution that in full resolution, but it only analyzes one of every two pixels. In order to make the fairest comparisons as possible, we used half of the resolution, with a resolution of 400x304 pixels on iPhone 3G and a resolution of 1280x720 pixels on iPhone 4.

V. Performance Evaluation

The main performance metrics in distributed systems are throughput and latency. From the point of view of graphics, throughput is measured in frames per second (FPS) and latency on milliseconds (ms). However, in distributed systems the latency of data exchanged among different devices cannot be measured with accuracy, due to potential clock skews between the sending and the receiver clocks. In these cases, the round-trip-time (RTT) is used, since it allows that the sending and received instant are measured by the same clock.

Table II shows the performance evaluation, in terms of the average number of FPS achieved by each mobile phone, and the latency for the transmission stage of the application.

TABLE II
Throughput (in terms of FPS) and RTT for each smartphone

	FPS	RTT (ms)
Milestone	1,43	14,14
Nexus One	5,98	5,54
iPhone 3G	2,51	15,42
iPhone 4	1,91	7,06

Table II shows that the Motorola Milestone is the handheld device with the poorest throughput, and the Nexus One is the one with the best throughput. Although the iPhone 3G provides better throughput than the iPhone 4G, this behavior is due to the fact that the iPhone 3G has to analyze much less amount of data than the iPhone 4, since the size of its image is much lower. The right column in this table shows that the latency is quite similar in all the mobile phones considered, as it could be expected, because it depends on network features more than the computing capabilities of the considered mobile phone.

The next step in our characterization study consists of the analyzing of the throughput achieved by each device considered. Table III shows the decomposition of the results shown in Table II. Concretely, the first four columns show the average duration of each stage per cycle for each device, and the rightmost column shows the total aggregated cycle time. The inverse of these times result in the number of frames per second shown for each device in Table II.

TABLE III
Execution time (ms) per stage for each considered mobile phone

Stages(ms)	Acq.	Detect	Render	Send	Total
Milestone	249	289	30	14	698
NexusOne	40	78	13	5	167
iPhone3G	33	58	28	15	398
iPhone4	18	182	23	7	523

Each row in Table III shows the number of milliseconds that each stage needs to finish on each device. The Motorola Milestone provides the worst throughput because it is six times slower in obtaining images, and almost twice slower in detecting a mark, than the next in the list. Regarding the rendering and sending stages, there are also significant differences with other devices. When comparing both OSs, we can see that the differences among the "slow" devices are not significant, but the difference between the "fast" devices (Nexus One and iPhone 4) are important, providing best rendering and transmission times the Nexus One.

On other hand, the behavior shown by iOS-based devices for AR purposes is different from the behavior shown by another mobile phones. In this sense, the image acquisition process is twice faster on a iPhone 4 than the same stage performed on an iPhone 3G. However, the marker detection stage in an iPhone 4 is three times slower than in the iPhone 3G. Although the CPU of an Iphone 4 is twice as powerful as the CPU included in an Iphone 3G, the reason of this result is that the images processed in an iPhone 4 are six times bigger than the images processed by an iPhone 3G. In this sense, Table III shows that iPhone 4 is slightly faster than the former model for both the rendering and sending stages.

When Android-based devices are compared to Apple's mobile phones in the analysis of the marker detection stage, Table III shows that the Android devices are faster than iOs phones because the images captured from the on-board cameras equipped in the Android devices are four times smaller than the images captured by an iPhone 4. In this sense, iPhone 3G takes less time than Android devices to complete the marker detection stage taking not only into account that the images captured by an iPhone 3G has a size similar to the images obtained by Android devices, but also the CPU of an HTC Nexus One (Android) is twice as powerful as the CPU included in an Iphone 3G. Regarding the rendering stage, the iOS-based smartphones are twice slower than the fastest Android. Finally, the sending stage is completed slightly faster on Android devices.

Table III shows that if total execution time (included in the last column) is compared to the reciprocal value of the throughput (FPS in Table II)

both Android devices have almost a perfect matching. On the contrary, iOS smartphones need twice times to complete the cycle. This unexpected and non-sequential behavior can be due to: i) Android seems to leave its applications very often. ii) Android applications do not need to manage the memory directly because they are executed on a Virtual Machine (VM), which manages automatically the memory. iii) The CAR stages in Android-based devices are implemented in independent threads and are not executed in a blocking manner. Since the rendering thread is decoupled with the rest of the threads of the CAR framework, this thread does not wait the update performed by the marker detection stage.

These results seem to indicate that the Android devices render the final Augmented Reality scenes more often than the iOS devices. In order to confirm this result, we have measured the number of completed rendering stages compared to the rest of the threads of the CAR framework. The obtained averaged values are 6.28 renderings per cycle in Motorola Milestone and 5.55 renderings per cycle in the case of the HTC Nexus One.

The number of extra renderings per cycle depends on the number of polygons and the amount of texture data in the 3D model. Table IV shows the relation between the number of renderings per cycle and the complexity of the 3D scene for the Motorola Milestone and the HTC Nexus One. In order to differentiate complexities we have selected two classical 3D models as benchmarks consisting in a cylinder (simple 3D model, labeled as "simple") and a plane (complex 3D model, labeled as "complex").

TABLE IV
RELATION OF THE NUMBER OF RENDERINGS PER CYCLE AND THE COMPLEXITY OF THE 3D SCENE FOR DIFFERENT MODELS ON ANDROID DEVICES

	Render (ms)	R_{Render}	FPS
Milestone (simple)	21,40	18,24	1,23
Milestone (complex)	48,32	9,90	1,25
Nexus One (simple)	5,48	6,27	5,80
Nexus One (complex)	24,46	4,47	5,87

The first column in Table IV shows the number of milliseconds that the rendering stage needs to finish on each device and 3D model. The next column (R_{Render}) indicates the the number of renderings per AR cycle (repetitions of the same control programs), and finally the last column (FPS) shows the application throughput expressed in frames per second. A similar experiment in iOS devices only generates an slight increase in the total time of the AR cycle. Table IV shows that both Android smartphone (HTC Nexus One and Motorola Milestone) require more time to complete the rendering stage as the complexity of the 3D models is increased. In this experiment, the average time required to complete the rendering stage for the complex 3D models is the double of the time needed in the case of the simple 3D models. This variation is more evident for the parameter corresponding to the number of repetitions of the rendering stage. Since the rendering stage needs more time as the complexity of the 3D model is increased, the number of repetitions of this stage in the regular cycle of the AR application is decreased to maintain a constant application throughput.

VI. Conclusions and Future Work

In this paper, we have proposed a performance characterization of mobile phones oriented to provide a robust and efficient design for Collaborative Augmented Reality (CAR) Applications. In order to ensure an analysis of the mobile phone market for CAR purposes (mid-and high-end mobile device), we have considered different mobile phones based on Android OS and iOS operating systems. These devices have been used to execute a simple AR marker tracking process on a real system where we have measured the achieved performance.

The results of the performance evaluation show that when the same AR marker tracking process is executed on different mobile phones, the best throughput, measured in frames per second (FPS), is obtained for smartphones based on Android operative platforms. However, as the hardware capabilities of the mobile phones decrease, iOS-based devices reach and exceed the performance of Android-based smartphones.

In order to analyze more in detail the relation between the hardware features and the performance of AR marker tracking process on different mobile phones, we have studied the different stages that compose a common AR marker tracking process. The results show that the most time consuming stage in this process is the marker detection stage, followed by the image acquisition stage, the rendering stage and finally, the transmission stage. Therefore, any improvement of the AR marker tracking process should be oriented to enhance the image acquisition process. Regarding the execution of CAR applications on mobile phones in different operating systems, the results show that the rendering stage is decoupled on devices using the Android OS, in such a way that it is executed with the rest of the stages concurrently. However, this stage can be programmed to work as in iOS operating system in ad-hoc implementations. Moreover, the results also show that some recent mobile phones like iPhone 4 [6] only works with high resolution images. As a result, these mobile devices achieves the most visual quality at the expense of needing a lot of time for detecting the markers in the camera image plane.

Acknowledgements

This work has been jointly supported by the Spanish MICINN and the European Commission FEDER funds under grants Consolider-Ingenio CSD2006-00046, TIN2009-14475-C04-04, and TIN2010-12011-E.

References

[1] M. Billinghurst, I. Poupyrev, H. Kato, and R. May, "Mixing realities in shared space: an augmented reality inter-

face for collaborative computing," in *IEEE International Conference on Multimedia and Expo (ICME 2000)*, 2000, vol. 3, pp. 1641–1644.

[2] Wayne Piekarski and Bruce H. Thomas, "Tinmith-hand: Unified user interface technology for mobile outdoor augmented reality and indoor virtual reality," 2002.

[3] A. Henrysson, M. Billinghurst, and M. Ollila, "Face to face collaborative ar on mobile phones," in *Mixed and Augmented Reality, 2005. Proceedings. Fourth IEEE and ACM International Symposium on*, October 2005, pp. 80 – 89.

[4] Victor Fernández Bauset, Juan M. Orduña, and Pedro Morillo, "Performance characterization on mobile phones for collaborative augmented reality (car) applications," in *Proceedings of the 2011 IEEE/ACM 15th International Symposium on Distributed Simulation and Real Time Applications*, Washington, DC, USA, 2011, DS-RT '11, pp. 52–53, IEEE Computer Society.

[5] Google, "Android," 2011, Available at http://www.android.com/index.html.

[6] Apple, "ios 4," 2011, Available at http://www.apple.com/iphone/ios4/.

[7] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg, "Pose tracking from natural features on mobile phones," in *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, Washington, DC, USA, 2008, ISMAR '08, pp. 125–134, IEEE Computer Society.

[8] Seung Eun Lee, Yong Zhang, Zhen Fang, S. Srinivasan, R. Iyer, and D. Newell, "Accelerating mobile augmented reality on a handheld platform," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, October 2009, pp. 419 –426.

[9] Dr. Hirokazu Kato, "Artoolkit," 2011, Available at http://www.hitl.washington.edu/artoolkit/.

[10] Mathias Mahring, Christian Lessig, and Oliver Bimber, "Video see-through ar on consumer cell-phones.," in *ISMAR'04*, 2004, pp. 252–253.

[11] S. Srinivasan, Zhen Fang, R. Iyer, S. Zhang, M. Espig, D. Newell, D. Cermak, Yi Wu, I. Kozintsev, and H. Haussecker, "Performance characterization and optimization of mobile augmented reality on handheld platforms," in *Workload Characterization. IISWC 2009. IEEE International Symposium on*, 2009, pp. 128 –137.

[12] D. Wagner, G. Reitmayr, A. Mulloni, T. Drummond, and D. Schmalstieg, "Real-time detection and tracking for augmented reality on mobile phones," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 16, no. 3, pp. 355 –368, May-June 2010.

[13] D. Wagner, D. Schmalstieg, and H. Bischof, "Multiple target detection and tracking with guaranteed framerates on mobile phones," in *Mixed and Augmented Reality. ISMAR 2009. 8th IEEE International Symposium on*, 2009, pp. 57 –64.

[14] Michael Haller; Mark Billinghurst; Bruce Thomas, *Emerging Technologies of Augmented Reality: Interfaces and Design*, IGI Global, 2007.

[15] Tomi Ahonen, *TomiAhonen Phone Book 2010*, TomiAhonen Consulting, 2010.

[16] Sharon P. Hall and Eric Anderson, "Operating systems for mobile computing," *J. Comput. Small Coll.*, vol. 25, pp. 64–71, December 2009.

[17] Andreas Schmeil and Wolfgang Broll, "An anthropomorphic ar-based personal information manager and guide," in *Proceedings of the 4th international conference on Universal access in human-computer interaction: ambient interaction*, Berlin, Heidelberg, 2007, UAHCI'07, pp. 699–708, Springer-Verlag.

[18] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, June 2005, vol. 2, pp. 590 – 596 vol. 2.

[19] Dieter Schmalstieg and Daniel Wagner, "Experiences with handheld augmented reality," in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, 2007, pp. 3 –18.

[20] Nyatla, "Nyartoolkit:artoolkit class library for java/c#/android," 2011, Available at http://nyatla.jp/nyartoolkit/.

[21] Benjamin Loulier, "Augmented reality on iphone using artoolkitplus," 2011, Available at http://www.benjaminloulier.com/.

[22] Benjamin Loulier, "Virtual reality on iphone," 2011, Available at http://www.benjaminloulier.com/articles/virtual-reality-on-iphone-code-inside.