# An Analysis of Android Smartphones as a Platform for Augmented Reality Games

Andrés L. Sarmiento, Margarita Amor, Emilio J. Padrón, Carlos V. Regueiro
*Dept. Electronics and Systems*
*University of A Coruña*
*A Coruña, Spain*
*andreslopezsarmiento@gmail.com, margarita.amor@udc.es, emilioj@udc.es, cvazquez@udc.es*

***Abstract*—In this work, we analyse the capabilities of an Android smartphone with the OpenGL ES API for the rendering of synthetic realistic images. The aim is to find out the facilities and the main limitations of the platform for the development of augmented reality games. Thus, our research covers mainly three fields: an analysis of the information provided by the camera, a study of the tracking and positioning capabilities of current smartphones and an outline of the rendering facilities usually found in these devices. The performance, in terms of frames per second and latency, has been tested in different smartphones, in addition to evaluate the reliability and efficiency of the sensors and the quality of rendering. In order to show all the results obtained from this study we have developed an augmented reality game trying to combine quality, performance and velocity of response.**

***Keywords*-Augmented reality; Android; Positioning sensors; Image processing; Realistic image synthesis**

## I. INTRODUCTION

Smartphones have gathered functionalities and features of an increasingly number of different devices, from those used in a more professional environment (i.e., mobile devices, electronic agendas, GPS) to others with recreational aspects (such as cameras or video game consoles). Although this means an obvious saving of money and space, the major advantage of these new devices is the integration of all those capabilities in increasingly complex and innovative applications.

Most of the operating systems available for these devices have been developed ad hoc for each model. Android [1], however, has a very different origin since it is a multi-platform linux-based OS promoted by a group of companies. This open source and cross-platform nature, together with the growth it has experienced over the past few years, made us adopt Android as the platform for this work.

Augmented reality (AR) [2] is one of the newest and most popular applications that have recently shown up within the sphere of smartphones. Most of the existing proposals may be classified at one of the following three groups: AR browsers; applications that allow us to move through a completely synthetic environment; and, lastly, applications that use the camera information to show virtual objects in the phone.

AR browsers are outdoor AR applications that do geopositioning of virtual objects in the real world by using the orientation sensors and the GPS or a triangulation positioning system to determine the position where they must be placed [3], [4]. The information about the objects to be positioned is pre-computed and these applications do not demand a great accuracy in the positioning and orientation of the mobile device.

The second type of AR applications use only the movement and orientation of the device to readjust the vision of a synthetically generated scene [5], [6]. In these applications all elements are generated in a virtual scene that is shown in the mobile screen. The image captured by the camera can also be shown, but it has not any influence in the applications as it is not processed by the device.

Finally, some applications apply artificial vision techniques [7], [8]. This type of applications processes the perceived image and uses that information to put the virtual models in the right place. Obviously, this approach means higher computational requirements and a greater application complexity. As a consequence, there are really few AR applications in Android based on exploiting data obtained by the camera and most of them are basically technical demonstrations.

In our research, we focus on this last line of work since the best approach to integrate synthetic information with the immediate real-time data from the environment in a realistic scenario such as a dynamic and complex environment seems to be the exploitation of both the camera and the positioning sensors of these devices. Since Android is a brand new platform, analysing the viability of this kind of AR application is a necessary preliminary step. This analysis is complemented in this work with the development of a simple AR game for indoor environments as a demonstration of the possibilities of this approach.

Thus, Section II goes into the study of Android smartphones as an AR platform. We have divided our analysis in three big sections: firstly, a study of the possibilities for processing the information captured by the camera; next, a survey of the positioning and tracking capabilities of these smartphones in an indoor environment and, lastly, the possibilities for real-time rendering of realistic models. A brief outline of all the aspects studied in the analysis is in the end of this section. Section III describes the AR game we have developed taking into account the results from our

 ISBN: 978-1-61208-171-7 

Table I: Technical data for the smartphones used in our tests.

| | *Motorola Milestone* | *GeeksPhone One* | *Samsung Galaxy S* |
|---|---|---|---|
| Android | 2.1 Eclair | 2.2 Froyo | 2.2 Froyo |
| CPU | ARM Cortex A8 550 MHz | ARM11 528 MHz | Samsung Hummingbird 1 GHz |
| GPU | PowerVR SGX 530 | built-in | PowerVR SGX 540 |
| Memory | 256 MB | 256 MB | 512 MB |
| Display | 3.7" 854x480 | 3.2" 400x240 | 4" 800x480 |
| GPS | ✓ | ✓ | ✓ |
| Acceler. | ✓ | ✓ | ✓ |
| Compass | ✓ | ✗ | ✓ |
| Camera | ✓ | ✓ | ✓ |

analysis, and Section IV shows the performance achieved with our proposal. Finally, the conclusions we have reached with this work are shown.

## II. Analysis of the capabilities of an Android smartphone with OpenGL ES

In this section, an analysis of the capabilities of the Android platform in the context of AR is presented. Table I shows the main features of the devices used in our study. These devices are representative of the current smartphone market.

### A. Image capture and processing

The camera of a smartphone is of great importance for AR applications, since the synthetic images are usually rendered over real images obtained by the camera. If the image from the camera is just being displayed, Android efficiently add it to the rest of layers shown by the application. Otherwise, if the image is going to be processed, it is captured by the system and provided to the application as a vector of bytes in a default format previously set in the camera. Many cameras (such as the ones used in our analysis) only work with the YUV image format.

Once an image from the camera is obtained, any image processing technique may be applied on it. Since image processing is usually a high-cost computationally task, any operation has to be spawned in a different thread to the one running the application's GUI. Otherwise, non-responding lags are probably to be experienced in the application. Besides, it is also a good practice to code image processing tasks in native code (C, C++) and use the NDK to integrate it in the application [9]. This way, we can achieve an important improvement, up to 400%, in the velocity of execution.

In order to analyse the possibilities of image capture and processing at iterative rates we started studying the maximum frequency at which data can be obtained, what allow us to get the top level of performance that can be achieved. Thus, this test captures the image and calls a naive processing image code that just computes the frame rate (fps, frames per second) with no additional computation. The results obtained for a *Motorola Milestone* with Android v2.1 and a configuration of $10\,\mathrm{fps}$ as the maximum capture frequency were $8.8\,\mathrm{fps}$.

Table II: Image capture, decoding and visualisation on *Motorola Milestone* with Android v2.1.

| Image size | FPS |
|---|---|
| 560×320 | 3.90 |
| 280×320 | 4.45 |
| 280×160 | 4.95 |
| 140×160 | 5.10 |
| 15×15 | 5.15 |

To study the effect of a simple image processing on the performance, we have extended our test by adding the display on the screen of the images obtained by the camera. Since images are obtained from the camera in YUV format and they must be in RGB to be displayed by Android, this test program takes each image captured by the camera, recodes it from YUV to RGB and gets it displayed on the screen. Additionally, our test program can be configured to encode only a region of the image. The results of running our tests in the *Motorola Milestone* are depicted in Table II. The table shows the fps rate as a function of the size of the region to process. As can be observed, a top value of $5.15\,\mathrm{fps}$ has been obtained, that does not make possible to keep a fluid stream of images on the screen. Furthermore, we have observed a delay of about one second in what is being displayed. Considering the results, we have kept the configuration of $10\,\mathrm{fps}$ as the maximum frequency for the rest tests, since it has provided the best results; probably because with this frequency the application is not saturated with images it is not able to process. Other tests adding different image processing algorithms, such as colour segmentation based on pixel colour, were carried out and similar execution times were obtained.

The obvious conclusion coming from the results of our tests is that the image processing velocity is really low in Android v2.1 and previous, obtaining a slow response even after implementing optimisations such as using NDK and running the processing in a different thread. The main reason for this performance seems to be in the process the system follows for each image captured by the camera, allocating memory, saving a copy of the image, calling the function to process it and, finally, removing the reference to the allocated memory [10]. This whole process entails a completely inefficient memory management, that is made still more acute by the high cost of garbage collection in Android, between 100 and 300 milliseconds. Not reusing the memory assigned to each image results in a frequent invocation of the garbage collector, burdening the performance.

This important issue with memory management is solved in Android v2.2, that includes other significant improvements as well, such as a *Just in Time* compiler. Regarding image processing, the API has been enhanced with new

 ISBN: 978-1-61208-171-7 

Table III: Image capture, decoding and visualisation in devices with Android v2.2.

| *GeeksPhone* | | *Galaxy S* | |
|---|---|---|---|
| Size | FPS | Size | FPS |
| 400×240 | 3.90 | 800×480 | 5.70 |
| 200×240 | 4.50 | 400×480 | 7.10 |
| 200×120 | 5.00 | 400×240 | 8.00 |
| 100×120 | 5.50 | 200×240 | 8.75 |
| 15×15 | 5.80 | 15×15 | 9.20 |

methods that work with a buffer passed as a parameter, removing the memory allocation and removal for each image to process.

We have analysed the improvements in Android v2.2 by running the same tests in two of our devices with the new version of the OS. Table III shows the results obtained with Android v2.2 for the simple capture and recoding test previously outlined in Table II. As can be observed, there is a performance increase of 50%, from 3.90 up to 5.70, and taking into account a 50% increase in the image size as well. The improvement is even more appreciable looking at the visualisation delay, that has been reduced from around 1 second to 0.5 seconds. However, bearing these results in mind, an efficiency analysis of the real world around us makes necessary the use of data from other sources, e.g., positioning sensors.

### *B. Device positioning and orientation*

In this subsection we outline the main positioning and tracking sensors included in most Android smartphones: accelerometer, compass and GPS. In order to check their performance, some test were executed on our *Milestone* phone, similar results were obtained in the rest of devices.

An accelerometer measures the proper acceleration of itself, i.e., a change in velocity, that involves a change in position. Mathematically velocity is the integral of acceleration, and position is the integral of velocity. Smartphones have usually three accelerometers, one for each spatial axis. Theoretically, the position of a smartphone could be guessed from data provided by these sensors. In practice, however, the measures are not very accurate due to the presence of gravitational and centripetal forces. Anyway, these sensors are handy for knowing the device's position relative to the floor with simple trigonometric calculations.

Figure 1 depicts the values received while a user is walking along the $Z$ axis with the mobile vertical to the floor (axis $Y$ is perpendicular to the floor and axis $X$ is on the side). As can be seen, there is a regular pattern of about a footstep per second, crests in axis $Y$. The lateral movement enclosed to each footstep can also be observed, but more complex movements would be hard to recognise.

A digital compass or magnetometer is a device that measures the strength and direction of the magnetic fields in its environment. Due to the magnetic field of the Earth, a

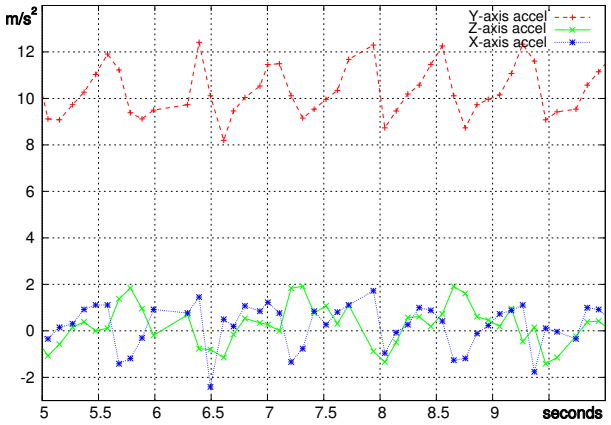


Figure 1: Values obtained by the accelerometers of a *Motorola Milestone* during a user's walk.

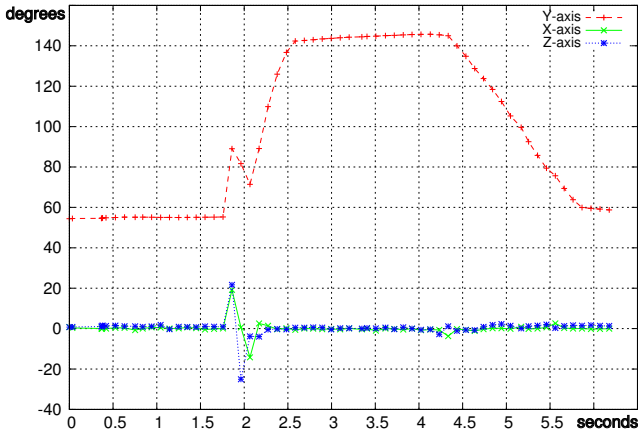


Figure 2: Values obtained by the compasses of a *Motorola Milestone* with a 90° turn.

compass is usually employed as an orientation device since it points out the Earth's magnetic north. A smartphone usually incorporates a chip that integrates three compasses arranged to detect magnetic fields in the three spatial axes [11]. Figure 2 shows the results obtained by a test consisting of making an abrupt 90° turn, almost instantaneous, just before returning to the initial position by means of a slighter turn, during about 3 seconds. As can be observed in the figure, the compass is too slow in measuring the new position after the first sudden movement, what introduces wrong values during a short period. However, it behaves really well in the presence of slight movements, with accurate values and very little noise.

Therefore, to track the direction in which a smartphone moves with Android is recommended to take together data from the accelerometers and the compasses. By previously setting a default position for the device when the application starts we get enough accuracy, since measures of smooth changes in the local environment are quite precise.

GPS is a space-based global navigation satellite system

 ISBN: 978-1-61208-171-7 

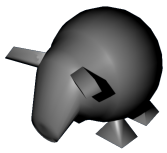

Figure 3: Test model for OpenGL ES.

that provides reliable location through an infrastructure comprising a network of satellites. This system can be used all around the world whenever there is an enough number of visible satellites. Otherwise, less accurate measurements are obtained or the device can even get out of network coverage, usual problem in the indoor locations. The values obtained by a GPS device points out its current position in the globe with a few meters of precision, about 10 meters outdoor. Besides, it doest not provide reliable information about the direction or inclination of the device and data is obtained with a delay of about 1 and 2 seconds. All this makes difficult to realistically locate and move synthetic objects that are close to the device.

Nowadays, an alternative method to GPS is network-based tracking by using the service provider's network infrastructure to identify the location of the device. Although this technique has less accuracy than GPS, it has the advantages of a shorter initialisation time and an important reduction in power consumption, since only the telephone signal is used. Additionally, it can provide better results in indoor environments than GPS. Anyway, both the two methods are compatible as they are not mutually exclusive.

### C. Android and synthetic graphics

OpenGL ES [12] is the API for producing computer graphics in Android. It is a subset of the standard OpenGL designed for embedded devices and it has important simplifications. We have carried out a group of test on the devices shown in Table I to analyse the performance of graphic synthesis in Android.

The first test focused on measuring performance as the number of primitives to render increases. The experimental results obtained for a scene with the model of Figure 3 replicated multiple times are shown in the column C1 of Table IV. In view of these results it is clear that performance gets worse as the number of polygons increases except for *Galaxy S*, device in which we perceive a serious performance loss starting from 300K points.

Column C2 of Table IV shows the results after adding a texture to the model of Figure 3. This definitely improves the visual aspect of the virtual objects with a minimum loss of efficiency, up to a 17% for a model of 75000 points in our *Milestone*. Column C3 depicts the results when including transparency effects. This hardly has influence on performance comparing to the synthesis with textures. In

Table IV: Performance of OpenGL ES in Android (fps).

| Points | GeeksPhone | | | | Milestone | | | | Galaxy S | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | C1 | C2 | C3 | C4 | C1 | C2 | C3 | C4 | C1 | C2 | C3 | C4 |
| 3 K | 35 | 35 | 35 | 33 | 30 | 30 | 30 | 30 | 55 | 55 | 55 | 55 |
| 9 K | 18 | 19 | 19 | 15 | 30 | 29 | 29 | 28 | 55 | 55 | 55 | 55 |
| 15 K | 12 | 10 | 10 | 10 | 29 | 26 | 26 | 25 | 55 | 55 | 55 | 55 |
| 30 K | 8 | - | - | - | 25 | 22 | 22 | 19 | 55 | 55 | 55 | 55 |
| 75 K | - | - | - | - | 18 | 15 | 15 | 12 | 55 | 53 | 53 | 50 |
| 100 K | - | - | - | - | - | - | - | - | 55 | 44 | 44 | 41 |

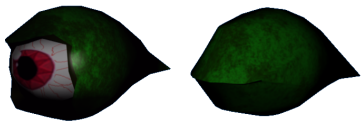

Figure 4: Morphing animation: starting state on the left and final state on the right.

column C4 the results are obtained after applying illumination to the models. The performance decreases now a 24% in *Milestone* for a scene with 30K points. Obviously, this loss of performance is due to the additional computation required to get the colour of each pixel in the scene. Furthermore, it is necessary to define the light sources in the scene, setting its position, type, colour and intensity, in addition to provide each vertex of the models with a normal vector. As can be observed, the fall of performance in *Galaxy S* is only noticeable for models with a certain complexity (100K points).

As regards animation, among all the different methods we have analysed the inclusion of morphing [13]. This technique gets a smooth transition between two models, using interpolation to compute the intermediate versions of the models. Since a new position for each point in the model has to be calculated for each frame, this kind of methods have a high computational cost. The model in Figure 4 (around 800 points and 300 polygons) has been used to test the performance of this kind of animation together with the application of textures and illumination in our target devices. The frame rates obtained for different scenes with this model are shown in Table V (only the most interesting results were measured). It can be observed that performance falls off dramatically except for low-complexity scenes (8K in the case of *Galaxy S*).

### D. Discussion

An important deficiency in the image processing capabilities of the platform has arisen, mainly in terms of image capture latency (a minimum of 0.5 seconds in high-end smartphones). The main AR applications of other platforms use the information obtained after a complex analysis of the images captured by the camera as the main source of information for positioning the synthetic objects in the

 ISBN: 978-1-61208-171-7 

Table V: Comparison of static (S) and animated (A) models in the scene (fps).

| Points | *GeeksPhone* | | *Milestone* | | *Galaxy S* | |
|---|---|---|---|---|---|---|
| | S | A | S | A | S | A |
| 800 | 40 | 21 | 30 | 30 | 55 | 55 |
| 1,6 K | 32 | 14 | 30 | 25 | 55 | 55 |
| 2,4 K | 27 | 10 | 30 | 18 | 55 | 55 |
| 4 K | 21 | 6 | 30 | 10 | 55 | 51 |
| 8 K | - | - | 27 | 5 | 55 | 29 |
| 12 K | - | - | - | - | 55 | 20 |
| 16 K | - | - | - | - | 55 | 15 |

scene. In view of the results of our analysis, this kind of applications are currently not possible at all in the Android devices we have tested.

On the other hand, multiple conclusions can be extracted from the analysis carried out using the Android positioning sensors. First of all, regarding the use of the built-in locating and tracking sensors, the accelerometers and the compass provide results relatively reliable with no important errors. However, GPS gives an excessive error in the measure to be used in the kind of AR indoor application we propose in this work.

Lastly, we have detected restrictions in size and complexity of the models to be rendered. From the results we can deduce that the graphic hardware is powerful enough to render non-excessively complex models with textures and illumination. Therefore, in the game we propose in the next section as an example of AR application, all the render capabilities we have analysed have been applied, but limiting the complexity of the model in order to get real time rendering.

## III. An AR game in an Android platform

To exploit the different aspects we have studied in our analysis we have developed a simple AR game. In this game each real-time image obtained by the camera is analysed and it determines the apparition of 'enemies' that the user/player must hunt down. Thus, we have implemented a simple system of events based on object colours and the different enemies are drawn when a certain event is triggered. These synthetic characters have to look as if they really were in the real world so they must behave properly with camera movements.

There are 4 different enemies in the game, each one of them with specific reactions and movements: *BatGhost* (Figure 5a), has been designed as an example of animation by parts, with its wings moving independently to provide a sense of flapping, *HulkGhost* (Figure 5b) with its blinking eye is an example of animation using morphing techniques, *EmGhost* (Figure 5c) was designed to have an enemy with bouncing capabilities, that could jump over the player, and *SuperGhost* (Figure 5d), that moves around the player while approaching to him.

(a) *BatGhost* (b) *HulkGhost*

(c) *EmGhost* (d) *SuperGhost*

Figure 5: Three-dimensional models.

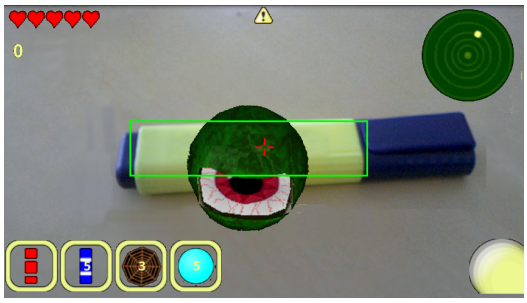

Figure 6: Event detection and triggering.

When it comes to rendering the different elements through OpenGL ES calls, the operating system itself executes these calls in a different thread, allowing a decoupled execution. Furthermore, the reuse of memory is a constant issue in our implementation, preventing the number of memory allocations as far as possible.

## IV. Experimental results

This section presents the performance achieved by our AR application. The resulting frame rates are shown in Table VI. The different columns show the frames per second for image processing (ImPr) and image synthesis (Syn) in each device. The results in rows 2 and 4 (ImPr deact.) are obtained by deactivating the image processing task once an event is triggered, as described below.

In *Motorola Milestone* the image processing rate ranges from 3.25 fps with no visible enemy to 2.75 fps when an animated (morphing) enemy appears. Besides, the image synthesis rate falls down from 15 fps to 8 fps with only an animated model in the scene.

The performance is slightly worse in *GeeksPhone One*, with a peak of 2.75 fps for image processing. As can be seen, the main performance loss is mostly noticeable in the graphic synthesis. While the stream of images obtained from the camera is being processed, the performance values of the graphic synthesis are lower than the ones for *Motorola Milestone* in about 50%.

 ISBN: 978-1-61208-171-7 

Table VI: Frame rates of the AR game.

| Test | | *Milestone* ImPr | *Milestone* Syn | *GeeksPhone* ImPr | *GeeksPhone* Syn | *Galaxy S* ImPr | *Galaxy S* Syn |
|---|---|---|---|---|---|---|---|
| Static | | 3.25 | 15 | 2.75 | 8 | 4.10 | 35 |
| | ImPr deact. | | 30 | | 21 | | 44 |
| Anim. | | 2.75 | 8 | 2.50 | 3 | 3.60 | 23 |
| | ImPr deact. | | 28 | | 17 | | 41 |

In the case of *Galaxy S* we have obtained better results, with a rate of image processing ranging from 3.6 fps to 4.1 fps along with a rate of synthesis of 35 fps for static models and 23 fps for animated, aspect in which the improvement is more appreciable.

On the other hand, the performance loss in the processing of the image has increased the delay in obtaining new images from the camera, reaching now about 1 second in our application.

As commented, once an enemy is discovered it does not keep still, it moves around the environment. To increase the frame rate and achieve a good response and fluid feeling we have stopped the image processing task while the enemy remains active in the screen. This restricts the appearance of multiple simultaneous enemies, but allow us to get an outstanding improvement in the rendering, reaching about 30 fps in *Milestone*, 21 fps in *GeeksPhone* and 44 fps in *Galaxy S*, a performance high enough to achieve an acceptable fluidity in an AR game.

## V. Conclusions

In this work we present a study of the capabilities of current smartphones in the context of AR applications. Thus, to test the feasibility of this kind of applications we start checking the main constraints in the obtaining of data from the device's camera. The maximum frame rate we can obtained is less than 6 fps. The main limitation is the latency in the image capture, near to 0.5 seconds in the best case.

Another point in our study has been to analyse the locating and tracking features of these devices. From our tests we have concluded that to obtain the device orientation is relatively simple and reliable. Nevertheless, to guess the device displacement is really complicated. Calculating it using the values obtained by the accelerometers is not very reliable, due to the numerical errors in the computation of the double integration. Additionally, geolocation systems have a margin of error too high for our requirements, about 10 meters.

With regard to the rendering of synthetic images with the OpenGL ES library, we have tested the inclusion of textures, illumination and transparencies. The performance achieved in scenes with up to 15K points has been acceptable for a mid-range smartphone as *Motorola Milestone*. Adding models with morphing animation means a loss higher than 20% each time the number of points is doubled.

As a proof of concept, to show the possibilities within the AR field of the different smartphones we have analysed, an interactive AR video game has been implemented. The performance we have achieved in this application is 3.25 images obtained through the camera per second and 28 fps in the synthesis of graphics in a mid-high end smartphone as *Motorola Milestone*. The results are better in a more powerful device as *Samsung Galaxy S*, 4.1 processed images per second and 35 fps, and appreciably worse in a low-end device as *GeeksPhone One*, 2.75 processed images per second and only 8 fps.

## Acknowledgements

This work was partially supported by the Ministry of Education and Science of Spain under the contracts MEC TIN 2010-16735 and TIN 2009-07737 and also supported by the Xunta de Galicia under the contracts 08TIC001206PR, INCITE08PXIB105161PR and INCITE08PXIB262202PR.

## References

[1] M. Gargenta, *Learning Android*. O'Reilly, 2011.

[2] D. Wagner and D. Schmalstieg, "Making augmented reality practical on mobile phones," *IEEE Computer Graphics and Applications*, vol. 29, no. 3, pp. 12–15, 2009.

[3] Layar, "Layar reality browser," http://www.layar.com, last access: 05/01/2011.

[4] T. Langlotz, C. Degendorfer, A. Mulloni, G. Schall, G. Reitmayr, and D. Schmalstieg, "Robust detection and tracking of annotations for outdoor augmented reality browsing," *Computer & Graphics*, vol. 35, no. 4, pp. 831–840, 2011.

[5] MADfirm, "Sky siege," http://madfirm.com, last access: 14/01/2011.

[6] Quest-Com, "Droidshooting," http://www.quest-com.co.jp, last access: 14/01/2011.

[7] H. Kato, "Artoolkit," http://www.hitl.washington.edu/artoolkit, Android port by A. Igarashi (2010), last access: 05/01/2011.

[8] N. SL., "Invizimals," http://www.invizimals.com, last access: 05/01/2011.

[9] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," in *Int. Conf. on Control, Automation and Systems*, 2010, pp. 1160–1163.

[10] "Android Google Code. Issue 2794: Camera preview callback memory issue," http://code.google.com/p/android/issues/detail?id=2794, last access: 10/01/2011.

[11] "Asahi Kasei Microdevices. 3-axis electronic compass," http://www.asahi-kasei.co.jp/akm/en/index.html, last access: 10/01/2011.

[12] D. Astle and D. Durnil, *OpenGL ES Game Development*. Thomson Course Technology, 2004.

[13] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*. A. K. Peters, 2008.

 ISBN: 978-1-61208-171-7 