AN AUGMENTED REALITY INTERFACE FOR MOBILE INFORMATION RETRIEVAL

Jonathan Mooser, Lu Wang, Suya You, and Ulrich Neumann

CGIT Lab, University of Southern California
{mooser, luwang, suyay, uneumann}@graphics.usc.edu

ABSTRACT

Recent years have seen growing interest in mobile augmented reality. The ability to retrieve information and display it as virtual content overlaid on top of an image of the real world is a natural extension to a mobile device equipped with a camera and wireless connectivity. Such applications need to address a number of technical hurdles including target recognition and camera pose estimation. Beyond these fundamental challenges, however, there is the problem of data connectivity and user interface presentation. How do we connect mobile clients to multiple data sources that may be changing in real time and provide the user with a flexible interface for navigating through the relevant content? We present an AR user interface framework specifically designed to expose disparate data sources through a single application server. Our proposed system uses an multi-tier architecture to separate back-end data retrieval from front-end graphical presentation and UI event handling. We describe how it might be used to build an oil platform equipment maintenance system, as an example of a collaborative, data-driven mobile application.

1. INTRODUCTION

Mobile AR is a natural platform for a number of potential "killer apps." Schmalsteig and Wagner, for example, describe an interactive AR museum, where virtual media complements and annotates real-world exhibits [1]. Träskbäack and Haller propose a training application that allows oil refinery employees to view instructional diagrams overlaid on top of the very equipment they are learning to use [2]. Other applications include multiplayer games [3], equipment maintenance [4], and document annotation [5], just to name a few.

These applications all share a few essential features. Each depends on a large collection of dynamic and potentially distributed data, and must maintain associations between recognizable visual targets and relevant information. These associations will inevitably change with the underlying data or as the application grows.

A presentation layer must define how to render data as virtual media. Depending on the nature of the data, it may make sense to render different combinations of text, icons, images, or 3D objects. The precise translation from information to virtual content is application specific.

Multiple users with separate mobile devices should be able to share data and collaborate. This implies the need for a central data store that keeps track of user actions and the overall state of the system.

Taken as a whole, we refer to this collection of features as an AR information retrieval system. While a number of systems have been developed to facilitate AR content creation, our primary contribution is an architecture that connects large sets of visual targets to dynamic, shared data sources.

2. RELATED WORK

The majority of augmented reality literature is devoted to the problems of target recognition and pose estimation. This is not surprising, as a complete knowledge of the visual context as well as the camera's position and orientation are necessary to accurately render virtual media on top of a captured image. Fischler and Bolles[6] describe one of the earliest algorithms for deriving camera pose from a set of point correspondences. When specifically applied to AR, visual targets often consist of artificial landmarks (fiducials), for which a wide array of designs have been proposed [5][7][8][9][10].

Some algorithms make use of natural features of the environment by themselves or in conjunction with artificial fiducials[11]. Others combine gyroscopic or magnetic sensor data with visual landmarks to form hybrid systems[12][13].

To facilitate the development of large AR systems, several programming APIs are available. The most popular is ARToolkit[14], a C++ library that uses fiducials and a training system for recognition. The core functionality of ARToolkit is target recognition and pose estimation. It is the responsibility of the application developer to write rendering code for each identifiable target. For richer, more immersive experiences, there is DART[15], an authoring environment based on Macromedia Director. It allows virtual objects to be associated with "behaviors" that define how to act and respond to user interaction. DART focuses primarily on single user experiences rather that multiuser collaboration.

Moving from single user to multiuser applications requires some form of network infrastructure. A framework for distributed AR is described in[16], which focuses on relieving the computational burden on lightweight mobile devices by transferring recognition and pose computations to a central server. Our system makes use of TriCodes, artificial landmarks that can be detected and identified rapidly even on a handheld device with minimal processing power. We leave the identification and pose computations on the client, the server thus functioning primarily as a central point for data access.

Perhaps the most similar work to our own is a collaborative AR system built on top of the Studierstube framework and its companion APRIL authoring language [17]. The authors demonstrate a multiuser tourist guide for exploring an historical district of Vienna. The primary focus of APRIL is to support a variety of hardware platforms, in particular wearable devices such as head mounted displays. The focus of the present work, by contrast, is seamlessly connect AR content to system wide data.

3. OUR APPROACH

We have implemented our system on a Sony Vaio® UX Micro PC (Fig. 2), which includes a built in camera, a wireless LAN card, a keyboard, and a stylus based touch screen. It is powerful enough to run Microsoft Windows XP, yet small enough to be considered a handheld device, making it highly suitable for our implementation.

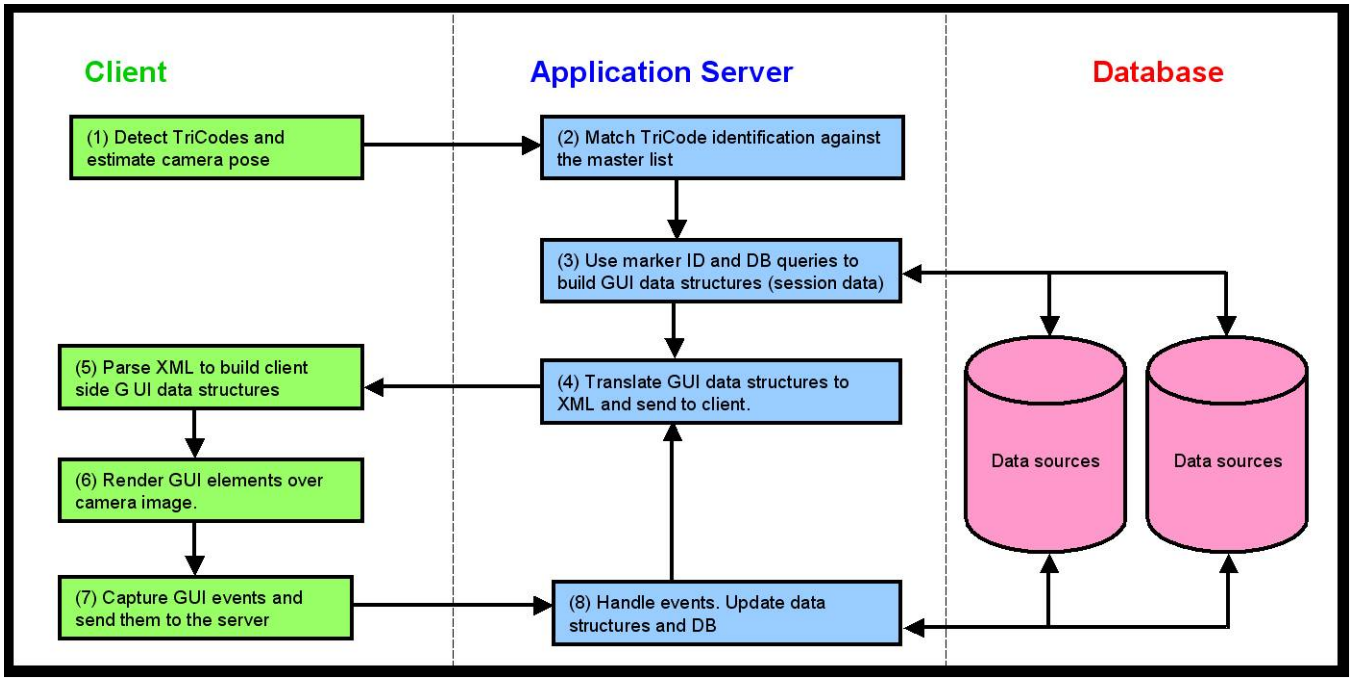

Fig. 1. A modular view of the client-server architecture

Fig. 2. The Sony Vaio® UX Micro PC

For target recognition and pose estimation, we use TriCode fiducials, originally inroduced in [18]. Briefly, TriCodes are visual barcodes composed of 8 triangular elements (Fig. 3). Each triangle can assume any of eight orientations, so the overall pattern corresponds to a 24-bit code. The result is an easily identifiable pattern that can be produced in large quantities without compromising the uniqueness of each individual marker. The recognition and pose estimation can runs a mobile client in real time, with the identity of a newly dected TriCode then passed to the server as a single numerical value.

Once we recognize a target and compute the camera pose, what remains is to determine what context specific information is relevant

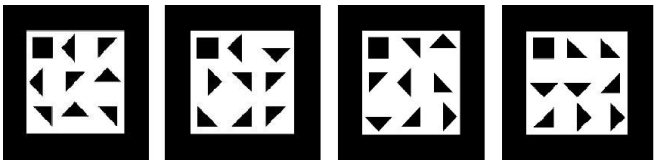

Fig. 3. Examples of tricode fiducials

to each target and how to render that information as virtual media. Taken by itself, this is an open-ended problem. Data can exist in many forms, such as relational databases, sensor readings, or locations of other users. It makes sense to provide a single server that is aware of all system-wide data and knows how to present data as a user interface.

We model this process as a multi-tier client-server architecture (Fig. 1). The back end consists of one or more data sources, which can exist in any form as long as they can be externally queried over a network. The application server, where most of the application specific functionality resides, links recognized targets and user actions to specific data queries, packaging the results as a virtual user interface. The mobile device then opperates as a "thin client," rendering 3D GUI elements on top of its captured image and leaving the core application processing to the server.

To illustrate each component of Fig. 1, we follow the application through a typical sequence of events.

1. On start up, each client opens and maintains a connection to a single application server, which handles all client communication. When the mobile device first detects a marker in its camera's field of view, the client reads the code and translates it into a 24-bit value. At this point, image noise may leave some of the elements misread or unreadable.
2. In any case, the client sends its best guess as to the TriCode

identity to the middleware server, which searches for that value against a master list of all TriCodes in the environment (2). If an exact match is found, the value is assumed to be correct. Otherwise the closest match in the list is used. As described in [18] six bits of each value are reserved for a checksum, ensuring that no two values are too similar, and making it unlikely that a misread code will be mismatched.

3. For every TriCode, the application server stores a function to be executed when it is initially detected. This initialization function may make one or more queries to the back-end data sources in order to collect and organize relevant data. Then it constructs the visual output using a set of pre-defined GUI elements. These graphical building blocks may include text areas, icons, images, buttons, or drop-down menus, each providing customizations such as size and color. The primary task of the developer is to write routines that call queries and convert the results into a data structure of user interface elements. This allows for considerable graphical flexibility without the need to write a separate low level rendering routine for each fiducial.
4. Once the user interface structures have been built, the application server serializes them into a simple XML stream, which is sent back to the mobile client. A complete description of our XML format is beyond the scope of this paper, but at a high level it contains a list of elements with a name, type, location, and attributes that define type-specific customizations.
5. Having received the XML stream, the client builds its own local version of the user interface structure.
6. Meanwhile, the four corners of the originally detected marker are used to estimate camera pose with respect to the marker coordinate frame. Because the locations of graphical elements are always specified with respect to the marker coordinate frame as well, the client now has enough information to render GUI elements aligned with its camera image.
7. The user now sees a 3D user interface consisting of static elements such as text labels and images as well as actionable elements such as buttons and text boxes. As the user interacts with the interface, certain actions, such as a button press, will trigger a "submit" event. When this happens, the client sends an XML stream that reflects which button was just pressed along with any other changes the user may have made, such as menu selections and text entry.
8. The server parses this stream and calls an event handler routine that performs the specific updates and queries appropriate for the event. Just as when the marker was first detected, the server sends an XML stream back to the mobile client that reflects any changes to the visual content.

4. EXAMPLE: EQUIPMENT MAINTENANCE

The current system is being developed for use on an oil platform, an environment with an extensive array of equipment requiring specialized maintenance and repairs. Normally, an engineer would assess a problem then return to a central location to consult the relevant manuals or data sheets. After making repairs or running tests, she would again return to the central location to record any updates. Our goal is to streamline this process by performing all information retreival and data updates on site. The mobile device would access data applicable to a specific piece of equipment and render it as AR content. Not only does this eliminate the need to leave the worksite to access information, but allows the information to be displayed on top of the physical environment while the engineer is working on it. Any change would be sent in real time through the application server to a central data store.

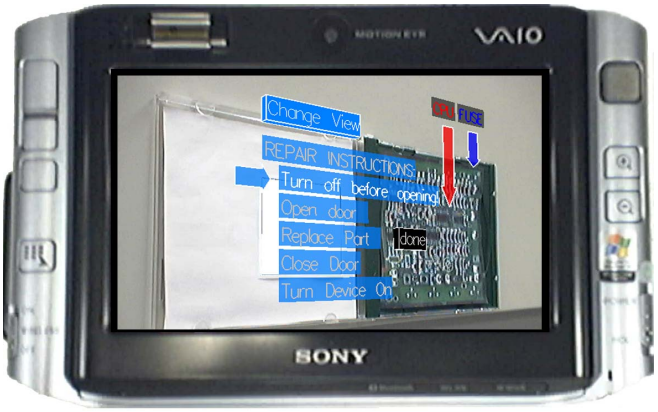

Fig. 4. A step-by-step repair guide

Fig. 5. Up to date repair log that keeps track of completed tasks

Fig. 4 shows a simulation of a circuit repair task, along with virtual arrows indicating the appropriate parts. Besides retrieving a list of repair steps from the database, the application server keeps track of the current state of the repair, and highlights the current step. By touching the "Change View" button, the user can access different views of dynamic data. For example, as tasks are completed, the application server updates a log in the database, to be retrieved as needed Fig. 5. Another engineer examining this equipment would immediately know which tasks have already been completed. Fig. 6 shows a schematic diagram image retrieved from the application server. The image is overlaid on top of the actual circuit to assist in the repair process.

Fig. 7 shows an example of a virtual discussion board. Each area of the environment might have a marker for users to record general purpose notes. As with the repair log, messages are stored centrally so that multiple users can access each other's additions at any time.

5. CONCLUSION AND FUTURE WORK

We have presented a flexible architecture for creating large scale AR systems. Besides a set of general-purpose components for creating

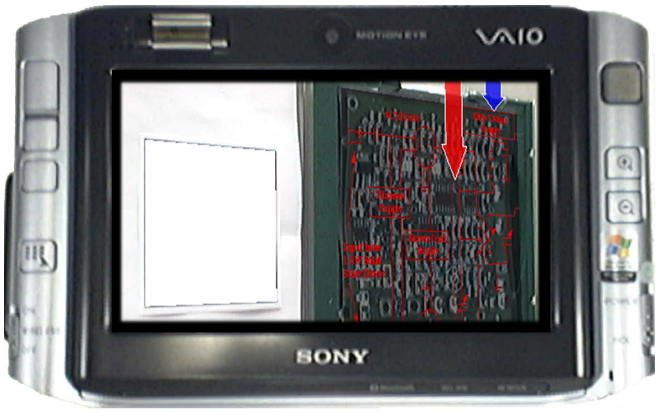

Fig. 6. A schematic diagram superimposed over the circuit

Fig. 7. A virtual discussion board where users can post messages

AR user interfaces, we have shown how those components can be used in the context of a data-driven application.

The greatest limitation of the current system is the need to place fiducial markers throughout the environment. Future versions may include the ability to learn to recognize natural features. While this would mean significant changes to the pattern recognition and pose computation components, the overall architecture would remain largely intact. The application server, rather than maintaining a master list of fiducial codes, would maintain a master list of trained natural features and associated code.

Another future direction is the creation of a graphical authoring tool to facilitate the development process. Rather than manually scripting the positions and sizes of user interface elements, A drag-and-drop interface might allows a users to create objects and position them interactively. These and other improvements should further our system's applicability to real world applications.

6. ACKNOWLEGEMENTS

This study was funded by the Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft); CiSoft is a joint University of Southern California-Chevron initiative.

This work made use of Integrated Media Systems Center Shared Facilities supported by the National Science Foundation under Cooperative Agreement No. EEC-9529152. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation

7. REFERENCES

[1] D. Schmalstieg and D. Wagner, "A handheld augmented reality museum guide," in *IADIS Mobile Learning 2005*, June 2005.

[2] M. Träskbäack and M. Haller, "Mixed reality training application for an oil refinery: user requirements," in *ACM SIGGRAPH VRCAI*, 2004, pp. 324 – 327.

[3] D. Wagner, T. Pintaric, F. Ledermann, and D. Schmalstieg, "Towards massively multi-user augmented reality on handheld devices," in *Pervasive 2005*, May 2005.

[4] S. Feiner, B. Macintyre, and D. Seligmann, "Knowledge-based augmented reality," *Commun. ACM*, vol. 36, no. 7, pp. 53–62, 1993.

[5] J. Rekimoto and Y. Ayatsuka, "CyberCode: designing augmented reality environments with visual tags," in *Designing Augmented Reality Environments*. 2000, pp. 1–10, ACM Press.

[6] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381 – 395, 1981.

[7] Y. Cho and U. Neumann, "Multi-ring fiducial systems for scalable fiducial-tracking augmented reality," *PRESENCE: Teleoperators and Virtual Environments*, vol. 10, no. 6, pp. 599 – 612, Dec 2001.

[8] F. Ababsa and M. Mallem, "Robust camera pose estimation using 2d fiducials tracking for real-time augmented reality systems," in *ACM SIGGRAPH VRCAI*, 2004, pp. 431–435.

[9] D. López de Ipiña, P. R. S. Mendonça, and A. Hopper, "TRIP: A low-cost vision-based location system for ubiquitous computing," *Personal Ubiquitous Computing*, vol. 6, no. 3, pp. 206 – 219, 2002.

[10] D. Claus and A. W. Fitzgibbon, "Reliable fiducial detection in natural scenes," in *ECCV 2004*, 2004, pp. 469 – 480.

[11] B. Jiang, S. You, , and U. Neumann, "Camera tracking for aumented reality media," in *ICME*, Jul 2000, pp. 1637 – 1640.

[12] A. State, G. Hirota, D. T. Chen, W. F. Garrett, and M. A. Livingston, "Superior augmented reality registration by integrating landmark tracking and magnetic tracking," in *ACM SIGGRAPH*, 1996, pp. 429 – 438.

[13] B. Jiang, S. You, , and U. Neumann, "A robust hybrid tracking system for outdoor augmented reality," in *IEEE Virtual Reality 2004*, Mar 2004, pp. 3 – 10.

[14] H. Kato, "http://www.hitl.washington.edu/artoolkit," 2007.

[15] B. MacIntyre, M. Gandy, S. Dow, and J. D. Bolter, "DART: a toolkit for rapid design exploration of augmented reality experiences," in *UIST*, 2004, pp. 197 – 206.

[16] J. Evers-Senne, A. Petersen, I. Schiller, and R. Koch, "A mobile augmented reality system with distributed tracking," in *3DPVT*, June 2006.

[17] G. Reitmayr and D. Schmalstieg, "Collaborative augmented reality for outdoor navigation and information browsing," in *2nd Symposium on Location Based Services and TeleCartography*, Janurary 2004.

[18] J. Mooser, S. You, and U. Neumann, "Tricodes: A barcode-like fiducial design for augmented reality media," in *ICME 2006*, July 2006, pp. 1300 – 1304.