Implementation of an Augmented Reality System on a PDA

Wouter Pasman
Delft University of Technology,
Faculty of Information Tech. and Systems
W.Pasman@twi.tudelft.nl

Charles Woodward
Technical Research Centre of Finland,
VTT Information Technology
Charles.Woodward@vtt.fi

Abstract

We present a client/server implementation for running demanding mobile AR 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 GSM to WLAN.

1. Introduction

Handheld displays provide an attractive way to present mobile augmented reality to the user. In particular, PDA devices are much easier to carry around than backpack PCs, and also nicer to handle than HMDs. 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.

Figure 1. Fully equipped iPAQ with the test model (an outdoors AR demonstration).

Previously only a few attempts have been made to implement mobile AR on handheld devices, c.f. [1, 2]. There are a number of challenges to be resolved, such as limited processing power, and lack of 3D hardware support. Many of such problems are overcome with the client/server AR implementation presented here. While the PDA is used primarily as the display device, tracking and rendering is done on the server. This makes the system practically independent of the 3D model size. Furthermore, we wanted the solution to scale such that it can be used not just with WLAN but even with GSM phone links. Thus, we investigate how different compression methods can be combined to minimize the amount of data transmitted between the client and server.

2. Software Architecture

Figure 2 shows the software architecture of the system. First, the camera in the client captures an RGB image, and the original image is stored away for later use. Once the virtual objects have been rendered on the server, their image is sent to the client and overlaid on top of the original image.

In the server, we use the ARToolkit software to track the markers and to render the virtual objects. ARToolkit internally thresholds the camera image for tracking purposes. To exploit this, we threshold the image already on the client, and send only a bitmap image to the server. 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. On top of this, we apply run length encoding (RLE) to the thresholded image. Each run is encoded with a variable length Elias Gamma code [3] which in practice gives a compression factor of about 5.

The overlay image (just the smallest rectangle area containing the virtual objects) is encoded using Motion Vector Quantization (MVQ), a video coder developed by our research group at VTT [4]. Decoding with MVQ is extremely light, it mainly relies on just lookup tables and motion vectors, which suits well our application on handheld processing units.

Using a color for indicating transparency would be tricky, as colors will be changed and blurred out a bit in video coding. Therefore we transmit a 320x240 binary image as a separate transparency mask, RLE compressed with practical compression ratios now 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.

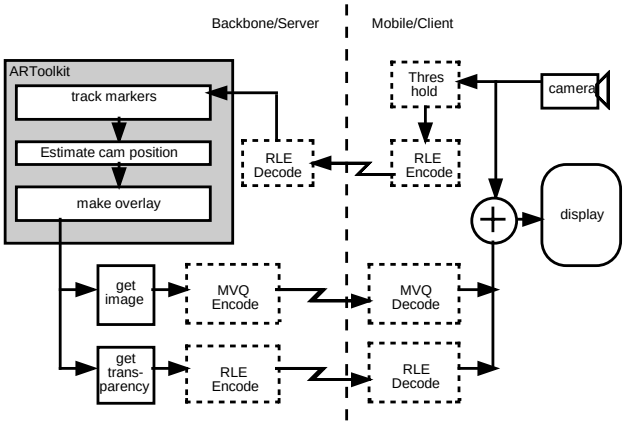

Figure 2. Software architecture. Dashed blocks show optional compression components.

3. Hardware

Our prototype system consists of the following hardware. The client is a Compaq iPAQ H3800 series, a LiveView FlyJacket with LiveView FlyJacket iCAM, and a D-Link DCF-650W WLAN card (see Figure 1). We used two different servers, a 1.8 GHz Pentium 4 with GeForce4 440 Go for WLAN (11Mbit/s), and a 800 MHz Pentium 3 with Matrox Millenium G400 for GSM HSCSD data link (41 kbit/s). When using GSM, the iPAQ communicates via Bluetooth to a Nokia 7650 mobile phone, which in its turn contacts the server.

4. Performance

Table 1 shows some of the measurements we did on our prototype system. When using WLAN, the video image was MVQ compressed to "high quality" 30 kbytes/frame (PSNR 28.2 on average), and with GSM to "moderate quality" 4 kbytes/frame (PSNR 22.8).

We used a 3D model with 60,000 polygons, occupying most of the 320x240 PDA screen. With WLAN, the achieved refresh rate was 800 ms per image. When the virtual object occupied a smaller part of the screen (as in Figure 1), the refresh time went down to some 600 ms.

Using the slow GSM connection, communication overhead takes half of the time alone, and rest of the time is mainly used for transmission of the images. The achieved refresh time then varied between 3 to 5 seconds, depending on the virtual model's size on the screen.

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., [2], are difficult to make due to hardware differences, such as discussed in [5]. Nevertheless, slow transmission channels such as GSM or UMTS would in principle make comparisons favourable to our architecture, where RGB images are transmitted only from server to client.

Table 1. Timing of prototype performance (worst case). All times are in milliseconds.

Step	WLAN / Pentium 4	GSM / Pentium 3
Capture camera image	110	110
Camera image threshold	10	10
Transmission to server	2	400
Tracking + Rendering	150	220
Get overlay & mask	20	110
MVQ compress	200	200
Communication overhead	100	2500
Transmission to client	25	1280
MVQ decompress	50	50
Overlay virtual objects	60	60
Show result on screen	60	60
Total time per refresh	800	5000

5. Acknowledgements

Wouter Pasman was invited to carry out this work in Finland with funding by VTT Information Technology. Many thanks to Mika Hakkarainen and Petri Honkamaa for their instantaneous support on many implementation and for writing the basic communication code for the system. Mr. Ale Torkkel kindly provided the CAD model of the virtual building used in our tests and presentation.

6. References

[1] Matsuoka, H., Onozawa, A., Hosoya, E. (2002). Environment Mapping for Objects in the Real World: A Trial Using ARToolkit. *Proc. First IEEE Intl. Augmented Reality Toolkit Workshop (ART02)*, Darmstadt, Germany, 29 September 2002.

[2] Geiger C., Kleinjohann B., Reimann C., Stichling D. (2001): Mobile AR4All, in *Proc. The Second IEEE and ACM International Symposium on Augmented Reality (ISAR'01)*, New York, October 2001.

[3] Soboroff, I. (2002). Compression for IR: Lecture 5. Available Internet: http://www.csee.umbc.edu/~ian/irF02/lectures/05Compression-for-IR.pdf.

[4] Valli, S. (2002). Video Coding. VTT Information Technology, Finland. Available Internet: http://www.vtt.fi/multimedia/index_vc.html.

[5] Pasman, W. (2002). Speed of FlyJacket Grabber. *Internal Report*, VTT Information Technology, Helsinki. Available internet: http://graphics.tudelft.nl/~wouter.