The MORGAN Framework: Enabling Dynamic Multi–User AR and VR Projects

Jan Ohlenburg, Iris Herbst, Irma Lindt, Thorsten Fröhlich, Wolfgang Broll
{jan.ohlenburg, iris.herbst, irma.lindt, thorsten.froehlich, wolfgang.broll}@fit.fraunhofer.de
Fraunhofer Institute for Applied Information Technology (FIT)
Schloss Birlinghoven
Sankt Augustin, Germany

ABSTRACT

The availability of a suitable framework is of vital importance for the development of Augmented Reality (AR) and Virtual Reality (VR) projects. While features such as scalability, platform independence, support of multiple users, distribution of components, and an efficient and sophisticated rendering are the key requirements of current and future applications, existing frameworks often address these issues only partially. In our paper we present MORGAN – an extensible component–based AR/VR framework, enabling sophisticated dynamic multi–user AR and VR projects. Core components include the MORGAN API, providing developers access to various input devices, including common tracking devices, as well as a modular render engine concept, allowing us to provide native support for individual scene graph concepts. The MORGAN framework has already been successfully deployed in several national and international research and development projects.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed applications;
H.5.1 [**Multimedia Information Systems**]: Artificial, augmented and virtual realities;
I.3.2 [**Graphics Systems**]: Distributed/network graphics;
I.3.6 [**Methodology and Techniques**]: Device independence, Graphics data structures and data types, Interaction techniques;
I.3.7 [**Three–Dimensional Graphics and Realism**]: Virtual reality

General Terms

Performance, Design, Reliability

Keywords

Augmented reality, virtual reality, framework, render engine, distributed system design, tracking

VRST'04, November 10-12, 2004, Hong Kong.

1. INTRODUCTION

The lack of efficient frameworks including tailored render engines is still a major drawback for the efficient development of AR and VR applications. Since no standard for such projects has evolved yet, a large number of frameworks and render engines – often designed for very specific application needs – exist. Our paper will present a combined solution, which enables sophisticated dynamic multi–user AR and VR projects. Using well–known software patterns and the CORBA middleware, the MORGAN framework allows for easy access to external devices such as trackers, while hiding the details of the distributed system. Application developers are provided with a high level interface – the MORGAN API – giving the developer full access to the scene graph stored in the render engine, trackers and other external devices, while being fully configurable for the application needs. Since the focus during the design of the framework and the render engine had been on extensibility, flexibility and multi–user aspects, including scalability, the developer does not need to care about the distribution or the synchronization of data. The framework already integrates several devices and third–party software, e.g. head tracking devices, computer vision devices and speech recognition software. It is able to abstract from these input devices and their locations, i.e. applications can be easily configured using different head tracking systems running on different hosts, without changing the application logic.

We designed the MORGAN render engine to be independent of a specific file format, such as VRML97. The internal scene graph held by the render engine contains only data required for rendering. All application specific data is stored in external scene graphs (XSG). These XSGs associate further information to the nodes of the internal representation for their specific needs. For instance a VRML97 XSG maps its rendering information onto the internal scene graph and stores the VRML97 specific data, e.g. a `WorldInfo` node, itself. This approach allows the combination of different external representations, and keeps the internal design lean.

In the next section we will provide an overview of existing frameworks and render engines, before we will introduce the MORGAN framework in Section 3. In Section 4 we will outline previous projects realized utilizing the MORGAN framework, before we will conclude and provide an outlook on our future work in the final section.

2. RELATED WORK

In this section we will review a couple of recent devel-

opments in the area of AR/VR related frameworks, which influenced our work and our decision to develop the MORGAN framework. For a more general quite comprehensive overview on existing AR and VR frameworks please see Bierbaum and Just [2].

DWARF [1] is a component–based AR framework. Its main concept is to provide core functionality for AR systems by collaborating distributed services. Each service describes its abilities, needs and connectors in XML. Service managers running at each system site connect the distributed services according to their needs and abilities. CORBA is used for communication setup, but services may deploy more efficient protocols. Developers of an AR system can easily use or extend existing services or develop new ones. Several projects, e.g. SHEEP [10] or FataMorgan [8], demonstrate the power of DWARF. However, the main focus of DWARF is the application of software engineering theory to the domain of AR [11].

ARToolKit [7] is an open source computer vision tracking library that has been widely used to create AR applications. It provides computer vision techniques to calculate a camera's position and orientation relative to certain square patterns. Live video images are captured, transformed into binary images, and square regions are identified and compared with predefined pattern templates. The result is a 3x4 matrix that contains the video camera real world coordinates relative to the square pattern. A predefined square pattern can be associated with a VRML file or an OpenGL–based application, to overlay geometry on a recognized marker.

OpenSceneGraph[1] is one of the leading open source scene graphs and one of the most often used VR toolkits; this is due to its portability and its high performance in graphics applications. A wide variety of applications and research institutes already utilize this framework. One of the reasons for the wide acceptance of OpenSceneGraph is its high performance in displaying highly detailed 3D scenes. This is achieved by optimizing the OpenGL state changes and restricting the update opportunities of the application. The latter one requires to handle multi–source updates as well as distribution aspects such as consistency on its own.

OpenSG [12] is another scene graph API for VR projects. Since it focuses only on being a single user scene graph, all limitations of OpenSceneGraph apply to it as well. The application is responsible for distributing changes and keeping a consistent state of the local instances of OpenSG. An important difference to OpenSceneGraph is that OpenSG is especially designed to be used by several updating threads. Each thread maintains a local copy of the data until it is synchronized with other threads.

The distribution aspects of OpenSceneGraph and OpenSG are more focused on clustering than on distributed applications. In a cluster there is one sender doing the distribution and a number of receivers. In distributed applications changes can be made from any host.

3. THE MORGAN FRAMEWORK

The MORGAN framework is a component–based framework for dynamic multi–user AR and VR projects. It utilizes well–known software design patterns [5, 6] to be easily understandable and extendable. For the same reason, we rely on the CORBA middleware, which allows the framework to be extended by external devices and accessed by external applications. The MORGAN framework consists of a set of components enabling application developers to deploy several input devices, such as tracking devices, or speech input, while providing a sophisticated render engine. The MORGAN render engine plays a decisive role within the MORGAN framework, since it is especially designed for supporting multiple users within a distributed framework. (see Section 3.2).

3.1 Basic Components and Devices

The central component of the MORGAN framework is the `Broker`. It provides a convenient interface for creating, deleting and retrieving other components of the framework. All components are derived from a base component to assure a common interface.

A number of devices have already been implemented and integrated into the MORGAN framework. These devices are realized using a *publisher–subscriber* pattern. Each device is a publisher, providing new samples at a certain rate. Components, which are interested in these samples, subscribe for this service. Some of the devices, which have been integrated using this pattern, are the InterSense tracking systems (IS–600, IS–900 and InertiaCube2) as well as three types of computer–vision based object tracking (including ARToolkit [7]) and a speech input device based on IBM's ViaVoice. The support of different tracker devices is completed by components for recording and monitoring the tracking data, and for tracker data fusion.

The *factory method* pattern is used to implement remote component creation and instantiation. This is important if components can only run on a designated computer, e.g. since a particular device is connected to it. The factory class of that component is responsible for the instantiation of the component and informs the Broker about the existance of this component.

Another software design patterns used, is the *proxy* pattern. The *proxy* pattern is used to wrap non–CORBA based communication protocols or as a defined interface for external software, e.g. to connect TCP/IP based services.

The collaboration and the distribution of components including the MORGAN render engine rounds up the collaboration between components within the MORGAN framework. Since the framework is able to hide the distribution of components, no limitations are put on the possible architecture of applications.

3.2 The MORGAN Render Engine

In contrast to most other frameworks MORGAN comes along with its own designated render engine, particularly designed to meet the requirements of distributed multi–user VR and AR applications. In our approach a consistent state of the scene graph is guaranteed, even if several sources manipulate its nodes concurrently, which may happen at any time. This is realized by separating an update thread, which actually applies the changes to the rendering, from the actual render threads.

Thus, in contrast to for instance *OpenSceneGraph* or *OpenSG*, an application may apply modifications at any time except when the update thread is actually updating that particular part of the scene graph. As changes to the structure of the scene graph are made, e.g. when adding, removing or referencing nodes, in almost all cases an update

[1]http://www.openscenegraph.org

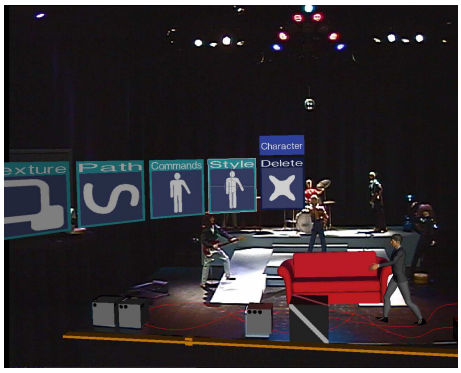

Figure 1: The Mixed Reality Stage.

would be required. In order to guarantee a well–defined state of the scene graph all the time, `Group` nodes, which form the structure of the scene graph, collect all structural changes and apply them as the first operation during the next update.

A major difference between our approach and existing scene graphs is the separation of render data from other scene graph data into separate scene graphs. Thus the render scene graph stores render information only, while all other scene graph related information is stored in external scene graphs (XSG). In order to do this, the scene graph manages a unique identification number for each node. XSGs and external applications may use this identifier to associate additional information with these objects. The concept of XSGs complements the compact design of the scene graph, since other scene graph designs, file formats and rendering interfaces can easily be mapped onto the native scene graph. This provides a universal and extendable approach for the integration of several types of scene graphs (e.g. VRML97, X3D, or CAD data).

The MORGAN framework allows all these features to coexist in a single scene graph, with each being represented in sub–scene graphs. Other XSGs do not need to be aware of these sub–scene graphs, but if they are, the application developer has the flexibility to define all the interoperations needed by either using interfaces to multiple XSGs at the same time, or by restricting the application to the manipulation of the render scene graph.

Another important feature of the scene graph in a multi–user environment is its built–in ability to distribute its data to all users. The database–like unique identification introduced above is essential in enabling a simple and quick distribution of changes. All distributed scene graphs share those identifiers. In order to reduce overall communication load, identifier ranges are alloted to each local instance of a distributed scene graph.

3.3 The MORGAN API

The MORGAN API builds up on the MORGAN framework and is an easy to use C++ library for AR and VR projects. It abstracts from framework design issues and reduces the complexity for application developers. The API is especially designed for rapid prototyping, since it provides abstract classes to access devices, third–party software and different scene graphs. Therefore, the MORGAN API allows modifying the current scene and includes methods for accessing input from external devices and tools.

The API provides methods to access scene graph nodes directly through the render engine or through an XSG. Nodes can be inserted, deleted, referenced, replaced, moved and copied. A node usually has different representations in the render engine and in the XSG, the API allows full access to both representations.

The MORGAN API also provides easy access to picking results and collision detection results. Picking is performed by sending a request to the render engine or the XSG. For collision detection, collision interests are defined. Objects are registered for collision detection by defining individual objects or sub–scene graphs.

4. CASE STUDIES

The MORGAN framework has already been used in several research projects proving the suitability of its concepts and components. In this section two of these projects are outlined.

4.1 The Mixed Reality Stage

The Mixed Reality Stage (MRS), developed within the mqube project [3], provides an AR environment for collaborative planning of stage events such as theater performances, product presentations or concerts. The planning of an event involves several professionals, such as light and stage designers, and technical directors. These people have to share ideas from the very beginning. It is common practice to use real downscaled stage models for sampling, including model lights, props and jointed dolls representing the actors. Such a model allows for testing different lighting and prop arrangements on the stage. However, it is does not support modifying the appearance of the props or animating a character representing an actor on the stage.

The MRS overcomes these disadvantages by providing a collaborative AR environment supporting face–to–face collaboration. The real model stage is extended by superimposed virtual props and virtual interface elements (see Figure 1). A hybrid collaborative user interface was developed by employing real control desks for stage machinery and light controls in addition to tangible user interfaces, viewpoint based interactions and voice commands.

The MORGAN framework has been extended by several new components including a stage machinery control component, a light control component and a speech component. Additionally, two new tracking systems developed within this project were integrated into the MORGAN framework: BlueTrak [9], a novel wireless inertial tracking system, used for head tracking, and an optical tracking system used to realize the tangible user interfaces. Based on the existing tracker interface, support of these trackers required only a small amount of work. The same applies to the corresponding extensions of the MORGAN API.

The MRS application logic has been realized on top of the MORGAN API. The multi–user features of the framework and the MORGAN API supported the design and the realization of the application. The ability to declare parts of the scene graph as private made it possible to have private virtual menus and states.

Figure 2: The ARTHUR System

4.2 ARTHUR

The ARTHUR project (Augmented Round Table for Architecture and Urban Planning) [4] is an interdisciplinary research project between technology developers and end users (architects). Architectural design and urban planning are highly cooperative tasks. Several iterations within the design process are needed to complete a project, each iteration consists of close cooperative situations for instance during design and review meetings, and individual work by the participants or third parties. The problems and solutions discussed during the meeting are then integrated into the design during the subsequent individual work phases. Real collaboration is therefore very limited and final decisions are often postponed until the next meeting.

The ARTHUR system (see Figure 2) focuses on enabling natural interaction with virtual 3D objects. Three types of intuitive input mechanisms are used for user interaction, real placeholder objects are used tangible interfaces, wands are used for 5DOF pointing, and hand gestures and fingertip tracking may be used to draw lines in space, navigate in menus, select objects or execute actions.

Utilizing the MORGAN framework, the ARTHUR system allows several users to share a virtual space projected into their common working environment. The participants see and interact with the same virtual objects, while personal menus and individual information may be provided to each user Changes made to (shared) virtual objects are distributed by the framework and are immediately visible to all other users.

The capabilities of the MORGAN framework are exploited by the integration of external application, e.g. a commercial CAD software as well as a pedestrian simulation program. The application logic was developed using the MORGAN API and the input devices and gestures are integrated as trackers. Therefore, the application is independent from particular input devices. Input devices may be replaced by simply adjusting the configuration script.

5. CONCLUSIONS AND FUTURE WORK

In this paper we presented the MORGAN framework. We introduced its basic concepts and services as well as two of its major components: the designated scene graph concept, distinguishing between render information and external scene data, and the programming API. We showed how this integrated approach is superior to existing stand–alone solutions and outlined the successful use of the framework within previous projects.

In our future work we will investigate into the concept of making framework components replicable and self–replicable. Thus replicated copies of components may be used for load balancing to increase the overall system performance.

6. ACKNOWLEDGMENTS

The ARTHUR project is partially funded by the European Commission as part of the IST programme within the 5th framework (project no. IST–2000–28559). The mqube project was partially funded by the German Federal Ministry of Education and Research. We would like to thank the partners of both projects for their contributions and fruitful cooperation.

7. REFERENCES

[1] M. Bauer, B. Bruegge, G. Klinker, A. MacWilliams, T. Reichert, S. Riß, C. Sandor, and M. Wagner. Design of a Component–Based Augmented Reality Framework. In *Proceedings of ISAR 2001*, 2001.

[2] A. Bierbaum and C. Just. Software Tools for Virtual Reality Application Development. In *SIGGRAPH 1998 Applied Virtual Reality course notes*, 1998.

[3] W. Broll, S. Grünvogel, I. Herbst, I. Lindt, M. Maercker, J. Ohlenburg, and M. Wittkämper. Interactive Props and Choreography Planning with the Mixed Reality Stage. In *Proceedings of ICEC 2004*, 2004. to be published.

[4] W. Broll, I. Lindt, J. Ohlenburg, M. Wittkämper, C. Yuan, T. Novotny, C. Mottram, A. Fatah, and A. Strothmann. ARTHUR: A Collaborative Augmented Environment for Architectural Design and Urban Planning. In *Proceedings of HC 2004*, 2004. to be published.

[5] F. Buschmann, R. Meunier, H. Rohnert, and P. Sommerlad. *Pattern–Oriented Software Architecture. A System of Patterns.* John–Wiley & Sons, 1996.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object–Oriented Software.* Addison–Wesley, 1995.

[7] H. Kato, M. Billinghurst, B. Blanding, and R. May. ARToolKit. Technical report, Hiroshima City University, December 1999.

[8] G. Klinker, A. Dutoit, M. Bauer, J. Bayer, V. Novak, and D. Matzke. FataMorgan – A Presentation System for Product Design. In *Proceedings of ISMAR 2002*, 2002.

[9] H. Krüger, L. Klingbeil, E. Kraft, and R. Hamburger. BlueTrak: A wireless six degrees of freedom tracker motion tracking system. In *Proceedings of ISMAR 2003*, 2003.

[10] A. MacWilliams, C. Sandor, M. Wagner, G. Klinker, and B. Bruegge. Herding Sheep: Live System Development for Distributed Augmented Reality. In *Proceedings of ISMAR 2003*, 2003.

[11] T. Reichert, A. MacWilliams, and B. Bruegge. Towards a System of Patterns for Augmented Reality Systems. In *International Workshop on Software Technology for Augmented Reality Systems*, 2003.

[12] D. Reiners, G. Voß, and J. Behr. OpenSG: Basic Concepts. In *1. OpenSG Symposium OpenSG 2002*, 2002.