Augmented Reality with Large 3D Models on a PDA – Implementation, Performance and Use Experiences

Wouter Pasman[2], Charles Woodward[1], Mika Hakkarainen[1], Petri Honkamaa[1], Jouko Hyväkkä[1]

[1]VTT Technical Research Centre of Finland, Tekniikantie 4 B, Espoo, Finland, Charles.Woodward@vtt.fi
[2]Delft University of Technology, Mekelweg 4, Delft, The Netherlands, W.Pasman@twi.tudelft.nl

ABSTRACT

We have implemented a client/server system for running augmented reality applications on a PDA device. The system incorporates various data compression methods to make it run as fast as possible on a wide range of communication networks, from mobile phone links to WLAN. We provide a detailed description of various implementation issues, data compression optimisations, and performance analysis of the system. Usability of the system is evaluated in a demanding architectural application, with a large virtual building rendered in an outdoors location.

Keywords: Augmented reality, 3D, mobile computing, PDA

1. INTRODUCTION

Augmented reality (*AR*) is a technique to show objects that exist only in a computer (virtual objects) as if they were in the real world. Architects have been using AR-like techniques for years, to show how a planned building will look in its environment before it is actually built. Traditionally a lot of handwork was involved in this, but advances in computer graphics now make it possible to apply these techniques in real time.

The most popular display solution for AR is to overlay the virtual objects on a camera image of the real world, and show the resulting video overlay to the user on a head mounted display (*HMD*). The usual hardware configuration for AR on a mobile system is then to use a fast laptop with 3D hardware accelerator, and put it in a backpack. Mobile processing on a laptop enables fast refresh times, but the physical burden is not very inviting for the potential user. Also, the blocking of natural vision with video-see-through HMDs hinders perception of the real world and the task being done there. In case of a system failure it would effectively blind the user which makes the system potentially dangerous to use. Optical see-through HMDs are more user friendly, but they are much more critical to system latencies, and (unless being very bulky, c.f. [1]) they can only render virtual objects partly transparent.

Instead of HMD, handheld displays provide an alternative way to present mobile augmented reality to the user. Today's PDA devices provide some elementary computing resources for mobile processing, and they are of course easier to carry around than backpack PCs. As display devices, PDAs are much nicer to handle than HMDs, and for many applications a handheld display is even more useful than HMD. For instance, it can be viewed by multiple users, and the screen can be frozen to study and discuss details of a certain view.

However, there are a number of challenges that have to be resolved in order to apply AR techniques on a PDA (or on a mobile phone). First, the processing power available is still quite limited, and especially camera-based position tracking is very problematic to do in real time on the handheld. Second, there is no 3D hardware support available, so any rendering has to be done in software. If the scene is to be rendered within a few seconds in the PDA, the scene may not contain more than about ten thousand polygons, and less than a thousand for real time interaction.

2. RELATED WORK

The topic of mobile AR on handheld devices has gained increasing attention recently. Matsuoka, Onozawa and Hosoya [2] used a tablet PC to overlay small virtual objects such as a teapot on the real scene. Their work focused on realistic rendering, not accounting for performance issues. Closer related to our work, Wagner and Schmalstieg [3] have presented a stand-alone AR system running on a PDA device. Their system makes efficient use of their in-house developed SoftGL rendering system, but still there are obvious restrictions on rendering 3D models of even moderate complexity.

Alternatively, part of the AR computation can be offloaded to a fast remote computer. An early implementation of a client/server system that could be extended for AR was shown in the InfoPad project [4], using a cellular radio network to communicate with the server. Pasman and Jansen [5] analysed the client/server AR scenario to determine how the scene has to be compressed in order to minimise network and CPU load,

given a certain quality target. However, their scenario assumes 3D rendering hardware support available in the client, which is not the case for our current situation.

Client/server systems for the PDA device have previously been presented by Newman, Ingram and Hopper [6] and by the AR-PDA project [7, 8, 9]. The former implementation primarily deals with just simple 2D augmented objects. The AR-PDA project's approach is more similar to our work [10], but they transmit full video streams (or JPEG compressed RGB images) both ways over the wireless link, for which they rely on WLAN to provide sufficient speed.

In general, previous literature on augmented reality PDA systems contains very little (if any) implementation details or performance figures. We present here an in-depth discussion of some critical hardware and software architecture issues, as well as a performance analysis explaining the main bottlenecks. We put special emphasis on the video compression aspects, so that the system could be used not just with WLAN but even with mobile phone links.

To create a serious challenge for the system, we chose an architecture application where a large building is projected at an outdoors location. Besides outdoors usability evaluation, such a large range situation is also particularly challenging for the tracking system, which is a critical part for mobile AR. We conclude the discussion with remarks on markerless tracking and mobile phone implementation with the client/server AR architecture.

3. ARCHITECTURE

We start with an overview of our system components. The software is implemented in C/C++. While detail explanations of each of the software components are given in the following sections, Figure 1 shows the general software architecture.

First, the camera in the client captures an RGB image. The original image is stored away for later use, and a copy of it is thresholded to a binary value bitmap. The bitmap is run length encoded (RLE) and transmitted to the server. In the server, we use the ARToolkit software to track markers from the thresholded image and to render the virtual objects. The image containing the virtual objects is encoded using our MVQ video compression technology and sent to the client. A transparency mask is made to distinguish foreground and background in the overlay. This mask is RLE encoded and also sent to the client. In the client the overlay and mask are decoded, recombined, and overlaid on top of the original camera image.

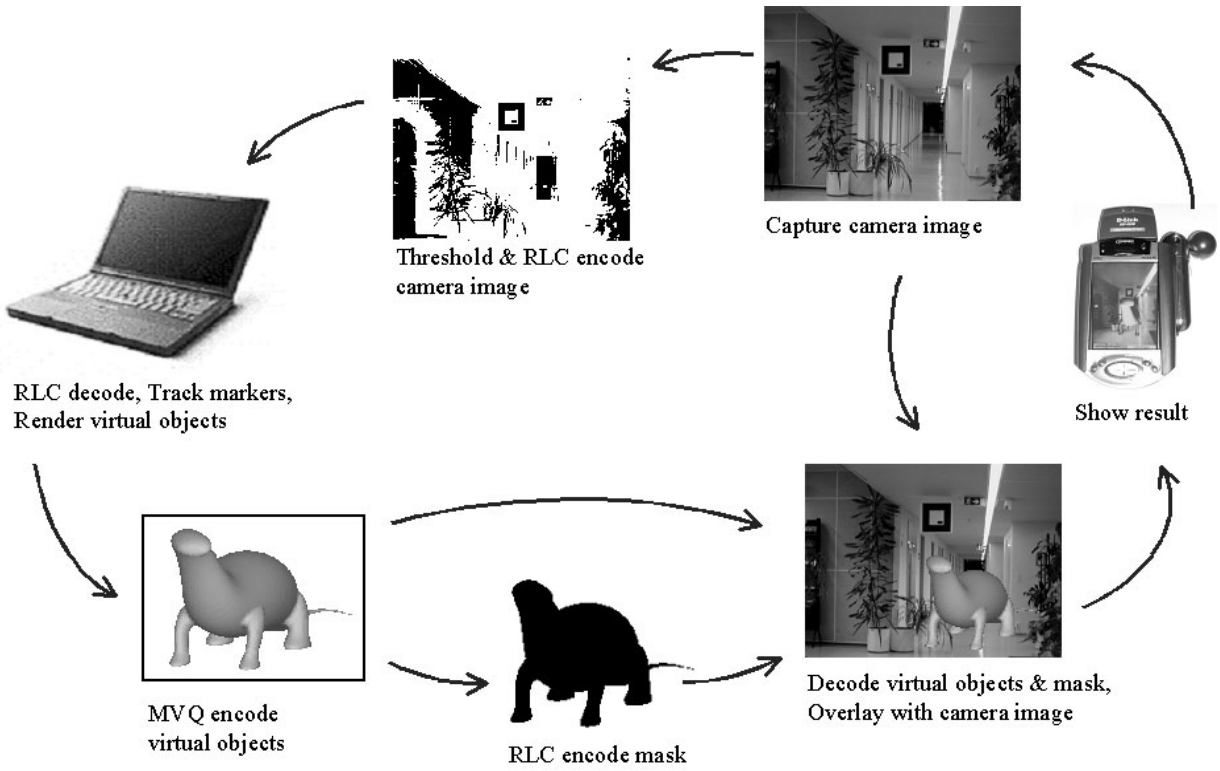

Figure 1. Software architecture.

Figure 2. Fully equipped iPAQ in the outdoors demo (composite image). The camera is rotated physically to match the display used in landscape mode.

We implemented the system on the following hardware. The client (see Figure 2) is a Compaq iPAQ H3800 series, a LiveView FlyJacket with LiveView FlyJacket iCAM, and a D-Link DCF-650W WLAN card. When using GSM, the iPAQ communicates via Bluetooth to a Nokia 7650 mobile phone, which in its turn contacts the server. The client runs on Windows CE for Pocket PC 2002.

We used two different servers, one for WLAN and USB service, and one for GSM service. The WLAN/USB server is a Dell Latitude C840 with 1.8 GHz Pentium 4, GeForce4 440 Go graphics accelerator and D-Link DWL-660 WLAN card. The GSM server runs on a nameless 800 MHz Pentium III with Matrox Millenium G400 graphics accelerator. Both servers run Windows 2000 Professional.

WLAN has a maximum bandwidth of 11Mbit/s. Typically about half of this is available for the raw data to be transmitted. Our GSM HSCSD link theoretically gives 57.6 kbit/s but this can change with the provider load. During our tests we actually measured and used 41 kbit/s.

4. TRACKING AND RENDERING

In order to overlay virtual objects over the user's view, we must determine the position of the camera attached to the viewer's mobile, and render the virtual objects accordingly. For this purpose, we decided to use ARToolkit version DirectShow2.52.vrml [11]. ARToolkit is a popular toolkit integrating tracking and OpenGL rendering, and it also includes VRML support to define the 3D CAD models.

ARToolkit uses special markers in the scene for determining the relation between the camera and virtual objects' coordinate systems. We use ARToolkit's multimarker system, which enables recovery of the camera position also when not all available markers are visible in the camera image. This is especially important for outdoors architecture applications, where the typically large areas involved can not be spanned with a single marker.

For the 60 meter wide building we were using, viewing distances in the order of 60 meters were very normal, and to get a manageable number of markers the system has to be able to track markers at least from 10 meters distance. Following marker sizes as recommended in the ARToolkit documentation [12], this would require markers of 1.5 meters wide. Our system was significantly better in practice, we found that markers of 76 cm can just be tracked at 10 meters distance.

5. USER INTERFACE

When launching the server program the setup window (Figure 3a) appears, allowing selection of a VRML scene and selection of the marker configuration. The thresholding window (Figure 3b) appears on the server after pressing Start.

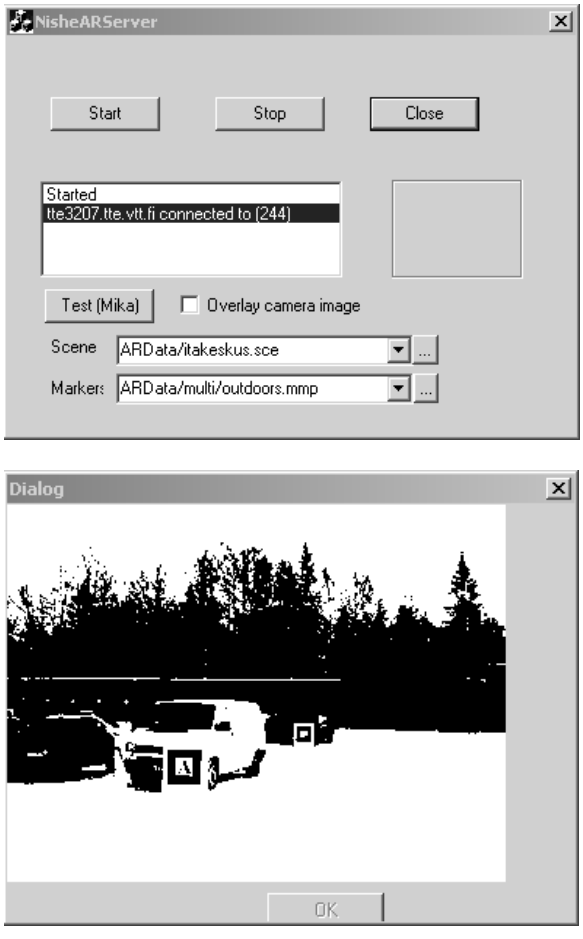

Figure 3. User interface of the server program. Above (a) the setup window and below (b) the thresholded camera image with a marker.

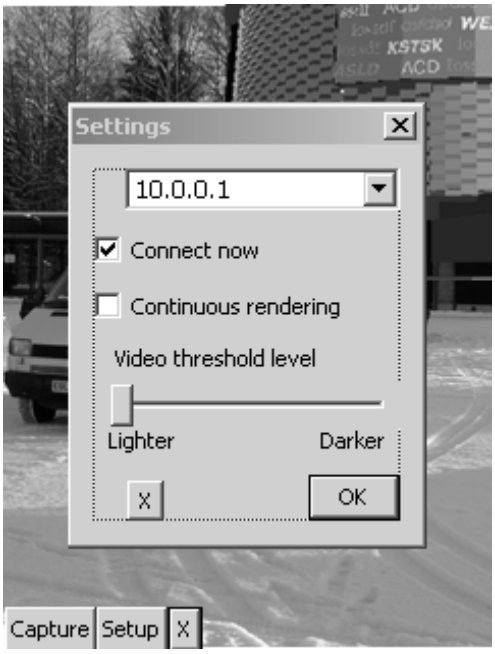

Figure 4. User interface of the client program.

The client user interface was kept minimal in order to maximise the utility of the 240x320 iPAQ screen (see Figure 2). Note that the scene is automatically shown in portrait or landscape mode just by rotating the iPAQ in the hand (c.f., Figures 2 and 4).

The image is refreshed either manually by pressing the 'Capture' button, or automatically if 'Continuous rendering' is selected in the settings window (Figure 4). The 'Combo box' on the top is to select the server IP address. A connection with the server is created after clicking 'Connect now'. The camera threshold level is set with the slider.

6. IMPLEMENTATION ISSUES

To create a single event loop in the server program, dealing both with OpenGL rendering and MS Windows button and window events, we used a modified version of glut-3.7.6 [13]. Care had to be taken that the OpenGL calls are made from the same thread as where OpenGL was initialised. Therefore, we used messages between threads to trigger the rendering after the thresholded camera image was received from the client.

Accurate camera calibration is important because in our application the markers and the virtual building can be tens of meters apart and because we want the building to stay in place no matter which marker is tracked. The calib_camera2 utility of ARToolKit worked only with live camera images, but now we had the camera attached to the iPAQ and the calib_camera2 utility would not run there. Thus we changed calib_camera2 so that it could read image files from disk, and we captured calibration images using the standard grabber tool coming with the FlyJacket.

7. DATA COMPRESSION

As the measurements in the Performance section show, a straightforward client/server AR implementation would spend most of the time on data transmission. Therefore we applied a number of compression methods to the transmitted images.

ARToolkit internally thresholds the camera image for tracking purposes. To exploit this, we send a thresholded bitmap image to the server instead of an RGB image. Note that this is also more accurate than thresholding a lossy compressed RGB image on the server.

Thresholding compresses the camera image size with a factor 24 (1 bit instead of 24 bits per pixel). On top of this, we applied run length encoding to the thresholded image. Each run is encoded with a variable length Elias Gamma code [14] which gives a practical compression factor of around 5 for practical images with not too much noise. It might be possible to get slightly more compression using other coding schemes such as Golomb-3 or Golomb-6 as these compress especially short runs a little better.

The image part of the overlay is compressed in the server before sending it to the client. For this we use Motion Vector Quantization (MVQ), a video coder developed by our research group at VTT [15, 16]. MVQ is highly asymmetrical, decompression takes much less CPU cycles than compression, which suits well our application on handheld processing units. Decoding with MVQ is extremely light, as it mainly relies on just lookup tables and motion vectors, not using cosine transformations for instance. MVQ can detect and manage motion vectors up to 64 pixels. This is much more than conventional video coders, and it is especially appropriate for shaky camera motion and low frame rates typical for our application. The specifics and performance of MVQ are further discussed in the sections below.

Using a color for indicating transparency would be tricky, as MVQ is a lossy coder and colors will be changed and blurred out a bit. Therefore we transmit a separate transparency mask. The transparency mask is a 320x240 binary image where 0 means transparent and 1 means opaque. The overlay is compressed with Elias Gamma run length encoding. As those images contain less noise than camera images, practical compression ratios are around 9. The mask will also correct contours that may have been blurred out by the MVQ coder, making the images look quite sharp even when the MVQ-encoded overlay image has low quality.

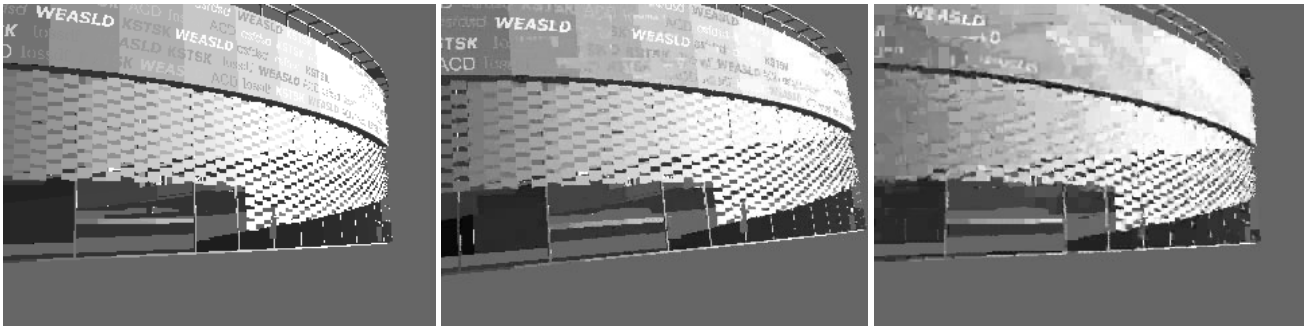

Figure 5. Impression of image qualities of combined MVQ coder and masking: original image (left), SNR= 15dB (middle) and SNR=10dB (right).

8. ADAPTING MVQ CODING MODES

Originally we had a choice of two basic settings for the MVQ coder. The first option (Offline) is optimised for offline coding, evaluating all available compression modes to find the best one for given bandwith. In particular it tries about eight encoding modes that seem most suited for sharp images at low framerates, but the compression modes are quite CPU intensive to test. The second option (Online) is optimised for near-realtime compression of natural images. It tries only a subset of the encoding modes in a certain order, and stops once a satisfying compression mode is found.

But instead of natural images we now have synthetic images, and the order of trying various compression schemes should be altered for optimal compression. Especially large smoothly shaded areas are not very likely to be found in real images but are essential to detect in synthetic images. We picked what we thought the most important compression modes to get a nice balance between coding speed and image quality. This gave a third setting for the MVQ coder (Synthetic).

Table 1. Performance of MVQ coder using various settings and target sizes.

Quality setting / compressed size	MVQ setting	Encoding (ms)	SNR (dB)	PSNR (dB)
Modem / 4kB	Offline	510	10.8	23.5
Modem / 4 kB	Online	160	9.8	22.7
Modem / 4 kB	Synthetic	200	10.1	22.8
WLAN / 30 kB	Offline	510	15.3	28.2
WLAN / 30 kB	Online	160	15.2	28.0
WLAN / 30 kB	Synthetic	200	15.3	28.2

Table 1 shows the performance of the various settings. Because we are pressing the MVQ coder really hard we find that the encoding time is independent of the target compressed size. The (peak) signal-to-noise ratio (P)SNR values were based on the luminance values of the image rendered in the backbone and the final reconstructed image on the iPAQ screen. Figure 5 shows the image quality for different SNR values.

As can be seen the synthetic coder is quite succesful, giving a nice compromise between compression speed and quality if just one solution is used for different bandwidths. But when high compression rates for transmission over modem are needed, the Offline setting gives still better quality, and the standard Online coder is a little bit faster when more bandwidth is available.

9. PERFORMANCE

Table 2 shows some of the measurements we did on our system. Different implementation phases are presented in order to appreciate how each employed method affected the performance. Communication overhead could not exactly be measured because we don't have synchronised clocks in client and server. Instead we estimated it from the difference between the time spent in the server and the time waiting on the server in the client. The estimates are consistent with the times measured manually with a wrist watch.

In Version 1, raw RGB images were sent by the server to the client, taking a lot of time to transmit. The backbone is so fast that most operations such as rotation, scaling, thresholding etc. take less than 0.5 ms to complete. Therefore we grouped them together in the two main steps "Tracking + Rendering" and "glReadPixels". At this time we still used a simple VRML scene filling only a part of the screen and using just a few hundred polygons. We scaled the camera image twice, which proved quite inefficient at the time we had settled for a 320x240 image both for tracking and display. Furthermore the scaling function was very general purpose with pixel interpolation, adding to the inefficiency. But most of the time goes into waiting for the 320x240 RGBA image to be transferred over the USB. Note that the system would have been even twice as slow if we had left out the thresholding of the camera image.

In Version 2 we separated the RGBA image into an RGB image and a binary mask. The mask only takes 320x240

bits, or just 1 kB after run length coding. The resulting speedup was as expected, and the overlaying of virtual objects using the mask was also sped up with 25% (less memory access). We also put the MVQ coder in place. We used the full-quality "Offline" coder (see above), and therefore encoding was pretty slow.

For Version 3, we figured out some problems with the flyjacket grabber [17]. For this reason and also for better speed, we decided to grab at 320x240 instead of having to downscale a 640x480 camera image for display. We switched to the "Online" MVQ coder which had less static image quality, but much faster encoding. We also integrated the camera rotation and mask decoding into a single step. We started using the large building with 60,000 polygons and occupying most of the 240x320 PDA screen (as in Fig. 5), resulting in longer rendering times. Altogether, this gave over a two times speedup.

Table 2. Timing of system performance. All times are in milliseconds. Communication overhead could not exactly be measured. (*) calculated given the estimated bandwidth (800 kB/s for WLAN, 180 kB/s for USB, 5.1 kB/s for GSM).

Step	V1 USB	V2 USB	V3 USB	V4 WLAN	V4 GSM
Capture camera image	300	300	110	110	110
Scale cam image for tracking	520	520	-	-	-
Camera img threshold	10	10	10	10	10
Transmission to server (*)	60	60	60	2	400
Tracking + Rendering	40	40	80	150	220
glReadPixels	20	20	20	20	110
MVQ compress	-	500	150	200	200
Comm overhead	350	350	350	100	2500
Transmission to client (*)	1700	135	135	25	1280
MVQ decompress	-	50	50	50	50
Scale cam image to for display	520	520	-	-	-
Rotate to landscape, Overlay virtual objects	60	60	60	60	60
Show result on screen	60	60	60	60	60
Total time per refresh	3600	2600	1100	800	5000

In Version 4 we optimised the MVQ settings to produce sharper images with the "Synthetic" compression. We switched to WLAN, which is much faster and has lower communication overhead costs, but part of the gain was consumed by slower MVQ compression and longer tracking/rendering times. It is not clear why tracking and rendering was so much slower in this situation. Anyhow, the achieved refresh rate was 800 ms per image. If the virtual object would occupy a smaller part of the screen, this would go down to some 600 ms.

Performing parts of the computation in parallel might improve the overall performance, but not more than some 30 % as the PDA is already being quite busy most of the time.

Comparisons with other implementations, e.g., [7], are difficult to make due to hardware differences, such as discussed in [19]. Nevertheless, slow transmission channels such as GSM or UMTS would in any case make comparisons favourable to our architecture, where video images are transmitted only from server to client.

Using Version 4 with the slow GSM server, the communication overhead is the main bottleneck, taking half of the time alone. Rest of the time is mainly used for transmission of the images (altogether some 8 kB). The achieved refresh time varied between 3 to 5 seconds, depending on the virtual model's screen size.

10. FIELD TESTS, USABILITY

Here we describe some field tests for evaluating the usability of the system. To make things as difficult as possible, we had decided to focus on an outdoors architecture application. For the test case it was then natural to take VTT Information Technology's new "Digitalo" building, to be built in 2004-2005 on a parking lot next to our current offices.

Before we had the 3D model of "Digitalo" available, we simulated the situation using a model of the "Itäkeskus" building, part of the main shopping centre in Helsinki. The virtual building (shown in Figures 2, 4 and 5) is about 60 meters wide and 15 meters high, and it contains about 60.000 polygons.

We opted for using two markers of 76 cm wide and high, placed on the ground at a distance of 40 and 50 meters in front of the building. In this configuration the viewer can not get close to the building, but even from 60 meters viewing distance the building does not fit on the iPAQ screen.

As could be expected, the tracking system had problems as the virtual object was located so far from the markers. The distant virtual building typically showed some jitter and inaccuracy in its placement, in particular when viewing the markers from almost straight ahead. Due to small perspective distortion, ARToolKit had then trouble in derminining whether the marker was tilted left or right away from the camera, causing the virtual building to suddenly flip long distances horizontally.

Using WLAN, the performance was quite acceptable with more than a frame per second. For GSM the framerate is a bit slow, thus the typical way of operating the GSM version would be using the snapshot mode instead of continuous video. Anyhow, the radio link solution seems attractive because the server can stay at the architects' office while viewing on site.

When operating in bright daylight, it turned out that the display of the iPAQ is quite reflective and not luminant enough. On more cloudy days it is about OK. Manual adjustment of the threshold was acceptable when having the WLAN laptop on site, but an automatic thresholding method [18] would be better for future implementations.

A second round of field tests was conducted once we had the 3D model of the real "Digitalo" building available. We had then learned from the earlier experience and obtained very stable tracking results by having the markers tilted away from the viewing path. A video of such an experiment is shown in the "Demos" page at http://www.vtt.fi/multimedia/.

The PDA visualisation of "Digitalo" was also presented to architects and other related people in charge of the construction project. While alternative Virtual Reality presentations (using VTT's proprietary CAVE system) provided much more detail information than the PDA augmentation, the feeling of "presence" by seeing the building rendered at the actual site was very much appreciated by the audience.

Overall, our field tests helped to understand and underlined the difficulties of outdoors AR applications. We also did some indoors tests where smaller virtual objects were augmented at marker locations, and there the tracking and the display worked fine. Figure 6 shows an example of such a test case.

Figure 6: Sofa augmented in middle of real furniture.

11. CONCLUSIONS AND FUTURE WORK

We have presented a client/server architecture and its implementation for running demanding mobile AR applications on a PDA device. The system is practically independent of the augmented 3D model size, and it incorporates various data compression methods to make the system generally applicable on a wide range of communication bandwidths.

The speed of the system can be considered quite acceptable on a handheld for architecture visualisation work, but it is way too large for consumer applications like games. In our current system there are no very weak performing components anymore, so there seem to be only two ways left to reduce the latency: (1) improving every component, using for instance a much faster CPU both in client and server (2) doing some client-side approximations of the image while waiting for updates from the server.

It is clear that lot of further work still has to be done on mobile AR. A particular challenge in outdoors applications is to prevent near-by real objects being occluded by the virtual objects. Also, realism of the overlays should be improved, for example with virtual light sources placed at real-world light locations. Technical solutions to these issues already exist but they have to be adapted to optimally suit mobile AR.

We mentioned various issues concerning tracking, such as the operating range and jitter. We recognise that our current marker setup is far from optimal. Better accuracy with multiple markers could be reached with a more careful implementation, e.g. using more markers and taking into accont information from all visible markers simultaneously. We could also use larger markers closer to the virtual building location, but the marker size can get quite unpractical in large range tracking, and occlusion of the markers will be a problem as the distance increases.

Eventually, it seems a markerless tracking system, c.f., [14], would be a preferrable solution for reliable large range tracking. Instead of thresholded images we might then need to send the original RGB images to the server. For architect applications, this is still quite workable in the WLAN configuration. Also, some time overhead for the markerless tracking system might be acceptable. For example, allowing some 0.2 seconds for tracking would still be in line with the currently achieved refresh rate.

The latest generation of mobile phones has exactly the technical capabilities of the PDA we used: same processor speed, same display resolution and an even better camera. Oplayo [19] has already ported the MVQ coder to mobile phones, and made it into a commercial product. Concluding, all techniques are in place to have AR also on mobile phones in the very near future.

REFERENCES

1. Kiyokawa, K., Billinghurst, M., Campbell, B., Woods, E., "An Occlusion-Capable Optical See-Through Head Mount Display for Supporting Co-located Collaboration", in *Proc. The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo, Japan, October 2003.
2. Matsuoka, H., Onozawa, A., Hosoya, E., "Environment Mapping for Objects in the Real World: A Trial Using ARToolkit", in *Proc. First IEEE Intl. Augmented Reality Toolkit Workshop (ART02)*, Darmstadt, Germany, September 2002.
3. Wagner, D., Schmalstieg, D., "ARToolKit on the PocketPC Platform", in *Proc. Second IEEE Intl. Augmented Reality Toolkit Workshop (ART032)*, Tokyo, Japan, October 2003
4. Seshan, S., Le, M. T., Burghardt, F., Rabaey, J., "Software Architecture of the Infopad System", *Mobidata Workshop on Mobile and Wireless Information Systems*, New Brunswick, NJ, November 1994. Available Internet: http://www-2.cs.cmu.edu/~srini.
5. Pasman, W., Jansen, F. W., "Comparing simplification and image-based techniques for 3D client-server rendering systems", *IEEE Transactions on Visualization and Computer Graphics 9 (2)*, pp. 226-240, 2003.
6. Newman, J., Ingram, D., Hopper, A., "Augmented Reality in a Wide Area Sentient Environment", in *Proc. 2nd International Symposium on Augmented Reality (ISAR01)*, New York, October 2001.
7. Geiger C., Kleinjohann B., Reimann C., Stichling D., "Mobile AR4All", in *Proc. The Second IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, New York, October 2001.
8. Raczynski, A., Reimann, C., Gussmann, P., Matysczok, C., Grow, A., Waldemar, R, "Augmented Reality @ Siemens: The Workflow Designer Project & Augmented Reality PDA", in *Proc. First IEEE Intl. Augmented Reality Toolkit Workshop (ART02)*, Darmstadt, Germany, September 2002.
9. Beier, D., Billert, R., Bruederlin, B., Stichling, D., Kleinjohann, B., "Marker-less Vision Based Tracking for Mobile Augmented Reality", in *Proc. The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo, Japan, October 2003.
10. Pasman, W., Woodward, C. "Implementation of an Augmented Reality System on a PDA", in *Proc. The Second IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'03)*, Tokyo, Japan, October 2003.
11. Billinghurst, I., "Shared Space/ARToolKit Download Page". Available Internet: http://www.hitl.washington.edu/artoolkit.
12. Kato, H., Billinghurst, M., Poupyrev, I., "ARToolKit Version 2.33". Available Internet: http://www.hitl.washington.edu/artoolkit.
13. Baker, S., "Hacking GLUT to Eliminate the glutMainLoop() Problem". Available Internet: http://sjbaker.org/steve/software/glut_hack.html.
14. Soboroff, I., "Compression for IR: Lecture 5". Available Internet: http://www.csee.umbc.edu/~ian/irF02/lectures/05Compression-for-IR.pdf.
15. Valli, S., "Video Coding", VTT Information Technology, Finland. Available Internet: http://www.vtt.fi/multimedia/index_vc.html.
16. Virtamo, J., Valli, S., "Method for Image Compression Coding in an Image Transmission System", *United States Patent*, Patent Number 5,692,012, (1997). Available Internet: http://www.uspto.gov.
17. Pasman, W. (2002), "Speed of FlyJacket Grabber", *Internal Report*, VTT Information Technology, Helsinki, 2002. Available Internet: http://graphics.tudelft.nl/~wouter.
18. Pintaric, T., "An Adaptive Thresholding Algorithm for the Augmented Reality Toolkit", in *Proc. Second IEEE Intl. Augmented Reality Toolkit Workshop (ART03)*, Tokyo, Japan, November 2003.
19. See Internet http://www.oplayo.com/products/oplayer.