J Supercomput (2013) 65:1179–1191
DOI 10.1007/s11227-013-0925-8

How mobile phones perform in collaborative augmented reality (CAR) applications

Víctor Fernández · Juan M. Orduña · Pedro Morillo

Published online: 6 April 2013

Abstract This paper presents the experimental analysis of mobile phones for Augmented Reality marker tracking, a core task that any CAR application must include. The results show that the most time consuming stage is the marker detection stage, followed by the image acquisition stage. 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. However, the use of multicore devices does not significantly improve the performance provided by CAR applications. Finally, the results show that unless a poor network bandwidth makes the network to become the system bottleneck, the performance of CAR applications based on mobile phones will be limited by the detection stage. These results can be used as the basis for an efficient design of CAR systems and applications based on mobile phones.

Keywords Collaborative augmented reality · Marker tracking · Mobile phones

1 Introduction

Since the beginning of Augmented Reality (AR) systems, the potential of collaborative AR (CAR) systems was exploited for different activities like Collaborative Computing [5] or Teleconferencing [4]. Wearable devices were used to provide CAR

V. Fernández · J.M. Orduña (✉) · P. Morillo
Departamento de Informática, Universidad de Valencia, Avda. Universidad, s/n. 46100 Burjassot, Valencia, Spain
e-mail: Juan.Orduna@uv.es

V. Fernández
e-mail: Victor.Fernandez-Bauset@uv.es

P. Morillo
e-mail: Pedro.Morillo@uv.es

systems where a wearable AR user could collaborate with a remote user at a desktop computer [9, 18].

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 [11, 15].

An essential process that takes place in any CAR application is the process of the AR marker tracking. 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 [22]. 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 [12]. The latter method uses fiducial markers to find a specific position of real world. 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. However, the wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems, can have significant effects on the AR marker tracking process, in terms of system latency, frames per second or number of supported clients with certain latency levels.

In this paper, we propose an in-depth performance characterization of different mobile phones for Augmented Reality marker tracking, starting from some preliminary results [3]. 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 [7], and iOS [2].

The results show that the most time consuming stage is the marker detection stage, followed by the image acquisition stage. Therefore, any improvement of the CAR applications (categorized as CPU-intensive but not memory-intensive [19]) 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. Therefore, CAR applications can provide better performance if the terminals use this operating system. However, this stage can be programmed to work as in iOS operating system in ad-hoc implementations. On other hand, the use of multicore devices does not significantly improve the performance provided by CAR applications. The results also show that some recent mobile phones like iPhone 4 [2] 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. These results can be used as the basis for an efficient design of CAR systems and applications based on mobile phones.

The rest of the paper is organized as follows: Sect. 2 presents some details on how AR marker tracking is implemented on mobile phones. Next, Sect. 3 describes the

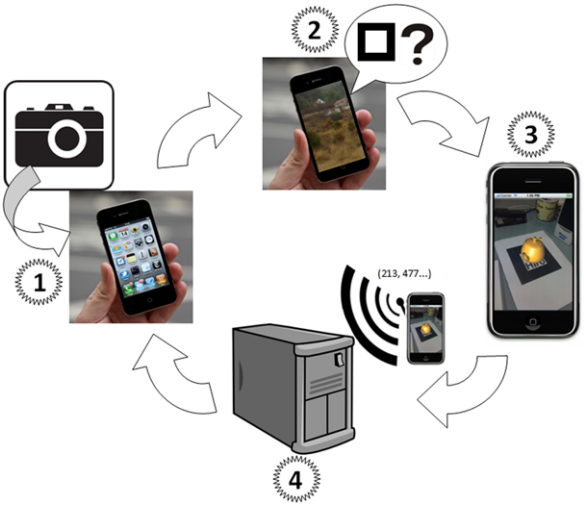

Fig. 1 A description of the most common stages in the AR marker tracking process

characterization setup, and Sect. 4 shows the characterization results. Finally, Sect. 5 presents some conclusions and future work to be done.

2 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 [21].

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 [1]. In this work, we are focusing on two of them, Android and iOS, because they share the vast majority of the current market [8].

The Augmented Reality marker tracking process in CAR applications can be split into four stages, as depicted in Fig. 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 [22], but the last one can be performed by using different technologies like WiFi, 3G or Bluetooth [20]. 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.

Table 1 Hardware features of the considered mobile phones

OS	Android			iOS		
model	Milest.	Nex. one	S.G.SIII	iPh 3GS	iPh 4	iPh 4S
CPU	TI OMAP 3430	Qual. QSD8250	32 bit Samsung Exynos 4412	Samsung S5PC100	Apple APL0398	Apple A5 APL0498
Freq. (MHz)	550	998	1400	412	800	800
GPU	SGX530	Qual. Adreno200	ARM Mali-400	SGX535	SGX535	PowerVR SGX543MP2
RAM (MB)	256	512	1024	128	512	512
Camera (MP)	5.02	4.92	7.99	1.92	4.92	7.99
N. of Cores	1	1	4	1	1	2

3 Characterization setup

In this work, we propose the characterization of each of the stages of a CAR with marker tracking over different mobile phones. For characterization purposes, we have considered the coordinates of the mark(s) found within the image as the information that the client should send to the server. We have considered both single core devices and multicore devices for characterization purposes. Also, we have tested two different mobile phones using Android and another two mobile phones using iOS operating system (when studying single core devices). Table 1 shows the main features of these mobile phones, including the CPU, GPU models, RAM capacity, camera resolution and number of cores. 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). The Motorola Milestone terminal executes the Android 2.0 version, while the Nexus One terminal executes the Android 2.1 version. For the iOS operating system, we have considered the iPhone 3GS, with 412 MHz of CPU frequency, and iPhone 4, with double CPU frequency (800 MHz). Both the iPhone 3GS and iPhone 4 execute the iOS 4.3.2 version. All of them include a 5 megapixels (5 MP) resolution camera, except the iPhone 3GS, which equips a 2 megapixels (2 MP) resolution camera. Regarding the GPU, the Milestone and both iPhone models contain PowerVR SGX graphical drivers, while the Nexus One has a Qualcomm GPU. Regarding Android OS, we have tested one low-end mobile phone called Motorola Milestone, a middle-end device named Nexus One, and a high-end multicore device named Samsung Galaxy SIII. For iOS operating system, we used an equivalent device to each category, which names are iPhone 3GS, iPhone 4 and iPhone 4S, respectively. The features of each mobile are included in Table 1.

As we mentioned earlier, CAR applications are closely related with AR applications. In fact, CAR is an extension of AR that includes the communication stage, as we will describe later. Previous studies on AR applications show that the mark size does not affect performance in which 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. The increased

resolution on the camera provides only minimal improvement in the tracking quality [22]. 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 [22].

Different types of markers are available, such as ARToolKit, ARToolKitPlus, ARTag [6], ARSTudio, QR-Code, ShotCode, etc. However, the most widely used are the first two ones, due to their source code availability [10]. 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. Concretely, we have used the ARToolKitPlus in its 2.1.1. version. Also, we focus on two different operating systems that are widely used in mobile phones: Android and iOS.

On the Internet, there are some implementations for Android and iOS that are open sources. We have used them as a starting point to obtain an AR marker tracking implementation. Concretely, for the Android implementation, we have used the code provided in the NyARToolKit web site [16]. For the iOS implementation, we have used the implementation made by Benjamin Loulier, which uses ARToolkitPlus, and can be found on his blog [13].

NyARToolKit [17] is a version of ARToolkit that was exclusively written in Java; it 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, using Open GL, and integrating both into the output stream. Concretely, we have used the NyARToolKit version 2.5.2. 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 the smaller one, in order to provide a fast way to find the mark from the image obtained. Concretely, we have used a resolution of 320×240 pixels for all mobile phones. For illustration purposes, the pseudocode corresponding to the instrumentalized version of the code (for the case of Android OS) is shown in Fig. 2.

We have developed two CAR applications starting from the AR libraries NyARToolKit (Android) and the implementation made by Benjamin Loulier (iOS). We added the send stage to these implementations in order to make them collaborative, and using these codes we developed the particular applications. Figure 3 shows two snapshots of the Android implementation taken during a test with real industrial elements. In this case, the system has been used as a remote assistant to control some critical steps in a car maintenance procedure. Concretely, the disconnection of the sixteen ignition coil connectors in an 8-cylinder BMW engine, as shown in the figure, should be carried out in a proper sequence in order to avoid important damages in the electrical installation of the engine. The two snapshots show two different steps of the repairing process: the release of some connectors and the extracting of an ignition coil.

As mentioned before, for iOS devices, we used the implementation developed by Benjamin Loulier [13], based on ARToolKitPlus [22]. Among the features provided

```
begin
  1. timeC1   //Camera_s tarts
  2.   mCameraDevice.setOneShotPreviewCallback(mOneShotPreviewCallback);
  3.   cb.onPreviewFrame(data, null);
  4. timeC2 //Camera_ends
  5. timeM1   //MarkerDetect_starts
  6.   createNyARTool(width, height);
  7.   found_markers = nya.detectMarkerLite(raster, 100);
  8. timeM2   //MarkerDetect_ends
  9. timeR1   //Render_starts
  10.          gl.glMatrixMode(GL10.GL_MODELVIEW);
  11.          model[ar_code_index[i]].disables(gl);
  12. timeR2 //Render_ends
  13. timeS1   //Send_to_the_server
  14.          out.println(message);
  15.          while ((c = in.read()) != -1);
  16. timeS2   //Receive_ACK_from_the_server
end
```

Fig. 2 Pseudocode of the instrumentalized source code (Android)

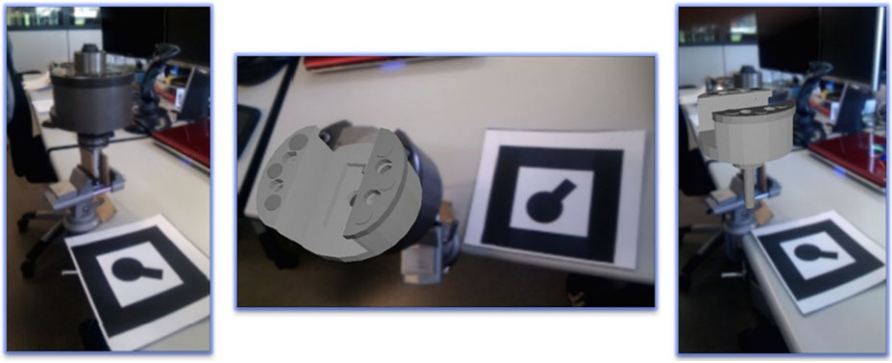

Fig. 3 Some snapshots of the Android implementation obtained from a HTC Nexus One

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 [14]. 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. The pseudocode shown in Fig. 2 is also valid here.

In contrast to Android, iOS only provides two camera resolutions: full or half. In half resolution, it obtains the same resolution that is in full resolution, but it only analyzes one of every two pixels. In order to make the fairest comparison as possible, we used half of the resolution, with a resolution of 400 × 304 pixels on the iPhone 3GS and a resolution of 1280 × 720 pixels on the iPhone 4.

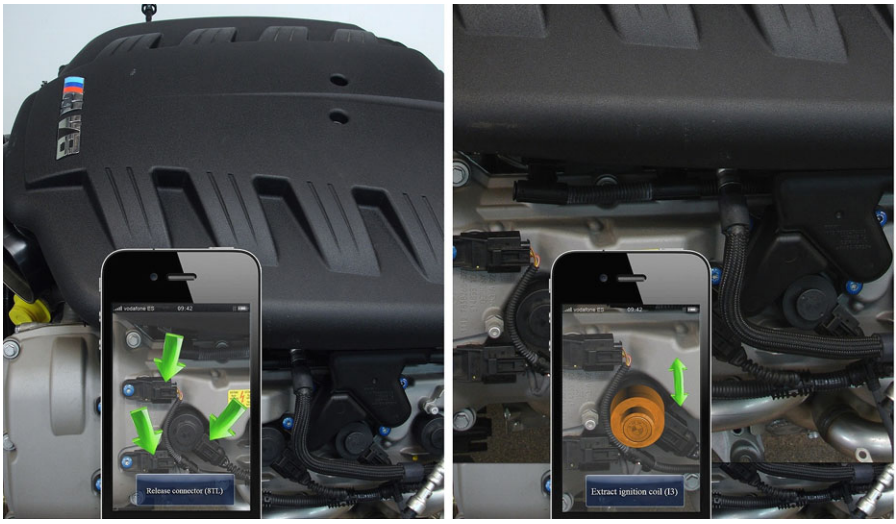

Fig. 4 Some snapshots of the iOS implementation using an iPhone 4 in a maintenance procedure

Figure 4 shows the use of the iOS implementation on an iPhone 4. In this case, the system has been used as a remote assistant to control some critical steps in a maintenance procedure. 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.

4 Performance evaluation

This section shows the performance evaluation of different mobile phones when used in a CAR application. Concretely, we have measured the latency and the number of frames per second that a CAR system could provide when using each model of mobile phone, analyzing the time required for each CAR stage. We have measured the latency (in milliseconds (ms)) as the time required for sending each new marker location update from each device to the server. 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 2 shows these times together with the total aggregated cycle time and its inverse value (FPS). We are looking for the most time consuming stage and we want to see how mobile phone manage in each stage, as well as see what OS must be chosen in order to achieve the best performance.

On Table 2, the first four columns show the average duration of each CAR stage per cycle for each device, the fifth column shows the total aggregated cycle time and the most right column shows the FPS. The column labeled as “total” shows the time elapsed between two same phases in consecutive cycles, that is, the time elapsed between two consecutive executions of the call “timeC1” in Fig. 2. So, it is the total

Table 2 Execution time (ms) per stage for each considered mobile phone

Stages (ms)	Acq.	Detect	Render	Send (RTT)	Total	FPS
Milestone	248.64	288.53	30.42	14.14	698.34	1.43
Nexus One	40.25	78.08	13.23	5.54	167.11	5.98
iPhone 3GS	33.29	58.07	28.26	15.42	398.00	2.51
iPhone 4	17.66	182.17	23.34	7.06	523.26	1.91
iPhone 4S	1.19	114.64	6.37	8.30	182.46	5.48
Samsung SIII	60.05	9.34	5.90	7.90	128.02	7.81

time needed to complete a CAR cycle. As mentioned before, the FPS showed on the most right column are obtained by converting the total time from milliseconds to seconds and inverting the resulting value.

Regarding the single core devices, 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. In this sense, it is worthwhile mentioning the great difference between the Milestone and the rest of the considered devices for the acquisition stage. The same goes for the rendering and sending stages, there are also significant differences with other devices. On the other hand, the Nexus One provides the best throughput, but this is a foregone conclusion because it is the faster device in all the stages, except for the first one. In the Sending stage, the latency remains almost constant (only ten milliseconds separate the best from the worst case), as it could be expected, because it depends on network features more than the computing capabilities of the considered mobile phone.

Table 2 also 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 3GS takes less time than Android devices to complete the marker detection stage taking not only into account that the images captured by an iPhone 3GS has a size similar to the images obtained by Android devices, but also the CPU of an Nexus One (Android) is twice as powerful as the CPU included in an iPhone 3GS. In this sense, the image acquisition process is twice faster on a iPhone 4 than the same stage performed on an iPhone 3GS. However, the marker detection stage in the iPhone 4 is three times slower than in the iPhone 3GS. Although the CPU of an iPhone 4 is twice as powerful as the CPU included in an iPhone 3GS, 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 3GS. On the other hand, the iPhone 4 is slightly faster than the former model for the sending and rendering stages, but both are twice slower than the fastest Android in this last stage.

Regarding the multicore devices, the quad-core device (Samsung SIII) does not provide a linear throughput with the number of cores with respect to the dual core device (iPhone 4S), and neither an inversely linear latency (RTT). Moreover, the single core device Nexus One provides better throughput than the iPhone 4S, a dual core device. Also, the latency (RTT) provided by the single core devices Nexus One and iPhone 4 are lower than the latencies provided by both multicore devices. These results clearly show that the use of multicore devices improve slightly the throughput

in CAR applications. This improvement is higher on iOS, since the iPhone 4 and the iPhone 4S work both at the same frequency (800 MHz), and with only an extra core the iPhone 4S obtains more than twice FPS. On the Android side, the benefits are lower, since Samsung Galaxy SIII, with more computing power and three extra cores only provides an increment of two FPS (from 5.98 to 7.81). So, it can be said that iOS has a better use of the extra cores than Android. Therefore, we have not achieved a significant improvement by adding new cores because the four stages described in the previous sections are inherently sequential and, therefore, they cannot take advantage of the parallelism offered by multicore devices. In fact, the code executed in both single core and multicore devices was exactly the same.

Although Table 2 shows execution times for very different devices and even single core and multicore platforms and, therefore, this table does not represent a fair comparison from the architectural point of view, it must be noticed that the purpose of this work is to analyze the resulting latencies that the different phones can provide to users in a CAR application, in order to select the best kind of device depending on the application constraints. Thus, for example, different devices manage images with very different resolutions, but CAR users/developers cannot change the resolution of images; they can only design CAR applications taking into account the latency provided by each kind of device. In this sense, the only important factor is the resulting latency, regardless the unfairness of the comparison.

Also, it could be expected that the total execution time is the aggregation of the time required by each CAR stage. In fact, this situation occurs in single core Android devices, which show a perfect matching between the total sum of the time required by all the stages and the cycle time (the column labeled as "Total" in Table 2). However, the cycle time in single core iOS smartphones is almost twice the aggregated time required by the four stages. This unexpected behavior can be due to the fact that Android applications do not need to manage the memory directly because they are executed on a Virtual Machine (VM), which automatically manages the memory. Alternatively, this unexpected behavior could be due to the fact that some CAR stages in Android-based devices are implemented in independent threads, and they are not executed in a blocking manner. As an example, the code for the rendering stage is decoupled with the rest of the CAR stages (it is executed as a separated thread), and it does not wait the update performed by the marker detection stage. In this way, when it is time to render the scene and the new position is not ready, the render thread works with the previous position.

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 3 shows the relation between the number of renderings per cycle and the complexity of the 3D scene for all the Android devices. In order to differentiate complexities, we have selected two classical 3D models as benchmarks consisting in a cylinder (simple 3D model) and a plane (complex 3D model).

Table 3 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 model)	21.40	18.24	1.23
Milestone (complex model)	48.32	9.90	1.25
Nexus One (simple model)	5.48	6.27	5.80
Nexus One (complex model)	24.46	4.47	5.87
Samsung SIII (simple model)	3.26	5.73	7.95
Samsung SIII (complex model)	9.83	3.11	7.96

Table 4 Memory and CPU consumption of the considered smartphones

Device	Memory (MB)	CPU (%)
Milestone	25	97
Nexus One	11	96
iPhone 3GS	7	95
iPhone 4	4	97
iPhone 4S	7.4	14.36
Samsung SIII	9.2	11.24

The first column in Table 3 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 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 3 shows that all Android smartphones 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. In this case, the use of multicore devices does not significantly change the performance obtained with the single core devices.

Additionally, we have measured the CPU and the memory utilization for the considered mobiles phones in the experiment. Table 4 shows the results of the performed test indicating in the first row the used memory size (in MB), as well as the percentage of CPU time executing the AR marker tracking process. The Android project is done using the development platform named Eclipse, and the iOS project uses the platform named XCode. In both architectures, the development platform provides tools to measure performance, so we have used these tools for measuring the results in the table.

Table 4 shows that all the considered single core smartphones are close to reach the saturation point, in terms of CPU usage since (100 % CPU usage). AR marker

tracking can be considered as a CPU-intensive process, since it demands the maximum microprocessor resources available in a coupled (iOS) or a decoupled (Android) mode of operation. However, the marker tracking process requires a significantly lower percentage of CPU utilization in multicore devices, decreasing from not less than 95 % in single core devices to 14.36 % in a dual core device. Also, it is worth mentioning that the small difference in the percentage of CPU utilization between the quad core and the dual core devices is around 3 %. Also, Table 4 shows that AR marker tracking processes are not necessarily memory-intensive. In terms of memory usage, the results obtained with the considered smartphones show that Android-based smartphones need more memory than the iOS-based mobile phones when the same AR marker tracking process is executed on them, regardless of the number of cores present in each device. The reason for this memory overhead is related to the management of the memory resources performed by the Android devices. Unlike iOS devices, where the programmer controls the amount of memory allocated to the AR marker tracking process, the management of the memory in Android smartphones is transparent to the application developers and it is based on a Virtual Machine (VM).

5 Conclusions and future work

In this paper, we have proposed a performance characterization of mobile phones in the AR marker tracking process, an essential process that takes place in any CAR application. In order to ensure an robust analysis of the mobile phone market for CAR purposes, we have considered different mobile phones based on Android and iOS operating systems. These devices have been used to execute a simple CAR application on a real system where we have measured the performance.

The performance evaluation results 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. The multicore devices do not provide a linear throughput with the number of cores, and even one of the single core devices provides a better throughput and latency. The reason for this behavior is that the four stages in AR marker tracking process are inherently sequential, and they cannot take advantage of the parallelism offered by multicore devices.

We have also 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. Regarding the 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. Therefore, CAR applications can provide better performance if the terminals use this operating system. Nevertheless, this stage can be programmed to work as in iOS operating system in ad hoc implementations. Moreover, the results also show that some mobile phones like the iPhone 4 only works with high resolution images. As a result, these

mobile devices achieve the most visual quality at the expense of needing a lot of time for detecting the markers in the camera image plane. Overall, we can conclude that unless a poor network bandwidth makes the network to become the system bottleneck, the performance of CAR applications based on mobile phones will be limited by the detection stage. These results can be used as the basis for an efficient design of CAR systems and applications.

For future work to be done, we plan to reproduce a similar study but focusing on CAR applications based on natural feature tracking. Since these applications tend to exploit the GPU's capabilities in the real-time feature tracking, the new graphic oriented hardware, included in the most recent high-end mobile phone, could have a significative impact in the performance of CAR applications.

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

References

1. Ahonen T (2010) TomiAhonen phone book 2010. TomiAhonen Consulting
2. Apple: iOS 4 (2011). Available at http://www.apple.com/iphone/ios4/
3. Bauset VF, Orduña JM, Morillo P (2011) Performance characterization on mobile phones for collaborative augmented reality (car) applications. In: Proc of IEEE/ACM 15th international symposium on distributed simulation and real time applications (DS-RT '11), pp 52–53
4. Billinghurst M, Kato H (1999) Real world teleconferencing. In: Proc of the conference on human factors in computing systems (CHI 99)
5. Billinghurst M, Poupyrev I, Kato H, May R (2000) Mixing realities in shared space: an augmented reality interface for collaborative computing. In: IEEE international conference on multimedia and expo (ICME 2000), vol 3, pp 1641–1644. doi:10.1109/ICME.2000.871085
6. Fiala M (2005) Artag: a fiducial marker system using digital techniques. In: Proc of IEEE conf on computer vision and pattern recognition (CVPR), vol 2, pp 590–596
7. Google: Android (2011). Available at http://www.android.com/index.html
8. Hall S, Anderson E (2009) Operating systems for mobile computing. J Comput Small Coll 25:64–71
9. Hallerer T, Feiner S, Terauchi T, Rashid G (1999) Exploring mars: developing indoor and outdoor user interfaces to a mobile augmented reality system. Comput Graph 23:779–785
10. Henrysson A, Billinghurst M, Ollila M (2005) Face to face collaborative ar on mobile phones. In: Proc of 4th international symposium on mixed and augmented reality, pp 80–89
11. Henrysson A, Ollila M (2004) Umar: ubiquitous mobile augmented reality. In: Proceedings of the 3rd international conference on mobile and ubiquitous multimedia (MUM '04), pp 41–45
12. Lee SE, Zhang Y, Fang Z, Srinivasan S, Iyer R, Newell D (2009) Accelerating mobile augmented reality on a handheld platform. In: IEEE international conference on computer design (ICCD 2009), pp 419–426. doi:10.1109/ICCD.2009.5413123
13. Loulier B (2011) Augmented reality on iphone using artoolkitplus. Available at http://www.benjaminloulier.com/
14. Loulier B (2011) Virtual reality on iphone. Available at http://www.benjaminloulier.com/articles/virtual-reality-on-iphone-code-inside
15. Mahring M, Lessig C, Bimber O (2004) Video see-through ar on consumer cell-phones. In: ISMAR '04, pp 252–253
16. Nyartoolkit: Nyartoolkit code (2011). Available at http://sourceforge.jp/projects/nyartoolkit-and/
17. Nyatla: Nyartoolkit: artoolkit class library for java/c#/android (2011). Available at http://nyatla.jp/nyartoolkit/
18. Piekarski W, Thomas BH (2002) Tinmith-hand: unified user interface technology for mobile outdoor augmented reality and indoor virtual reality
19. Schmalstieg D, Wagner D (2007) Experiences with handheld augmented reality. In: 6th IEEE and ACM international symposium on mixed and augmented reality (ISMAR 2007), pp 3–18. doi:10.1109/ISMAR.2007.4538819

20. Schmeil A, Broll W (2007) 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 (UAHCI '07). Springer, Berlin, pp 699–708
21. Thomas MHMBB (2007) Emerging technologies of augmented reality: interfaces and design. IGI Global. doi:10.4018/978-1-59904-066-0
22. Wagner D, Reitmayr G, Mulloni A, Drummond T, Schmalstieg D (2008) Pose tracking from natural features on mobile phones. In: Proceedings of the 7th IEEE/ACM international symposium on mixed and augmented reality (ISMAR '08). IEEE Comput Soc, Los Alamitos, pp 125–134