J Supercomput (2014) 70:552–563
DOI 10.1007/s11227-013-1082-9

Comparative performance evaluation of CAR systems based on mobile phones and feature tracking

Víctor Fernández · Juan M. Orduña · Pedro Morillo

Published online: 22 January 2014

Abstract Collaborative Augmented Reality (CAR) systems based on mobile phones have experienced a huge expansion last years, since the hardware features of most mobile phones provide excellent multimedia services and wireless network capabilities. In previous works, we improved the performance of large-scale CAR systems based on mobile phones that use fiducial marker tracking. However, CAR systems based on natural feature tracking have just emerged, changing the way in which Augmented Reality applications work. In this paper, we propose the performance evaluation of CAR systems based on feature tracking when using mobile phones, and their comparison with CAR systems based on fiducial marker tracking. The evaluation of the whole CAR system includes the rendering of the virtual environment with Unity3D. The purpose is to provide the reader with a reference about the performance that can be achieved with each kind of CAR system. The evaluation results of client devices show that they work faster with natural feature (commonly denoted as markerless) tracking than with fiducial marker tracking, regardless of the phone model and the operating system considered. The evaluation results of the whole CAR system show that natural feature tracking provides similar performance than fiducial marker tracking when the system reaches saturation. However, the use of natural feature tracking allows better performance for low workloads or when the system approaches saturation, since, it provides similar response times at the cost of increasing the percentage of CPU utilization in the server, instead of dropping messages. These results validate natural feature tracking as the best option for CAR systems based on mobile phones.

V. Fernández · J. M. Orduña (✉) · P. Morillo
Departamento de Informática, Universidad de Valencia, 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

Keywords Collaborative Augmented Reality · Natural feature tracking · Mobile phones

1 Introduction

Current mobile phones include full color displays, integrated cameras, fast processors and even dedicated 3D graphics chips. Due to their comprehensive hardware, these wearable devices have become an ideal platform for Collaborative Augmented Reality (CAR) systems [10,11,14].

Collaborative Augmented Reality systems are Augmented Reality systems where a user with a wearable device can collaborate with a remote user at a desktop computer [17]. Augmented Reality (AR) superimposes multimedia content—3D object, text, sound, etc—on real world through a display or screen. 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. Thus, any CAR application continuously performs a four-stage cycle. 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, one or more markers within the obtained image should be detected. Using the position of these markers, the third stage consists of drawing a 3D object on the image. Finally, in the fourth phase, some information [for example, the position(s) of the marker(s)] is sent to the other application nodes through some kind of broadcast communication.

Two methods are usually used in CAR applications for the marker detection stage: 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 [24]. This process can be grouped into three big phases: interest point detection, creation of descriptor vectors for these interest points, and comparison of vectors with the database [13]. The latter method uses fiducial markers to find a specific position in the image of the real world. This process can be divided into three phases: edge detection, rejection of quadrangles that are too large or too small, and checking against the set of known patterns [24].

On other hand, there is a wide variety of current mobile phones, with different graphic and processing capabilites, and different operating systems. 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 wide variety of mobile phone platforms 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, etc. For that reason, in previous works, we characterized the behavior of different mobile phones for AR marker tracking, and we also proposed some improvements for CAR systems based on fiducial markers and mobile phones as client devices [2,6]. However, the advent of Vuforia [18], released by the ARM-processor company Qualcomm at the end of the 2011, has allowed the widespread use of markerless-based CAR applications worldwide. The reason for its popularity is that, unlike another approaches such as NyARToolkit [15] or MetaiO

[23], the VuforiaSDK supports for more than 400 different smartphones and tablet models, and it tracks real objects on the current frame with a impressive fluidity and reliability. To test the performance of the Vuforia SDK, we made a preliminary study [5].

In this paper, we propose a comparison study of CAR systems based on mobile phones and feature tracking, including a complete application with a graphical environment, with the same application developed using fiducial marker tracking. The goal of this work is to provide the reader with a reference about the performance that can be achieved with each kind of CAR systems. To achieve this goal, we have developed some example application cases and graphical interfaces using Unity 3D [7,27]. The evaluation results of client devices show that they work faster with feature tracking than with fiducial marker tracking, regardless of the phone model and operating system considered. These results leave no space to improvements. Also, we have simulated CAR systems with client devices showing the behavior of the considered phone models. We have simulated different application examples, including the rendering of the virtual environment with Unity 3D, with up to 1,000 client devices. The characterization results of the whole CAR system show that natural feature tracking provides similar performance than fiducial marker tracking when the system reaches saturation. However, the use of natural feature tracking allows better performance for low workloads or when the system approaches saturation, since it provides similar response times at the cost of increasing the percentage of CPU utilization in the server, instead of dropping messages. These results validate natural feature tracking as the best option for CAR systems based on mobile phones.

The rest of the paper is organized as follows: Sect. 2 shows a brief description of some related work about AR applications on mobile phones. Section 3 shows some of application examples developed for the proposed characterization. Section 4 shows the characterization of some models of some popular types of client devices. Next, Sect. 5 shows the characterization of the whole system from the server point of view. Finally, Sect. 6 presents some concluding remarks.

2 Related work

Smartphones, tablets, UMPC or even lightweight backpack PCs have been widely employed as hardware platforms for the development of mobile Augmented Reality systems [16]. These systems allow users to move freely in real environments and have their hands free for input and interaction [20]. As in conventional Augmented Reality systems, the 2D/3D object tracking is a key aspect when developing mobile AR systems. In this sense, fiducial markers and predefined natural features [24,25] have been used for camera pose estimation. There are few solutions based on fiducial marker tracking over mobile phones. In 2003, ArToolKit [12], one of the most well-known software libraries for developing AR application, was released for Windows CE, and the first self-contained application was developed for mobile phones [26]. This software evolved later as the ArToolKitPlus tracking library [24]. A tracking solution for mobile phones that works with 3D color-coded marks was developed

[14], and a version of ArToolKit for Symbian OS was developed, partially based on the ArToolKitPlus source code [10].On other hand, the new software and hardware advances for mobile technologies allowed the development of markerless tracking techniques, which overcome the limitations of fiducial markers [24]. In this sense, computational complex approaches based on natural features (typically edges, corners or textures) use real object as targets (instead of invasive and artificial markers) for tracking purposes. Thus, as the computational power of recent mobile devices has increased, multi-target detection techniques and texture planar targets [25] have been included in the most important software frameworks for the development of AR mobile applications, such as Metaio [23], String AR [22] or Vuforia [18]. Although the performance characterization and optimization of mobile AR using this type of tracking have been analyzed on handheld platforms [21], to the best of our knowledge there are still no comparative studies that help the user to select which type of tracking is the most appropriate one when collaborative AR systems with many clients should be deployed.

3 Application examples

For characterization purposes, we have implemented two different application examples with the Vuforia SDK. We have developed full graphical interfaces using Unity 3D [7,27] for these applications. The first example is a 3D engine, where the CAR application consists of different user workgroups simultaneously visualizing and changing different parts of the engine. From a collaborative point of view, it is crucial that each time that a client changes something in the engine, the rest of the clients within the same workgroup properly update and display the current state of the engine. Since the view of the engine should be shared among all the users in the workgroup, but there should be different views of the engine for different workgroups, for evaluation purposes we have used this example with non-realistic engine sizes, to allow the same number of clients than in the other application example. The second example consists of a 3D warehouse, where clients simultaneously move the existing pieces of the stock. In this case, the workgroups are formed by those clients displaying the same shelves, and the warehouse can have different number of shelves, depending on the total number of clients. In this case, the relative size of the warehouse allows even hundreds of client devices displaying the same shelf. The users handle boxes, and they can move them from one shelf to another or to/from the lorry in the dock. The server displays each of these actions in an application developed with Unity 3D. Also, each time that a client performs an action with a box, the rest of the clients displaying the shelf should update their view. In both application examples, we have considered from 100 to 1,000 clients. For illustrative purposes, Fig. 1 shows a snapshot of the second application example (due to space limitations, we only include a snapshot of one of the application examples). The feature tracked is each labeled box (in the warehouse example) or the spare part to be replaced (in the case of the engine), and the percentage of success in the tracking was supposed to be 100 % , since it is the assumption that generates the highest workload to the server (the worst case).

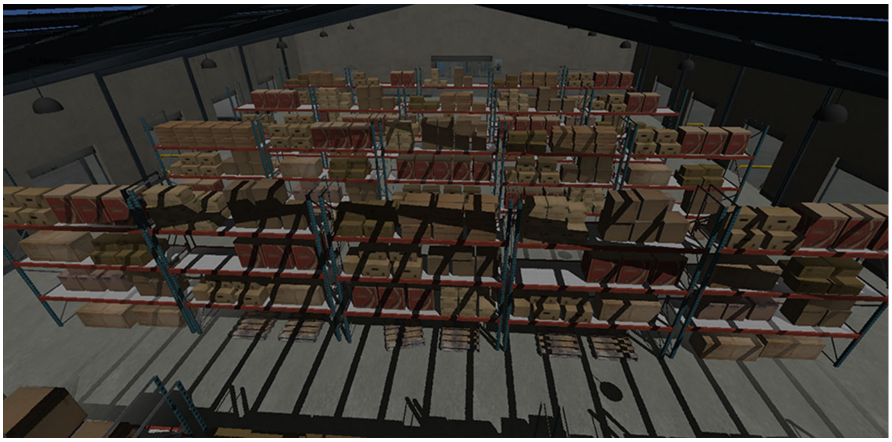

Fig. 1 A snapshot of an application example consisting of a warehouse

Table 1 Hardware features of the considered mobile phones

OS	Android		iOS	
Model	Milest.	Desire	iPh 3GS	iPh 4
CPU (MHz)	550	998	412	800
RAM (MB)	256	512	128	512
Camera (MP)	5.02	4.92	1.92	4.92

4 Clients characterization

The first step in improving the performance of CAR systems based on markerless tracking is the characterization of client mobile phones, in terms of both the time required by the system to complete each CAR stage, and the time required by each smartphone to complete a CAR cycle. Since we already performed a characterization with an implementation using fiducial markers [3,6], a performance comparison is made. For characterization purposes, the application performs the sending of the positions of the tracked markers.

We have tested two different mobile phones using Android and another two mobile devices using iOS operating system. We have selected the same phone models for comparison purposes with our previous studies for CAR systems based on fiducial markers. Table 1 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 HTC Desire, with almost double CPU frequency (998 MHz) and RAM memory (512 MB). 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). All of them include a 5 megapixels (5 MP) resolution camera, except iPhone 3GS, which equips a 2 megapixels (2MP) resolution camera.

We have used as testbed implementations the developer examples included in the Qualcomm site for both the Android and iOS [19] operating systems. These imple-

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

NyARToolKit and ARToolKit Plus					
Phases (ms)	Camera	Detection	Render	Sending	Total
Milestone	248.64	288.53	30.42	14.14	698.34
Desire	40.25	78.08	13.23	5.54	167.11
iPhone 3GS	33.29	58.07	28.26	15.42	398.21
iPhone 4	17.66	182.17	23.34	7.06	523.26

Qualcomm					
Phases (ms)	Camera	Detection	Render	Sending	Total
Milestone	19.58	3.95	2.38	15.06	86.41
Desire	9.97	18.03	1.76	8.48	56.68
iPhone 3GS	11.06	8.23	14.29	17.38	145.53
iPhone 4	8.15	6.21	17.85	9.68	105.91

mentations are very similar, in fact Qualcomm first implemented the Objective-C version, and then they used JNI to get the Android version. We added the sending step described above to these implementations, converting the AR application into a CAR application. This step involves a simple socket used to send to the server the position obtained each cycle from the real world.

To show the execution time for both CAR applications, Table 2 shows the same results when using fiducial markers (the upper part of the table, labeled with the name of the libraries used, NyARToolKit and ARToolKit Plus) and the results obtained with the markerless implementation (labeled with the name of the library used, Qualcomm).

Each row in Table 2 presents the results for a different mobile phone. The first two rows show mobile phones using Android OS (representative examples high-end and middle-end devices), and the last two rows show mobile phones executing iOS (idem). The same devices are used for markerless tracking, in the lower part of the table. The first four columns represent each phase of the CAR applications, and the last one represents the total time needed to complete a cycle. All values are represented in milliseconds (ms), and they correspond to average values. As it can be seen, the system based on markerless tracking is much faster than the system based on fiducial markers in each CAR stage, except that in sending stage, where both systems show similar times. This is due to the fact that the sending step exclusively depends on the network parameters and status. In this sense, the same network with the same configuration were used for both characterizations.

Table 2 shows that the time required for the image acquisition stage in the markerless tracking system is, at least, the half time required by the fiducial marker system (the case of iPhone 4), while for the case of iPhone 3GS it works here three times faster, from 33′29 to 11′06 ms. However, the best improvement is achieved for the Milestone, passing from the 248′64 ms required by the fiducial marker tracking to only 19′58ms required by the markerless tracking.

Regarding the second stage, the performance differences are even larger. The time required for completing this stage by the markerless implementation is at least one quarter of the time required by the fiducial markers implementation. The reason for this

Table 3 Throughput (in FPS) and RTT for each smartphone in both systems

	NyARToolKit		Qualcomm	
	FPS	RTT (ms)	FPS	RTT (ms)
Milestone	1.43	14.14	11.57	15.06
Desire	5.98	5.54	17.64	8.48
iPhone 3G	2.51	15.42	6.87	17.38
iPhone 4	1.91	7.06	9.44	9.68

behavior is difficult to find, since Qualcomm does not provide the source code of the implementation, but the compiled library. It is also worth mention that each considered device provides different image resolutions, and the size of these resolutions is very different from Android devices to iOS devices. As shown in our previous works [3], it is not the same to analyze a image of 320 $\times$ 480 pixels searching a marker that performing the same task for an image of 1280 $\times$ 720 pixels, because the last one needs too much time to be analyzed. Regarding the third stage, Table 2 shows that the markerless implementation does not improve this stage for all the considered devices, as it is the case for the iPhone 4, which requires a similar time in both implementations.

Finally, the last column represents the time (in ms) required by each device to complete a CAR cycle when using a given implementation. Comparing the two implementations, we can conclude that all devices work at least three times faster when using the markerless implementation. Therefore, the markerless implementation provides better performance not only for high-end devices but also for low or middle-end devices. One reason that can contribute to these results is that the fiducial marker tracking implementations were not designed explicitly for the current mobile phones. However, the main reason is the efficient markerless implementation, although we cannot deeply analyze its performance because it is not an open source code.

To show the practical effects of the performance achieved by both implementations, Table 3 shows the same values shown in Table 2, but expressed in performance parameters like Frames Per Second (FPS), that is, the amount of CAR cycles done in one second, and the Round Trip Time (RTT). As described before, each row represents the devices that we are using. The FPS are shown in the first column. The RTT values are represented in the second column (in ms), and they correspond to the values in the fourth stage showed in Table 2.

Table 3 shows that the slower device when using the Qualcomm implementation is the iPhone 3GS, working almost over 7 FPS. This “slowest” FPS on Qualcomm is faster than any of the performance results obtained with NyARToolKit or ARToolKit Plus, whose fastest device was the HTC Desire, with a FPS of almost 6. These results show that natural feature tracking provides better results for CAR applications executed on mobile phones than fiducial marker tracking, leaving no space for improvements.

5 CAR system performance

In the CAR server side, we have developed a multithreaded CAR server that supports simulated clients (simulated mobile devices) with the behavior measured in the characterization part, as we did for marker-based CAR systems [3,6]. We have

Table 4 System performance for the Apple iPhone 3GS smartphone

iPhone 3GS										
WG	5					25				
	RT	Dev	CPU	RT_S	% L	RT	Dev	CPU	RT_S	% L
100	6.43	3.53	19.20	3.10	0	9.25	6.42	71.63	3.82	0.82
300	9.44	7.31	29.10	4.29	0	28.53	37.83	80.53	13.03	0.94
500	8.96	12.66	42.63	4.26	0	53.52	35.90	83.90	25.16	0.92
700	12.10	20.22	59.63	5.14	0	77.87	55.10	84.57	37.50	0.98
900	8.14	20.92	72.83	3.41	0	102.51	72.44	84.73	49.56	1.03
1000	7.03	16.51	74.83	2.71	0	124.03	83.96	84.73	59.93	0.99

time-stamped every message generated within this CAR system, to measure the performance of every device. The system configuration consists of one server, and a certain number of mobile devices that are scanning the visual space of their video camera, looking for a marker that will be converted into a 3D object in their display. The action cycle performed by each client is composed of the following steps: first, it performs one new image acquisition followed by a marker detection stage. Then, the client waits until the cycle period (determined by the action frequency, a system parameter) finishes. Next, if the acknowledgments from all the neighbors have been received, then a new message with the new marker location is sent. If all the acknowledgments have not been received, then it waits for a maximum waiting threshold of 20 seconds, and then a new round of messages (with the latest marker location) are sent to the neighbors through the server. The neighbors simply return an ACK message to the sender device through the server. The server simply forwards the messages to the corresponding destination clients. We have simulated a CAR system containing client devices with that behavior. Like in the case of fiducial marker tracking [4], we have used a server based on UDP sockets. We have performed tests on the CAR system with different numbers of client devices (from 100 to 1,000), and different numbers of neighbors inside the workgroup (5, 10, 20 and 25). Due to space limitations, we will only show here the results for CAR systems using two intermediate client devices, the Apple iPhone 3GS and the HTC Desire, as representative example of mid-range mobile devices. For each device, we show the results for workgroup sizes of 5 and 25 neighbors, since they represent the lowest and highest system workload, respectively.

Table 4 shows the CAR system performance when all the client devices are iPhone 3GS smartphones (with and action cycle of 145,53 ms, according to Table 2). This table comprises two subtables, for the cases of 5 and 25 devices per working group. A working group is the subset of neighbor client devices that are collaborative working on the same task (displaying the same scene), and therefore they are the subset of neighbor devices to which the marker location should be sent in each cycle. The greater working group size, the higher number of messages should be sent and acknowledged per each cycle. To obtain comparable results with the case of fiducial marker tracking [4], the messages containing the marker location are sent through the server (that is, it sends the location update message to the server, and then the server re-sends the message to the appropriate clients). For performance evaluation purposes, the destination clients return an acknowledgment message (ACK) to the server, which, in turn, forwards it to the source client.

The first (most left) column in each subtable of Table 4 shows the number of simulated devices in the CAR system. The next five columns, from left to right, show the average system response time for all the clients (measured in ms), labeled as “RT”, and its standard deviation (labeled as “Dev”). This response time is measured as the time elapsed since the origin client device sends a new position to the server until the instant when the origin device receives all ACKs from its neighbors in the workgroup. The third column shows the percentage of CPU utilization in the system server during the simulation. The fourth column represents the server’s response time, measured as the time elapsed since the server sends a message to a client until the instant when the server receives the answer to that message. Finally, the last column represents the percentage of lost packets (this server works with UDP sockets) in regard to the total number of messages exchanged.

Table 4 shows that for a working group size of 5 devices the latency remains almost constant, around 10 ms. The standard deviation increases slightly as the population increase, from $3'53$ to almost 21 ms. The CPU consumption also increases but does not reach 75 %. The response time in the server also remains constant, and packets are not dropped. That is, when all the clients in the CAR system are iPhone 3GS terminals and the working group size is 5, then the system works under a low workload, even when supporting one thousand clients. The results obtained for a working group size of 25 devices show both longer response times and higher percentages of CPU utilization. However, the longest response time (obtained for the highest number of clients in the system, as it could be expected) is around half of the interactivity threshold value (250 ms) [9]. Nevertheless, the column labeled as “CPU” shows that the system reaches saturation, since with 600 clients the percentage of CPU utilization is very close to 85 %. Also, the column labeled as “RT_S” shows that a significant part of the total response time is due to the server response time. The reason for this behavior is that in spite of entering saturation, the system provides interactive average response times, because some messages (a very low percentage of messages) are dropped.

Table 5 shows the results for the same CAR system when all the client devices are HTC Desire mobile phones. The actuation cycle of this device is $56'68$ ms. Since it is three times faster than the Apple iPhone3 GS, it represents a higher workload for the server. As a result, Table 5 shows a similar behavior to the one shown in Table 4, except that the response times are higher.

Finally, for comparison purposes, Table 6 shows the results obtained for a marker-based CAR system (ArtToolkit Plus libraries) when using as client devices HTC Nexus One mobile phones [4]. This model of mobile phone has an action cycle of 167.11 ms, similar to the one of the iPhone 3GS when using the Vuforia SDK. The hardware features of this mobile phone are similar to the ones of the HTC Desire.

The comparison of the left half of Table 6 (marker-based server) with the left half of Table 4 (markerless server) shows that there are similar response times for small workgroup sizes (5 clients). Also, the rest of the columns in both subtables show similar values. However, the comparison of the right half of both tables shows that

Table 5 System performance for the HTC Desire smartphone

HTC Desire										
WG	5					25				
	RT	Dev	CPU	RT_S	% L	RT	Dev	CPU	RT_S	% L
100	5.96	3.27	24.77	2.70	0	11.77	6.58	70.33	4.30	0.48
300	3.62	3.34	64.93	1.68	0	29.96	21.11	79.83	13.29	0.82
500	59.71	43.59	84.33	28.59	0	59.76	55.61	84.30	26.81	0.99
700	87.02	29.87	83.77	37.39	0	85.62	45.93	83.50	36.31	0.97
900	94.89	56.81	86.10	33.73	0	129.14	108.77	83.67	24.31	0.69
1000	97.84	56.15	82.37	33.82	0	205.43	179.12	80.50	33.01	0.52

Table 6 Results for a marker-based CAR system with HTC Nexus One client devices

HTC Nexus One										
WG	5					25				
	RT	Dev	CPU	RT_S	% L	RT	Dev	CPU	RT_S	% L
100	4.80	7.06	38.40	1.95	0	9.86	6.78	72.50	4.06	0.83
300	9.57	9.74	26.00	4.44	0	26.01	21.91	79.60	11.61	0.69
500	4.59	6.41	41.60	1.81	0	48.68	39.68	83.80	22.84	0.74
700	5.28	8.24	53.00	2.11	0	79.70	97.87	85.10	37.26	0.76
900	5.85	11.69	66.30	2.61	0	93.64	66.36	83.90	40.71	0.93
1,000	7.15	15.47	69.50	2.87	0	122.37	85.35	85.00	44.98	0.90

for large workgroup sizes (25) which generate a higher workload, both systems reach saturation (percentages of CPU utilization around 85 %), they provide similar response times within the same range (half of the interactivity threshold value), and they drop similar percentages of messages.

These results show that natural feature tracking provides similar performance than fiducial marker tracking, with no significant differences between these two marker detection methods. Therefore, we can conclude that from the system point of view, natural feature tracking provides similar performance to CAR applications than fiducial marker tracking.

6 Conclusions

In this paper, we have proposed the performance evaluation of CAR systems based on natural feature tracking when using mobile phones, and its comparison with CAR systems based on fiducial marker tracking. The evaluation of client devices show that they work faster with markerless tracking than with fiducial marker tracking, regardless of the phone model and operating system considered. The evaluation results of the whole CAR system show that natural feature tracking provides similar performance than fiducial marker tracking. These results show that to take advantage of the ever-increasing computational power of current mobile phones, natural feature tracking is the best option for CAR systems based on mobile phones.

Acknowledgments This work has been jointly supported by the Spanish MICINN and the European Commission FEDER funds under grants TIN2009-14475-C04-04 and TIN2011-15734-E.

References

1. Ahonen T (2010) TomiAhonen Phone Book 2010. TomiAhonen Consulting
2. Bauset VF, Orduña JM, Morillo P (2012) On the characterization of car systems based on mobile computing. In: Proceedings of IEEE 14th International Conference on High Performance Computing and Communication (HPCC-ICESS), pp 1205–1210
3. Bauset VF, Orduña JM, Morillo P (2012) Performance characterization of mobile phones in augmented reality marker tracking. In: Proceedings of the 12th International Conference on Computational and Mathematical Methods in Science and Engineering, vol 2, CMMSE '12La Manga, Spain, pp 537–549
4. Bauset VF, Orduña JM, Morillo P (2013) How large scale car systems based on mobile phones should be implemented. In: International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, GRAPP 2013, pp 381–384
5. Bauset VF, Orduña JM, Morillo P (2013) On the characterization of markerless car systems based on mobile phones. In: Proceedings of the 13th International Conference on Computational and Mathematical Methods in Science and Engineering, vol 2, CMMSE '13Almeria, Spain, pp 618–629
6. Fernández V, Orduña JM, Morillo P (2013) How mobile phones perform in collaborative augmented reality (car) applications? J Supercomput 65:1179–1191. doi:10.1007/s11227-013-0925-8
7. Goldstone W (2011) Unity 3.x Game development essentials. Community experience distilled. Packt Publishing. http://books.google.es/books?id=RJ5fsGXbqXwC
8. Hall SP, Anderson E (2009) Operating systems for mobile computing. J Comput Small Coll 25:64–71
9. Henderson T, Bhatti S (2003) Networked games: a qos-sensitive application for qos-insensitive users? In: Proceedings of the ACM SIGCOMM 2003, pp 141–147. ACM Press / ACM SIGCOMM
10. Henrysson A, Billinghurst M, Ollila M (2005) Face to face collaborative ar on mobile phones. In: Mixed and Augmented Reality, 2005. Proceedings Fourth IEEE and ACM International Symposium on, 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 '04ACM, New York, pp 41–45
12. Kato DH (2011) Artoolkit. Available at http://www.hitl.washington.edu/artoolkit/
13. Lee SE, Zhang Y, Fang Z, Srinivasan S, Iyer R, Newell D (2009) Accelerating mobile augmented reality on a handheld platform. In: Computer Design, 2009. ICCD 2009. IEEE International Conference on, pp 419–426. doi:10.1109/ICCD.2009.5413123
14. Mahring M, Lessig C, Bimber O (2004) Video see-through ar on consumer cell-phones. In: ISMAR'04, pp 252–253
15. Nyatla (2011) Nyartoolkit:artoolkit class library for java/c#/android. Available at http://nyatla.jp/nyartoolkit/
16. Papagiannakis G, Singh G, Magnenat-Thalmann N (2008) A survey of mobile and wireless technologies for augmented reality systems. Comput Animat Virtual Worlds 19(1):3–22
17. Piekarski W, Thomas BH (2002) Tinmith-hand: Unified user interface technology for mobile outdoor augmented reality and indoor virtual reality. In: Proceedings of virtual reality, pp 287–288
18. Qualcomm: Vuforia sdk 1.5. Available at http://www.qualcomm.com/solutions/augmented-reality (2012)
19. Qualcomm: http://www.qualcomm.com/solutions/augmented-reality
20. de Sá M, Churchill E (2012) Mobile augmented reality: exploring design and prototyping techniques. In: Proceedings of the 14th international conference on human-computer interaction with mobile devices and services, MobileHCI '12, ACM, New York, pp 221–230
21. Srinivasan S, Fang Z, Iyer R, Zhang S, Espig M, Newell D, Cermak D, Wu Y, Kozintsev I, Haussecker H (2009) Performance characterization and optimization of mobile augmented reality on handheld

platforms. In: Workload characterization. IISWC 2009. IEEE International Symposium on, pp 128–137. doi:10.1109/IISWC.2009.5306788

22. String ar sdk 1.3.1. Available at http://www.poweredbystring.com (2011)
23. Ta DN, Chen WC, Gelfand N, Pulli K (2009) Surftrac: Efficient tracking and continuous object recognition using local feature descriptors. IEEE Conference on Computer Vision and Pattern Recognition, pp 2937–2944
24. 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 '08IEEE Computer Society, Washington, DC, pp 125–134
25. Wagner D, Reitmayr G, Mulloni A, Drummond T (2010) Real-time detection and tracking for augmented reality on mobile phones. IEEE Trans Vis Comput Graph 16(3):355–368. doi:10.1109/TVCG.2009.99
26. Wagner D, Schmalstieg D (2003) First steps towards handheld augmented reality. In: Proceedings of the 7th IEEE International Symposium on Wearable Computers, ISWC '03IEEE Computer Society, Washington, DC, pp 127–135
27. Wang S, Mao Z, Zeng C, Gong H, Li S, Chen B (2010) A new method of virtual reality based on unity3d. In: The 18th International Conference on Geoinformatics: GIScience in change, geoinformatics 2010, Peking University, Beijing, pp 1–5. IEEE (2010). doi:10.1109/GEOINFORMATICS.2010.5567608